
An Intelligent Semi-Automatic Workflow
for Optical Character Recognition
of Historical Printings

vorgelegt von

Christian Reul

Würzburg, 2020



Dissertation zur Erlangung des naturwissenschaftlichen Doktorgrades
der Bayerischen Julius-Maximilians-Universität Würzburg

Abstract

Optical Character Recognition (OCR) on historical printings is a challenging task mainly due to the complexity of the layout and the highly variant typography. Nevertheless, in the last few years great progress has been made in the area of historical OCR resulting in several powerful open-source tools for preprocessing, layout analysis and segmentation, Automatic Text Recognition (ATR) and postcorrection. Their major drawback is that they only offer limited applicability by non-technical users like humanist scholars, in particular when it comes to the combined use of several tools in a workflow. Furthermore, depending on the material, these tools are usually not able to fully automatically achieve sufficiently low error rates, let alone perfect results, creating a demand for an interactive postcorrection functionality which, however, is generally not incorporated.

This thesis addresses these issues by presenting an open-source OCR software called *OCR4all* which combines state-of-the-art OCR components and continuous model training into a comprehensive workflow. While a variety of materials can already be processed fully automatically, books with more complex layouts require manual intervention by the users. This is mostly due to the fact that the required Ground Truth (GT) for training stronger mixed models (for segmentation as well as text recognition) is not available, yet, neither in the desired quantity nor quality.

To deal with this issue in the short run, OCR4all offers better recognition capabilities in combination with a very comfortable Graphical User Interface (GUI) that allows error corrections not only in the final output, but already in early stages to minimize error propagation. In the long run this constant manual correction produces large quantities of valuable, high quality training material which can be used to improve fully automatic approaches. Further on, extensive configuration capabilities are provided to set the degree of automation of the workflow and to make adaptations to the carefully selected default parameters for specific printings, if necessary. The architecture of OCR4all allows for an easy integration (or substitution) of newly developed tools for its main components by supporting standardized interfaces like PageXML, thus aiming at continual higher automation for historical printings.

In addition to OCR4all, several methodical extensions in the form of accuracy improving techniques for training and recognition are presented. Most notably an effective, sophisticated, and adaptable voting methodology using a single ATR engine, a pretraining procedure, and an Active Learning (AL) component are proposed. Experiments showed that combining pretraining and voting significantly improves the effectiveness of book-specific training, reducing the obtained Character Error Rates (CERs) by more than 50%.

The proposed extensions were further evaluated during two real world case studies: First, the voting and pretraining techniques are transferred to the task of constructing so-called *mixed models* which are trained on a variety of different fonts. This was done by using 19th century Fraktur script as an example, resulting in a considerable improvement over a variety of existing open-source and commercial engines and models. Second, the extension from ATR on raw text to the adjacent topic of typography recognition was successfully addressed by thoroughly indexing a historical lexicon that heavily relies on different font types in order to encode its complex semantic structure.

During the main experiments on very complex early printed books even users with minimal or no experience were able to not only comfortably deal with the challenges presented by the complex layout, but also to recognize the text with manageable effort and great quality, achieving excellent CERs below 0.5%. Furthermore, the fully automated application on 19th century novels showed that OCR4all (average CER of 0.85%) can considerably outperform the commercial state-of-the-art tool ABBYY Finereader (5.3%) on moderate layouts if suitably pretrained mixed ATR models are available.

Contents

1	Introduction	1
1.1	Challenges of Historical OCR	2
1.2	Steps of a Typical OCR Workflow	4
1.3	Challenges for the Users	5
1.4	Proposed Workflow Solution: OCR4all	6
1.5	Contributions	7
1.6	Structure of this Work	8
2	Background and Related Work	11
2.1	Data Sets	12
2.2	Preprocessing	13
2.2.1	Sub Steps	13
2.2.1.1	Binarization	13
2.2.1.2	Skew and Orientation Detection	15
2.2.1.3	Page Frame Detection and Noise Removal	16
2.2.2	Tools	19
2.2.2.1	ScanTailor	19
2.2.2.2	Unpaper	20
2.2.2.3	Implementation in OCR Workflow Tools	20
2.3	Segmentation	21
2.3.1	Region Segmentation	21
2.3.1.1	Fundamentals	21
2.3.1.2	Basic Approaches	24
2.3.1.3	State-of-the-Art	26
2.3.1.4	Tools	28
2.3.2	Line Segmentation	30
2.3.2.1	Fundamentals	30
2.3.2.2	Basic Approaches	33
2.3.2.3	State-of-the-Art	34
2.3.2.4	Implementation in OCR Workflow Tools	35
2.4	Automatic Text Recognition	36
2.4.1	Historical Development and State-of-the-Art	36
2.4.1.1	Glyph-based Recognition	36
2.4.1.2	Line-based Recognition using LSTM Networks	36
2.4.1.3	Line-based Recognition using CNN/LSTM-Hybrid Networks	37
2.4.1.4	Calamari	37

Contents

2.4.2	Evaluation of ATR Engines	40
2.4.2.1	Book-specific Training on Early Printed Books	40
2.4.2.2	Modern English and 19 th Century Fraktur	41
2.4.2.3	Conclusion	41
2.4.3	Mixed Models	42
2.4.3.1	Focus on Early Printed Books	44
2.4.3.2	Modern English and 19 th Century Fraktur	44
2.4.3.3	Multilingual Mixed Models	45
2.5	Postcorrection	45
2.5.1	Overview	45
2.5.2	State-of-the-Art	46
2.5.3	Implementation in OCR Workflow Tools	48
2.6	Tools and Projects Providing an OCR Workflow	49
2.6.1	ABBYY	50
2.6.2	Tesseract	51
2.6.3	OCROPUS	51
2.6.4	Kraken	52
2.6.5	Transkribus	53
2.6.6	DIVA Services	54
2.6.7	OCR-D	54
2.6.8	Nashi	55
2.7	Data Standards and Formats	56
2.7.1	PAGE	56
2.7.2	ALTO	57
2.7.3	hOCR	57
2.7.4	Converters	58
3	Data and Resources	59
3.1	Complete Books for the Evaluation of the Entire Workflow	60
3.1.1	Early Printed Books	60
3.1.1.1	The Ship of Fools	60
3.1.1.2	Camerarius	62
3.1.1.3	Miscellaneous	63
3.1.2	Corpus of 19 th Century Fraktur Novels	63
3.1.3	Historical Lexicon: Sanders' <i>Wörterbuch der Deutschen Sprache</i>	65
3.2	Line-based Ground Truth for the Evaluation of Individual ATR Tasks	66
3.2.1	GT4HistOCR Corpus	66
3.2.1.1	Reference Corpus Early New High German	67
3.2.1.2	Kallimachos Corpus	68
3.2.1.3	Early Modern Latin Corpus	69
3.2.1.4	RIDGES Corpus	71
3.2.1.5	DTA19 Corpus	71

3.2.2	Further 19 th Century Fraktur Data	73
3.2.2.1	Archiscribe Corpus	73
3.2.2.2	JZE Corpus	76
3.2.2.3	Material from the CIS OCR Testset	76
3.2.2.4	Newspaper Daheim	76
3.2.3	Modern Antiqua Data: The University of Washington 3 Database	76
3.3	Mixed Models	77
3.3.1	OCRopus 1	77
3.3.2	Calamari	77
4	Methods	79
4.1	Cross Fold Training and Confidence Voting	80
4.1.1	Related Work	80
4.1.2	Materials and Methods	82
4.1.2.1	Data	82
4.1.2.2	Workflow	83
4.1.2.3	GT Allocation and Model Training	84
4.1.2.4	Alignment	84
4.1.2.5	Additional Information about the Recognition Confidence Values	85
4.1.2.6	Confidence Voting	86
4.1.3	Experiments	87
4.1.3.1	Default Application (5 folds, 150 lines)	87
4.1.3.2	Influence of the Number of Lines	88
4.1.3.3	Influence of the Number of Folds	88
4.1.3.4	Time Expenditure	91
4.1.4	Discussion	91
4.1.5	Conclusion and Future Work	92
4.2	Transfer Learning Utilizing Pretrained ATR Models	93
4.2.1	Related Work	94
4.2.2	Materials and Methods	94
4.2.2.1	Books	95
4.2.2.2	Mixed Models	96
4.2.2.3	Utilizing Arbitrary Pretrained Models in OCRopus 1	96
4.2.2.4	Extending the Codec	96
4.2.2.5	Reducing the Codec	97
4.2.2.6	Defining a Whitelist Containing Immune Characters	97
4.2.3	Experiments	98
4.2.3.1	Setup and Methodology	98
4.2.3.2	Building from the Latin Mixed Model	98
4.2.3.3	Utilizing the OCRopus 1 Standard Models	100
4.2.3.4	Varying the Number of Lines	100
4.2.3.5	Incorporating Individual Models	100

Contents

4.2.4	Discussion	102
4.2.5	Conclusion and Future Work	103
4.3	Combining Pretraining, Voting, and Active Learning	104
4.3.1	Related Work	105
4.3.1.1	Voting	105
4.3.1.2	Pretraining and Transfer Learning	105
4.3.1.3	Active Learning	106
4.3.2	Materials and Methods	107
4.3.2.1	Books	107
4.3.2.2	Active Learning	107
4.3.3	Experiments	109
4.3.3.1	Combining Pretraining and Voting	109
4.3.3.2	Incorporating Active Learning	113
4.3.4	Discussion	113
4.3.5	Conclusion and Future Work	116
5	OCR4all	119
5.1	Goals, General Idea, and Historical Development	119
5.2	Data Structure	121
5.3	Workflow	123
5.4	Preprocessing	124
5.4.1	Image Preparation	124
5.4.2	Binarization	124
5.4.3	Deskewing	127
5.5	Segmentation	128
5.5.1	Region Segmentation	128
5.5.1.1	LAREX	129
5.5.1.2	Dummy Segmentation	134
5.5.2	Line Segmentation	135
5.6	Automatic Text Recognition	139
5.6.1	Character Recognition	139
5.6.2	Model Training	140
5.6.3	Error Analysis	144
5.7	Result Generation	144
5.8	Web Application	145
5.8.1	General Software Design	145
5.8.2	Manual Corrections	146
5.8.2.1	Regions	146
5.8.2.2	Lines	146
5.8.2.3	Text	147
5.8.2.4	Practical Integration into the Workflow	149
5.8.3	Configurations	151
5.8.3.1	Process Flow	151

5.8.3.2	Settings	151
6	Evaluations	153
6.1	Data	153
6.1.1	Early Printed Books	153
6.1.2	Fully Automated Processing of 19 th Century German Novels printed in Fraktur	154
6.2	Precise Segmentation and Trained ATR of Early Printed Books	156
6.2.1	General Processing Approach	156
6.2.2	Overall Time Expenditure and ATR Accuracy	156
6.2.2.1	Results	157
6.2.2.2	Interpretations	157
6.2.3	Evaluating the Iterative Training Approach	160
6.2.3.1	Results	160
6.2.3.2	Interpretations	160
6.2.4	Segmentation Without Semantic Classification	165
6.2.4.1	Results	166
6.2.4.2	Interpretations	168
6.3	Fully Automated Processing	168
6.3.1	19 th Century Fraktur Novels	169
6.3.1.1	Results	169
6.3.1.2	Interpretations	169
6.3.2	Early Printed Books	171
7	Case Studies	173
7.1	Application of Pretraining and Voting to Mixed Models	173
7.1.1	Related Work	174
7.1.2	Data	174
7.1.2.1	Training Data	174
7.1.2.2	Evaluation Data	174
7.1.2.3	Transcription Guidelines and Resulting Codec	175
7.1.3	Training Procedure	177
7.1.3.1	Overview and Goals	177
7.1.3.2	Preprocessing	178
7.1.3.3	Pretraining	178
7.1.3.4	Adding Synthetic Data	179
7.1.3.5	Adding Real Fraktur Data	180
7.1.4	Experiments	180
7.1.4.1	Engines, Models, and Data	180
7.1.4.2	Evaluating the Trained Models	181
7.1.4.3	Influence of the Training Steps	182
7.1.4.4	Comparison with other Engines and Models	184
7.1.5	Discussion	186
7.1.6	Conclusion and Future Work	188

Contents

7.2	Automatic Semantic Text Tagging on Historical Lexica	189
7.2.1	Related Work	189
7.2.1.1	Font Identification	190
7.2.1.2	Script Identification	191
7.2.1.3	Our Method	192
7.2.2	Material: Sanders' Dictionary	192
7.2.3	Methods	193
7.2.3.1	Preprocessing and Segmentation	194
7.2.3.2	Automatic Text Recognition	194
7.2.3.3	Typography Recognition	194
7.2.3.4	Combining the Outputs	196
7.2.4	Experiments	197
7.2.4.1	Data	197
7.2.4.2	Automatic Text Recognition	197
7.2.4.3	Typography Recognition	198
7.2.5	Discussion	200
7.2.6	Conclusion and Future Work	201
8	Conclusion	203
8.1	Summary	203
8.2	Outlook	206
8.2.1	Generic Interfaces for and Extensions of Core Components within the Workflow	206
8.2.2	Extending the OCR Functionality	209
8.2.2.1	Preprocessing	209
8.2.2.2	Segmentation	209
8.2.2.3	Automatic Text Recognition	211
8.2.2.4	Postcorrection	212
8.2.3	Intelligent Interactive Tools	213
8.2.4	Usability	213
8.2.5	Miscellaneous	214
8.3	Concluding Remarks	214
	Bibliography	217

List of Figures

1.1	Main steps of a typical OCR workflow	4
2.1	Visualization of text line components	31
2.2	Calamari’s default network architecture	39
3.1	Example pages from different Narrenschiff editions	60
3.2	Example pages and line images from different Camerarius works	62
3.3	Example pages from various early printed books	63
3.4	Example images from the German novel corpus	64
3.5	Example page from Sanders’ dictionary	65
3.6	Example lines from the Early New High German incunabulum corpus	68
3.7	Example lines from the Kallimachos corpus	69
3.8	Example lines from the Early Modern Latin corpus	70
3.9	Example lines from the RIDGES Fraktur corpus	72
3.10	Example lines from the DTA19 corpus	73
4.1	Example lines from the books used to evaluate cross fold training and voting	83
4.2	The basic idea of the cross fold training	84
4.3	Example line containing a strongly degraded “e”	84
4.4	Alignment result of the individual output of five voters	85
4.5	Example lines from the books used to evaluate pretraining	95
4.6	Schematic view of extensions or reductions of the output matrix of the network	97
4.7	Effects of building from different models compared to the default approach	102
4.8	Example lines from the books used to evaluate the AL	108
4.9	Comparison of the CERs of four different approaches	110
4.10	Influence of the number of GT training lines compared for different approaches	112
4.11	Results of the AL experiments for two different sets of lines	114
4.12	Snippet of a scanned page and its best ATR output by combining mixed pretraining and voting	116
4.13	Best and worst example lines identified by the committee	117
5.1	Example segmentation results of an early printed book with very complex layout	120
5.2	Example of the OCR4all folder and data structure	122
5.3	The main steps of the OCR4all workflow	123
5.4	Example input/output of the image preparation and preprocessing steps	125

List of Figures

5.5	Steps of the OCRopus 1 <i>nbin</i> binarization algorithm	126
5.6	Skewed and deskewed version of the same region image	128
5.7	Example LAREX segmentation output	129
5.8	Typical LAREX workflow for a single image	131
5.9	Result using the default and slightly adjusted parameters	133
5.10	An example of the OCRopus 1 <i>gpageseg</i> functionality	137
5.11	Example input and corresponding output of the line segmentation step for a very challenging set of lines	138
5.12	Example for a failed line segmentation due to varying font sizes	138
5.13	Example model selection process in OCR4all	140
5.14	Training settings view in OCR4all	142
5.15	Overview over the LAREX correction GUI	147
5.16	Example applications of the labelling and segmentation functionality on layout level in LAREX	148
5.17	Two text correction views in LAREX	150
5.18	Overview over the settings and parameters view of the line segmentation .	152
6.1	Example images of early printed books used to evaluate OCR4all	155
6.2	Schematic representation of the iterative training approach	162
6.3	Analysis of the iterative training approach	163
6.4	Representative example images showing the difference between the basic and exact segmentation approach	166
7.1	Example line images of the 20 works used to evaluate the 19 th century Fraktur models	176
7.2	Examples of synthetically generated lines	180
7.3	Input and color-coded output of our method for an example article de- picting the five typography labels	193
7.4	Typography GT production process of a single line	195
7.5	Typography alignment for an example line	196

List of Tables

2.1	Notable data sets used during the related work chapter	12
2.2	Comparison and rating of the capabilities of four modern ATR engines . .	42
2.3	Summary of the results achieved by mixed and book-specific models . . .	43
2.4	Comparison of existing tools providing an OCR workflow	50
3.1	Summary of the data used during this thesis	59
3.2	Books of the early modern age used for our experiments	61
3.3	German Fraktur novels used for our experiments	64
3.4	Overview of the subcorpora of <i>GT4HistOCR</i>	67
3.5	The Early New High German incunabulum corpus	68
3.6	The Kallimachos corpus	69
3.7	The Early Modern Latin corpus	70
3.8	The RIDGES Fraktur corpus	71
3.9	The DTA19 Fraktur corpus	74
4.1	Books used to evaluate cross fold training and voting	83
4.2	Example lloc output	86
4.3	Confidences of the model outputs for the two characters in question . . .	87
4.4	Comparison of the ISRI and confidence voting	89
4.5	CERs and improvement rates of confidence voting over the average result of single models	90
4.6	CERs when using 5 or 10 folds with different size of the training set . . .	90
4.7	Required time expenditure for the ATR using different scenarios	91
4.8	Books used for Evaluation	95
4.9	Resulting CERs when using different models without and with utilizing the whitelist	99
4.10	Resulting CERs from models trained by following different pretraining approaches	101
4.11	Resulting CERs from models trained by following the default approach compared to building from different models	103
4.12	Books used during the Active Learning experiments	107
4.13	Evaluation of voting, pretraining, and their combination	111
4.14	Results of comparing active to passive learning	114
5.1	Example confusion table of an ATR result	144
6.1	Results of the precise segmentation and trained ATR of early printed books	158

List of Tables

6.2	Evaluation of the iterative training approach	161
6.3	Comparison of two users and two different segmentation approaches regarding the segmentation with LAREX	167
6.4	CERs achieved by ABBYY Finereader and OCR4all when being applied fully automatically	170
6.5	The ten most common confusions for ABBYY Finereader and OCR4all	170
7.1	Details on the data used for evaluating the 19 th century Fraktur mixed models	177
7.2	ATR results achieved by the final 19 th century Fraktur OCRopus 1 and Calamari models	182
7.3	Results for the models emerging from different training steps	183
7.4	In detail comparison of our trained models with various open-source solutions and ABBYY Finereader	185
7.5	The ten most common confusions for different engines and models	186
7.6	Statistics of the evaluation data	197
7.7	CER of the line-based ATR dependent on the number of GT lines	198
7.8	Evaluation of the line-based typography recognition	199
7.9	Effect of data augmentation on the line-based typography recognition	200
8.1	Summary of the obtained results with focus on OCR4all	205
8.2	Planned features for OCR4all	207

List of Abbreviations

AH	Mixed Calamari Historical Antiqua Model
AL	Active Learning
ALC	Arabic and Latin Corpus
ALTO	Analyzed Layout and Text Object
AM	Mixed Calamari Modern Antiqua Model
API	Application Programming Interface
ATR	Automatic Text Recognition
cBAD	Competition on Baseline Detection
CC	Connected Component
CER	Character Error Rate
CIS	Center for Information and Language Processing
CLARIN	Common Language Resources and Technology Infrastructure
CLI	Command Line Interface
CNN	Convolutional Neural Network
CPU	Central Processing Unit
CSS	Cascading Style Sheets
CTC	Connectionist Temporal Classification
DFG	Deutsche Forschungsgemeinschaft - German Research Foundation
DIVA	Document, Image, and Voice Analysis
dpi	dots per inch
DTA	Deutsches Textarchiv
DTA19	19 th Century DTA Fraktur Corpus
EML	Early Modern Latin
eMOP	early Modern OCR Project
ENHG	Early New High German
ENG	Mixed OCRopus Modern Antiqua Model

List of Abbreviations

FCN	Fully Convolutional Network
FgPA	Foreground Pixel Accuracy
FH	Mixed Calamari Historical Fraktur Model
FRK	Mixed OCRopus 19 th Century Fraktur Model
GPU	Graphical Processing Unit
GT	Ground Truth
GT4HistOCR	Ground Truth for Historical OCR
GUI	Graphical User Interface
HisDB	Historical Document Image Database
HMM	Hidden Markov Model
hOCR	HTML OCR
HTML	Hypertext Markup Language
HTR	Handwritten Text Recognition
HCI	Human Computer Interaction
ICDAR	International Conference on Document Analysis and Recognition
IIIF	International Image Interoperability Framework
IMPACT	Improving Access to Text
ISRI	Information Science Research Institute
IoU	Intersection over Union
JSON	JavaScript Object Notation
KNN	K-Nearest Neighbours
LAREX	Layout Analysis and Region Extraction
LCS	Longest Common Substring
LDR	Levenshtein Distance Ratio
LH	Latin Historical
lloc	LSTM Location Of Characters
LSTM	Long Short-Term Memory
METAe	Metadata Engine
METS	Metadata Encoding and Transmission Standard
MST	Minimum Spanning Tree
NLP	Natural Language Processing

OCR	Optical Character Recognition
OpenCV	Open Computer Vision
PA	Pixel Accuracy
PAGE	Page Analysis and Ground-Truth Elements
PDF	Portable Document Format
PoCoTo	Post Correction Tool
PRImA	Pattern Recognition & Image Analysis
PSET	Page Segmentation and Evaluation Toolkit
RAM	Random-Access Memory
RAST	Recognition by Adaptive Subdivision of Transformation Space
ResNet	Residual Network
REST	Representational State Transfer
RGB	Red Green Blue
RIDGES	Register in Diachronic German Science
RLSA	Run Length Smoothing Algorithm
SMW	Semantic MediaWiki
SVM	Support Vector Machine
TEI	Text Encoding Initiative
UW3	University of Washington 3
WER	Word Error Rate
WL	Whitelist
XML	eXtensible Markup Language

1 Introduction

Over 550 years after Gutenberg’s groundbreaking invention of printing with movable types in 1453 a milestone on the road to preserving our cultural heritage on a long-term basis is about to be reached in the foreseeable future: The majority of all books that have ever been printed will have been digitized and made publicly available online [278]. This important development can mainly be attributed to large-scale digitization projects like for example the forerunner Project Gutenberg [121] initiated already during the 1970’s, the Hathi Trust Digital Library [124], the Million Book Collection [185], and of course Google Books [109].

Despite these developments being without a doubt very positive, the digitization of printed material can only be considered the starting point when it comes to preserving the knowledge comprised within it. While it is nice to be able to look up pretty much any given book online and scroll through its pages, one major thing is missing: *machine-actionable text*. Obviously, it makes a big difference if a text can be gathered by a human brain combined with human eyes page by page or if a machine can read, process, and analyze heaps of texts in a matter of seconds without any human intervention. There is a wide variety of approaches, methods, and algorithms which can be allocated to research fields like distant reading [186] or text mining [126], including topic modelling [35], sentiment analysis [217], or stylometry [61], just to name a few.

Naturally, manually transcribing millions of books is neither sensible nor even remotely possible. Consequently, the challenge of turning scanned pages into machine-actionable text is passed on to one of the oldest computer vision and machine learning tasks namely OCR. As we¹ will show in the following, this cannot always be considered a trivial task at all, since the challenges considerably vary with the material at hand. Still, making the masses of digitized cultural heritage available as text was and is widely considered a necessity. This is particularly stressed by a report issued by the European Commission in 2011 [195] which warns that Europe is in danger of “entering a new Dark Age” if no sufficient means of preservation and discovery of cultural heritage material are created [71]. As a consequence, several large-scale projects dealing with the OCR of historical

¹By definition, a thesis like the one at hand has to mainly focus on the contributions of an individual person. Nevertheless, many aspects of this work, especially the main outcome *OCR4all*, which has grown to be a very comprehensive project, resulted from the various contributions of several helping hands providing ideas, implementations, discussions, and feedback. This is also shown by the multitude of different co-authors who contributed to the individual publications leading up to this thesis. Hence, the author sticks to the “we” narrative throughout this thesis just like we did in the papers and articles before.

1 Introduction

printings in a mass digitization context were funded, most importantly *Improving Access to Text (IMPACT)* [32, 131] and the *early Modern OCR Project (eMOP)* [87, 173].

While great progress has been made during these and other projects, the quality of the resulting text is often not considered sufficient, especially when dealing with early printed books [278, 287] and/or texts printed in Fraktur types [92]. This is very problematic since, naturally, noisy OCR has a negative effect on many, if not all, research tasks [301] and a very recent report even states that the progress in digital scholarship is hindered by insufficient OCR quality [272]. Naturally, it depends on the individual task how good an OCR result has to be in order to be considered “sufficient”. Moreover, the difficulty and consequently the amount of manual effort necessary to achieve such a result is equally dependent on the properties of the material at hand, as we will discuss in the following.

1.1 Challenges of Historical OCR

While OCR on contemporary material is regularly considered to be a solved problem [85], gathering the content of historical printings using OCR can still be a very challenging and cumbersome task, due to various reasons. First, the pages of historical books are often in bad condition and heavily degraded which manifests in incomplete or partially faded letters, various types of soiling, bad contrast, or bleed-through of the ink from the back page. Referring to a more content-related/textual level, among the problems that need to be addressed for early printings is the often intricate layout containing images, artistic border elements and ornaments, marginal notes, and swash capitals at section beginnings whose positioning is often highly irregular. Another of course appears in view of typical early modern layout arrangements and varying, non-standardized printing conventions. These can lead to a highly complex order of text regions on single print sheets that is determined by a variety of columns, shifting text blocks, and drastically reduced distances/spaces between various layout and text components. Taking this into account, accurate layout segmentations and text recognition are complicated. This is especially true for very early printed books continuing the richly illustrated manuscript tradition after Gutenberg invented modern printing in 1453, where typesetters were especially keen on an artistic rendition of the page image, resulting in headings which conclude the page or swash capitals which severely overlap the adjacent text.

Also, the non-standardized typography represents a big challenge for OCR approaches. While modern fonts can be recognized with excellent accuracy by so-called omnifont or polyfont models, that means models pretrained on a large variety of customarily used fonts, the lack of computerized historical fonts prevents the easy construction of such polyfont or mixed font models for old printing material and one needs to resort to individual model training instead.

Therefore, very early printings like incunabula² but also handwritten texts usually require book-specific training in order to reach CERs well below 10% or even 5% as

²Books printed before 1501.

shown by Springmann et al. [279, 280] (printings) and Fischer et al. [95] (handwritten documents). For a successful supervised training process GT in form of line images and their corresponding transcriptions has to be manually prepared as training examples.

Additionally, the highly variant historical spelling, including a frequent use of abbreviations, severely hinders automatic lexical correction, since sometimes identical words are spelled differently not only among different books of the same period but even within the same book.

Character recognition rates in the high nineties are now routinely possible for even the earliest printings. However, this can only be achieved by training specific recognition models for each individual book, or at least for books coming from the same print shop and printed with the same font. This does not scale up very well for conversion of the already available substantial amount of scanned book pages from the 15th to 18th century [284]. Ideally one would construct models resembling the mixed recognition models employed by standard OCR engines such as Tesseract [274] or ABBYY Finereader³. They achieve very good overall recognition rates to more recent printings from the 19th century onwards, often with CERs of 1% and below.

The prime factor preventing the construction of effective models for earlier printings is the scarcity of GT training material, i.e. diplomatic⁴ transcriptions of real printings. The production of GT is a costly and slow manual process, which in the case of early printings often entails specialized knowledge to decode the meaning of palaeographic glyphs into Unicode [299] characters.

This barrier can be overcome for modern printings by the creation of synthetic training material, starting from available electronic text, which gets rendered into synthetic images using available computer fonts, often with some noise added to make the model more robust.⁵ For early printings we lack the pertinent fonts containing the specific shapes and glyphs used by individual printing shops. In the incunabula period from ca. 1450-1500, as many as 2,000 individual print shops employing 6,000 different fonts have been identified and collected in printed tables accompanying Haebler's monumental *Typenrepertorium der Wiegendrucke* [302]. Furthermore, a recognition model for early printings does not just depend on specific fonts but also on the interword distance, as printers meticulously cared for justified right margins and ran words closely together to make this happen if no convenient break point was possible. The difficulty of getting tokens correctly recognized becomes apparent when trained individual models have wrongly split or merged words as their most frequent error. The next more frequent error types are insertions, deletions, and substitutions such as $e \leftrightarrow c$.

³www.abby.com/Finereader

⁴A diplomatic transcription is one that records only the characters as they appear on the support, with minimal or no editorial intervention or interpretation.

⁵The new Tesseract neural network models for Latin scripts have been constructed using synthetic images with 4,500 fonts: <https://github.com/tesseract-ocr/tesseract/wiki/TrainingTesseract-4.00>

1.2 Steps of a Typical OCR Workflow

The actual text recognition in itself only represents one subtask within an OCR workflow usually consisting of four main steps (see Figure 1.1) which often can be split up into further sub steps. To avoid confusion, we use the term “OCR” when referring to the complete workflow and the term “Automatic Text Recognition” (ATR) when dealing with the sub step that focuses on the actual text recognition functionality. We avoid notations like “recognition” since they would be misleading as the step comprises more sub tasks than the text recognition alone. The steps of the workflow defined in Figure 1.1 as well as the sub tasks listed in the following will be discussed in greater detail during the upcoming *Background and Related Work* chapter.



Figure 1.1: Main steps of a typical OCR workflow. From left to right: original image, preprocessing, segmentation, ATR, postcorrection. Adopted from [229].

1. **Preprocessing:** First of all, the input images usually have to be prepared for further processing. Generally, this includes a step which simplifies the representation of the original color image by converting it into binary and sometimes grayscale, enabling further image processing operations later on, as well as a deskewing operation in order to get the pages into an upright position, which simplifies the upcoming steps. Additional routines like dewarping to rectify a distorted scan or denoising or despeckling to clean up the scan may be performed. Beforehand, it can be worthwhile to crop the printing area in order to remove unwanted scan periphery. It is worth mentioning that the usage of different image display formats can vary considerably during the upcoming steps, dependent on the individual workflow and its comprising tools. For example, it is well possible to perform the segmentation on a result of the preprocessing step, e.g. the binarized image, but then return to using the grayscale or even the original color image during the ATR.
2. **Segmentation:** Next, one or several layout analysis steps have to be conducted, mostly depending on the material at hand and the requirements of the user. After separating the text regions from non-text areas individual text lines have to be identified and, if required by the used ATR approach, split up even further into single glyphs. Optionally, non-text elements can be further classified (images, ornaments, ...), while text regions can be broken down into more or less fine-grained semantic classes (running text, headings, marginalia, ...), already on layout level. Another important sub task is the determination of the reading order which defines the succession of text elements (regions and/or lines) on a page. Naturally, the

option to manually correct the results achieved during these sub tasks is highly desired, preferably via a comfortably-to-use GUI.

3. **ATR:** The recognition of the segmented lines (or single glyphs) leads to a textual representation of the printed input. Depending on the material at hand and the user requirements this can either be performed by making use of existing mixed models and/or by training models which are specifically geared to recognize the font it was trained on. Again, for a comfortable correction of the textual ATR output and for producing good examples to be used for book-specific training, a GUI is mandatory.
4. **Postcorrection:** The raw ATR output can be further improved during a postprocessing step, for example by incorporating dictionaries or language models. This step can support or replace the manual final correction phase depending on the accuracy requirements of the user.

As for the final output, plain text, that is the (postprocessed) ATR output, has to be considered the minimal solution. Apart from that, almost all of the information acquired during the entire workflow can be incorporated into the final output: region coordinates and their types, line coordinates, character positions and information about how sure the ATR is about its predictions, and others. Several formats which can incorporate most or all of the aforementioned information have been proposed, for example Analyzed Layout and Text Object (ALTO)⁶, HTML OCR (hOCR) [55], or Page Analysis and Ground-Truth Elements (PAGE) [223].

Regarding the text recognition step it is worth mentioning that the models used for the recognition can be obtained in different ways with highly varying degrees of effort based on the material at hand. As quoted above modern fonts can usually be recognized by applying an existing standard model which has been trained on a variety of similar fonts, for example 19th century Fraktur, due to the comparatively high degree of homogeneity of the typography. On the contrary, (very) early prints often require type-specific training to reach character recognition rates in the high nineties.

1.3 Challenges for the Users

To produce training data for the ATR one has to manually transcribe text lines (considering a line-based approach), which is a highly non-trivial task when dealing with very old fonts which are often difficult to decipher and contain numerous ligatures and abbreviations whose transcription and/or decomposition requires knowledge about the historical language and the content of the texts. In fact, this step and often also several other steps of the OCR workflow cannot be performed fully automatically and require the user to interact and to invest manual work. The combination of all steps represents a highly interdisciplinary task and therefore requires both domain expertise regarding the

⁶<https://www.loc.gov/standards/alto>

1 Introduction

content as well as technical expertise, a combination which is difficult to come by in a single person. Since it is not possible to simplify the content related part of the problem, we have to focus on the technical aspect. Fortunately, large parts of these steps can be covered by open-source tools such as OCRopus [206], Tesseract [295], or Calamari [63] which have been made available.

While these tools are highly functional and very powerful their usage can be quite complicated, as they

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

These aspects are particularly problematic for inexperienced users with limited technical background. Unfortunately, this often includes humanities scholars as one of the main target audiences for all tools which allow to produce machine-actionable text from scans of historical printings. Making an entire workflow available to and usable by non-technical users is a challenging task, since most tools usually do not cover the entire workflow described above, at least not in a satisfactory manner, but rather excel on smaller sub tasks. Combined with the shortage of (user-friendly and GUI-supported) ways to manually interfere with the process and turn it into a semi-automatic approach, this considerably reduces the applicability of existing open-source tools.

1.4 Proposed Workflow Solution: OCR4all⁷

To deal with these issues, we present our open-source [196] tool OCR4all⁸ [229] which aims to encapsulate a comprehensive OCR workflow into a single *Docker* [183] or *VirtualBox* [311] application, ensuring easy installation and platform independency. The goal is to make the capabilities of state-of-the-art tools like OCRopus or Calamari available within a comprehensible and applicable semi-automatic workflow to basically any given user with the option to decide on different compromises between the resulting accuracy and

⁷This section as well as considerable parts of the entire thesis, most importantly chapters *Introduction* (Chapter 1), *Background and Related Work*, (Chapter 2), *OCR4all* (Chapter 5), *Evaluations* (Chapter 6), and *Future Work* (Chapter 8) are based on a previously published article [229]: C. Reul, D. Christ, A. Hartelt, N. Balbach, M. Wehner, U. Springmann, C. Wick, C. Grundig, A. Büttner, and F. Puppe, “OCR4all—An Open-Source Tool Providing a (Semi-) Automatic OCR Workflow for Historical Printings,” *Applied Sciences*, vol. 9, no. 22, p. 4853, 2019. [Online]. Available: <https://doi.org/10.3390/app9224853>

⁸<https://www.uni-wuerzburg.de/en/zpd/ocr4all>

manual effort. This is achieved by supplying the users with a comfortable and easy to use GUI and a modular approach, allowing for an efficient correction process in between the various workflow steps in order to minimize the negative effects of consequential errors. Another important aspect is the option to iteratively reduce the CER by constantly retraining the recognition models on additional training data which has been created during the processing of a given book. During development the primary goal was to identify a workflow and tool composition which enables the users to deal with even the earliest printed books on their own and extract their textual content with great quality. Due to the challenges concerning condition, layout, and typography described above, this is far from a trivial task and often requires the users to invest a substantial amount of manual effort into correcting segmentation results and transcribing line images in order to produce training data for the book-specific models. However, in our experience, humanist scholars who are dealing with early printed books are usually perfectly fine with investing the required effort to obtain high quality OCR results which had been considered almost impossible to achieve on this material only a decade ago [247]. This is especially true since the alternatives are either to manually transcribe everything or to not get to the text at all. Naturally, we did not want to restrict the users to the processing of very early printed books and therefore added further functionality to ensure a fluent passage towards a fully automated approach when dealing with later and more uniform works.

Despite our focus on user friendliness, operating OCR4all is still not an entirely trivial task (and will not be for the foreseeable future), especially when dealing with very early prints with a complex and irregular layout as well as a non standard typeface that makes a thorough book-specific training indispensable. Consequently, there is an obvious need for detailed, comprehensible, and descriptive operating instructions. To start things off we provide both, a setup guide and a comprehensive step by step user manual together with some example data at GitHub [197] and set up a mailing list⁹ where we inform about latest developments and new version releases.

1.5 Contributions

OCR4all represents the main outcome of this thesis. To arrive at the fully functional tool which is still under active development and used by an extensive and always growing user community, several challenges had to be mastered: First, a workflow able to deal with all the peculiarities of early printings had to be designed. Next, for each step of the workflow suitable approaches and tools had to be identified and combined into a flexible and ergonomic open-source end-to-end OCR workflow. On the one hand this included the integration of existing tools by implementing common interfaces and defining sensible default parameter configurations. On the other hand the workflow had to be complemented with several in-house developments including the means for extensive segmentation and correction capabilities as well as the encapsulating OCR4all tool.

⁹<https://lists.uni-wuerzburg.de/mailman/listinfo/ocr4all>

1 Introduction

Moreover, an important additional contribution of this thesis is that the extensive interactive post correction functionalities provided by OCR4all not only allow to optimize the result for the current book but also to produce large quantities of valuable training materials, both for layout analysis and text recognition, to support a process model towards continuously higher automation.

Apart from the main goal of enabling non-technical users to perform OCR even on the earliest printed books completely on their own, several methodical extension in the form of accuracy improving techniques for training and recognition were implemented and evaluated. Most notably an effective voting methodology was proposed, allowing to train several ATR models using just a single ATR engine and combining their outputs not only by a simple majority voting but by taking the intrinsic confidence information of the neural network into account. Combined with a pretraining procedure, that allows the training to profit considerably from already existing models, and an Active Learning component, the effectiveness and efficiency of the training and recognition step was considerably improved.

The effectiveness and efficiency of OCR4all is thoroughly examined during two in-detail core evaluations on historical books. First, experiments on very challenging material from the 15th and 16th century led to results of previously unprecedented quality, despite being mostly performed by inexperienced users with (next to) no technical background. In addition, several user-centered studies are provided, dealing with the influence of user experience and quality requirements, to gain a better understanding of the interaction of human and computer. Second, the capabilities of the fully automatic approach are evaluated using 19th century novels with moderate layout, showing excellent error rates and outperforming the commercial state-of-the-art tool ABBYY Finereader.

Finally, two real-world case studies are provided. The first one deals with the transfer of the aforementioned accuracy-improving techniques from book-specific training to the area of mixed models using 19th century Fraktur script as an example. In the second case study we extend our focus from raw ATR to the adjacent topic of typography recognition by thoroughly indexing a historical lexicon that comprises a complex semantic structure encoded by different font types.

1.6 Structure of this Work

The remainder of this work is divided in eight chapters: In Chapter 2 we provide a comprehensive overview about previous work related to this thesis focusing on the general OCR workflow consisting of the four main steps introduced above: preprocessing, segmentation, ATR, and postcorrection.

Next, Chapter 3 introduces the data and resources we used for our evaluations including entire books to process with OCR4all, line-based GT for training and recognition tasks, and a variety of mixed models, either publicly available or self-trained ones.

The methodical contributions, namely accuracy-improving ATR techniques like cross fold training and subsequent confidence voting, pretraining, and AL, are described, evaluated, and discussed in Chapter 4.

In Chapter 5 we thoroughly describe OCR4all including the overall workflow and the single sub modules. Additionally, we provide deeper insight on the technical background of tools and methods we included from external sources.

While the partial evaluations in Chapter 4 exclusively deal with specific ATR sub tasks, Chapter 6 mainly focuses on the evaluation of the complete OCR workflow by describing several experiments performed on a variety of historical printings. Furthermore, Human Computer Interaction (HCI) aspects like the influence of the experience of the users are examined as well as the significance of the material at hand and the output complexity and quality desired by the users.

Chapter 7 contains two real-world case studies covering 19th century Fraktur mixed model training and typography recognition.

Finally, Chapter 8 concludes the thesis by summing up the results and their implications and pointing out our goals for the future of OCR4all and our work in the area of (historical) OCR in general.

2 Background and Related Work

This chapter gives a comprehensive overview over OCR related tools, methods, and topics oriented towards the following guidelines:

- We address all steps of the OCR workflow in chronological order (as defined in Figure 1.1) but to varying levels of detail.
- The main focus lies on steps which we either directly and considerably contributed to over the course of this thesis (region segmentation and OCR workflows) or which we consider to be very important or especially interesting for our past, present, and future work (line segmentation).
- In general, we aim to first provide an overview over important fundamentals, basic concepts, and methods for each step and then discuss selected state-of-the-art approaches.
- For the main part, the focus lies on applications related to historical OCR but depending on the task and the transferability of existing solutions, we also incorporate solutions for adjacent topics like modern printings or handwritten texts.
- Fitting stand-alone tools are given and discussed for each step. Due to our general goal of an end-to-end open-source workflow, we focus on solutions that are freely available. We discuss several OCR workflow tools as a whole but also briefly address their capabilities regarding individual steps during the respective sections. Because of ABBYY Finereader’s proprietary and closed-source nature and the resulting lack of backed information about the workflow, Kraken’s proximity to OCRopus 1, and the black box nature of OCRopus 3, we focus on OCRopus 1 and Tesseract 3/4 for these parts (cf. Section 2.6 for an in detail description of the individual workflow tools). To the best of our knowledge, published evaluations dealing with OCR workflow tools either assess the achieved results as a whole or focus on the actual training and recognition step. Therefore, in most cases, we refrain from providing results achieved by OCRopus 1 or Tesseract for the individual workflow steps.
- Specialized related work dealing with our methodical contributions like voting and pretraining for book-specific and mixed models, their combination and the inclusion of AL, as well as typography recognition, will be presented in the corresponding subsections in the methods and case study chapters (Chapter 4 and Chapter 7, respectively).

2.1 Data Sets

Before focusing on the individual steps of the workflow we briefly introduce some data sets (see Table 2.1) which we will refer to throughout the remainder of the chapter. This list is not intended to be exhaustive but only lists selected data sets which are used by several approaches we address. For example, non-public data sets or those without any usable description or documentation are ignored altogether. Moreover, we skip data sets that are used extensively during our own evaluations or that have been created by ourselves as they are covered in detail during Chapter 3.

Table 2.1: Notable data sets applicable to the tasks of page frame detection (*PF*), layout analysis or region segmentation (*RS*), line segmentation (*LS*), and *ATR*.

Name	Source	Task(s)	Material
UW3	[222]	PF, RS, ATR	various scientific printings from the 20th century
DIVA-HisDB	[268]	RS, LS	three handwritten documents written from the 11th to 14th century
cBAD	[269]	RS, LS	various handwritten documents written from the 15th to 20th century
PageNet	[293]	PF	cf. cBAD

The University of Washington 3 (UW3) database [222] consists of ca. 1,600 pages from late 20th century English speaking sources. As GT it contains bounding boxes enclosing text and non-text regions, text lines, and words as well as the corresponding textual contents. For page frame detection tasks the required GT can be derived from the region GT by detecting the enclosing rectangle of all text regions. Furthermore, the UW3 data set is frequently used for evaluating the performance of ATR engines which will be addressed in greater detail in Section 3.2.3.

The *DIVA-HisDB* (Document, Image, and Voice Analysis (DIVA) Historical Document Image Database (HisDB)) [268] is a precisely annotated data set containing three challenging medieval handwritten documents, that have been written between the 11th and 14th century in Switzerland and Italy. The annotations consist of tightly fitting (mostly line) polygons that distinguish between three categories: the main text, decorations (mostly swash capitals), and comments (marginalia and glosses as well as corrections and explanations). After the initial annotation and a subsequent correction iteration, all pages have been checked and scrutinized by an expert, assuring high quality annotations. Overall, 150 pages have been annotated, 50 for each of the three works, and are publicly available [127].

An important resource frequently used in recent International Conference on Document Analysis and Recognition (ICDAR) competitions on layout analysis and baseline detection (see for example [84]) is the *Competition on Baseline Detection (cBAD)* data set [116]. It

is publicly available [67] and comprises 2,035 images of handwritten documents produced between 1470 and 1930 and collected from nine different European archives. According to its authors it is the first text line segmentation data set that exclusively relies on baselines (cf. Figure 2.1 for an explanation of text line components, including the baseline) as annotated GT. Apart from line segmentation tasks cBAD can also be used in region segmentation tasks, since, while there is no explicit markup of non-text layout parts like images or ornaments, columns and marginalia are naturally identified as different text blocks, by splitting their baselines. Consequently, the data set is further divided into two tracks: The first one solely contains pages with simple layouts and is therefore intended for evaluating baseline detection methods only. The second track is considerably more difficult as it contains multi column layouts and rotated text lines, and extensive tables, adding the additional challenge of correctly splitting the baselines.

The *PageNet* data set [293] for page frame detection is derived from the cBAD data set introduced above. Both tracks of the competition data were used and the comprised images were divided into training, validation and test sub sets containing 1,635, 200, and 200 images, respectively. In the original publication several other, sometimes non-public or insufficiently specified collections were added to the data set which consequently were not used for comparison by other approaches and are therefore ignored.

2.2 Preprocessing

The main purpose of the preprocessing step is to prepare the scanned images for the remainder of the OCR workflow. This includes enabling the application of the upcoming algorithms, for example by providing simplified image formats like binary color (black and white), as well as cleaning up the image. In the following we first address the different sub steps before introducing several tools that cover one or several of those sub steps.

2.2.1 Sub Steps

We treat the preprocessing step as combination of three sub steps, namely binarization, skew orientation detection and correction, as well as page frame detection and noise removal, which we discuss separately in the upcoming sections. Further special cases like the splitting of double pages that have been scanned at once [286] or the dewarping of curled pages [258] are not addressed since they are either rather trivial to solve or, judging from our experience, do not occur very regularly, at least not to a degree that leads to a notable negative influence on the segmentation or ATR steps.

2.2.1.1 Binarization

While the actual ATR can also be performed on grayscale or color images, sometimes even resulting in superior results compared to using binary inputs [326], converting an image

2 Background and Related Work

into binary is usually an indispensable step in an OCR workflow since it simplifies the description of an image considerably and consequently represents a prerequisite for the efficient application of many image processing operations. In general, existing approaches can mainly be divided into two groups, *global* and *local* thresholding techniques which we will briefly explain in the following. For further reference, we refer to a very recent and comprehensive survey by Sulaiman et al. [288].

Global Thresholding Techniques The first group consists of methods that determine a *global* threshold, in most cases simply a grayscale value, and that afterwards assign 0 (black) to all pixels whose grayscale value is lower than this threshold and 1 (white) to all others. The most prominent and still widely used representative of this group is *Otsu's Method* [213] which finds a statistically optimal global threshold by searching for an allocation of pixels into two classes (foreground and background) that optimizes a function that takes the inter- and intra-class variances into account.

The drawback of Otsu and similar methods that operate on a global level is that they can suffer considerably from effects like uneven illumination or soiled document parts. For example, if the left half of a page is significantly brighter than the right half, applying a global binarization operation may well lead to background pixels (for example paper) on the right being treated as foreground in the binary image, as they are as dark as the actual foreground pixels (for example ink) on the left.

Local Thresholding Techniques To deal with these issues, the second group of methods, so-called *adaptive* ones, were developed which try to determine *local* thresholds instead. A noteworthy approach of this group was introduced by Sauvola and Pietikäinen [250] who consider a page as a collection of subcomponents like text, images, and background. In a first step a hybrid switch takes small, resolution adapted windows on the page into account and assigns one of the aforementioned subcomponent classes using simple features derived from the input grayscale image. Based on this initial classification two different methods are applied: the soft decision method uses noise filtering and signal tracking capabilities to deal with background and images and the text binarization method uses a modified version of *Niblack's* algorithm [193] to binarize textual components. In a final step, the outcomes of both algorithms are combined.

Evaluation Challenges Many performance metrics for the evaluation and comparison of binarization techniques have been proposed over the years. Since binarization does not represent a core topic of this thesis, we refrain from an in-detail discussion but instead refer to the comprehensive overview given in [288].

However, it is worth mentioning that, despite the simplicity of many performance measures like the *F-measure* which is directly computed from a pixel-based binarization result and the corresponding GT, obtaining a meaningful comparison can be a challenging task. This is explained by the fact that determining the GT, i.e. which pixels belong

to the foreground and which to the background, usually contains a certain degree of arbitrariness, especially when dealing with challenging historical images.

To circumvent this issue and the need to create explicit binarization GT altogether, sometimes the result of a subsequent step of the workflow, namely the text recognition, is utilized to indirectly judge the quality of the binarization output (see for example [6]). Concretely, this could mean to binarize a given image with different techniques, then use the results as input for identical segmentation and ATR processes, and finally compare the ATR error rates, which only requires a comparatively easy to produce and rather unambiguous textual GT. While this simplifies the evaluation process considerably there are apparent drawbacks: First, the ATR in itself represents an element of uncertainty since the chosen engine and especially the utilized model naturally considerably influence the outcome and there is no guarantee that a “better” binarization result will lead to a better ATR result.

For example, if a model has been trained on flawed binary line images it might well perform better on other, equally flawed ones than on “perfectly” binarized ones. Hence, a fair comparison would require two models to be trained and evaluated on the respective binarization results. While this procedure might determine the “better” binarization result for this specific use case, its implications regarding the practical use remain unclear as existing mixed models might show a completely different behaviour. Unfortunately, these models have often been constructed using comprehensive training data which may not be available at all or only as individual line images, which most of the times have already been converted to binary. Consequently, these models cannot be reproduced using the “better” binarization method. Second, relatively small failures or rather changes during the binarization step may have a significant effect on the segmentation and consequently the text recognition step, falsifying the results and leading to incorrect conclusions regarding the quality of the binarization. For example, during our experiments for [230] we encountered a related problem where slightly different and completely inconspicuous region segmentation results, which could well have occurred because of different binarization outcomes, led to the merging of two vertically adjacent text lines, resulting in a completely useless ATR result.

While to the best of our knowledge there currently is no entirely satisfactorily solution available for the problems described above, it has to be said that their impact on real world applications is rather small. When dealing with historical printings, especially with (very) early printed books, visual inspection and comparison of different binarization results is highly recommended when more than one fitting approach is available.

2.2.1.2 Skew and Orientation Detection

Similar to binarization, skew detection and correction is an important preprocessing step in OCR workflows since subsequent steps like segmentation and ATR can suffer considerably from skewed images. The same is especially true for the related but less challenging task of orientation detection, which focuses on deciding whether a page is

2 Background and Related Work

in a (somewhat) upright position or rotated by (approximately) 90, 180, or 270 degrees [310].

In the following we describe a selected approach for skew and orientation detection because it builds from several previously released methods and achieved state-of-the-art results. For a broader overview we refer to the comprehensive survey by Cattoni et al. [66].

Van Beusekom et al. [310] use a geometric model based on text line properties for one-step skew and orientation detection as well as text line extraction, if necessary. The text line model [51] relies on a branch-and-bound approach [54] to find a globally optimal solution for text lines with respect to a least square error model. To achieve this the baseline and descender line¹ are modeled by parameterizing the baseline using polar coordinates and its distance to the descender line which is assumed to always be constant for an entire text line. The Recognition by Adaptive Subdivision of Transformation Space (RAST) algorithm [50] is utilized to find these parameters for all lines in a page image efficiently. The key idea of the proposed approach is to calculate the descender model for a page for orientations of 0, 90, 180, and 270 degrees and then choose the one with the highest quality. Since the method also detects text lines and the angle for the individual lines is known, the skew angle of the entire page can be easily derived by simply choosing the skew of the line with the highest quality value as page angle which has proven to be very accurate [51]. Evaluations on two public data sets, including UW3, showed the superiority of the approach compared to the methods incorporated in Tesseract and Leptonica [36, 162].

2.2.1.3 Page Frame Detection and Noise Removal

The page frame of a scanned document is defined as the smallest enclosing rectangle of all foreground elements of the page [266]. Scanned book pages often contain considerable amounts of periphery like scan background, (textual) noise from adjacent pages, or scan artifacts like fingers and tools of the processor or black borders caused by the binarization of page boundaries. Identifying the actual page area in a scanned image considerably simplifies the subsequent steps of an OCR workflow as especially the page segmentation step can suffer from noise [260, 264]. To the best of our knowledge there are no surveys available that exclusively or at least extensively deal with the problem of page frame detection or noise removal.

Because of this and the fact that there are three perfectly fitting and powerful open-source solutions available (dhSegment, ScanTailor, and Unpaper) which will be discussed in detail in the following, we just provide a very brief survey based on the contributions in [266] and [259].

Early approaches for noise removal focused on filtering out Connected Components (CCs) based on simple features like their size and aspect ratio [29, 53]. Of course, this does not

¹Fictitious lines supporting the main character bodies (baseline) or descenders (descender or lower line) of a text line. We refer to Section 2.3.2.1 for a more in-detail description of the line components.

work for textual noise. Fan et al. [91] detect marginal noise by utilizing a region growing method and remove it using a local thresholding technique in grayscale images.

Many methods rely on projection profiles to detect the page frame. Projection profiles are usually created by summing up pixel values along an image’s rows or columns and can thereby provide information about the structure of the document. For example Le et al. [158] propose a rule-based algorithm utilizing heuristics to classify rows and columns within the page into blank, text, and non-text. Cinque et al. [72] rely on image statistics like difference vectors and row luminosities. Peetawit et al. [221] use projection profiles of edges to detect non-textual marginal noise under the assumption that these areas have a much higher density of edges than normal text. Another projection profile-based approach was presented by Stamatopoulos et al. [285] who analyze the profiles of the smeared image combined with a CC labeling techniques, again based on rules. A method especially geared towards the splitting of two pages that have been scanned at once is presented by Stamatopoulos et al. in [286] and uses vertical white run projections.

All the approaches introduced above rely on rules designed to identify noise regions near the page border which cannot be expected to work on a large variety of documents which were scanned under very different conditions [266]. Two more robust approaches that have also been evaluated on a public data set will be briefly described in the following.

Shafait et al. [266] detect the page frame of documents by exploiting their text alignment properties in two steps: After producing a geometric model for the page frame of a scanned image, a globally optimal frame is determined by geometric matching utilizing a quality function. Evaluations on the UW3 data set showed error rates below 4% for three different performance measures and the subsequent CER achieved by a commercial ATR system was reduced from 4.3 to 1.7%.

A quite simple and very comprehensible method of Shafait and Breuel introduced in [259] consists of three successive steps: First, a black filter detects large black areas at the page border. Next, the detected CCs near the border of the image are removed. Finally a white filter finds big white blocks near the borders and, starting from there, removes everything until the border, to deal with page transitions. Experiments showed a reduction of the noise ratio from 70% to 20% while losing less than 1% of the actual page contents. Despite the simplicity of the approach evaluations on the UW3 data set yielded comparable results to the approach introduced above [266].

dhSegment A very interesting and promising approach for various tasks within the OCR workflow, including page frame detection, is *dhSegment* [212], an open-source [83] and well documented [211] generic deep learning framework for document-related segmentation tasks. The system aims to provide a flexible and adaptable solution for pixel-wise segmentation related tasks on historical documents by following a general approach of two successive steps: In a first step a Convolutional Neural Network (CNN) predicts a map of probabilities of labels, e.g. page content or periphery, for each pixel using the original document image as input. The probability map is then transformed

2 Background and Related Work

into the desired output of a specific task, e.g. a binary mask marking the page content page frame detection, by utilizing a selection of standard image processing techniques in a second postprocessing step.

dhSegment was used to tackle several task which are addressed during this chapter, namely page frame detection, layout analysis, and detecting the baseline of text lines. Since this is the first occurrence we will briefly discuss technical aspects like the network architecture and the training process before providing details regarding postprocessing and discussing the task of page frame detection. For the other tasks, we refer to the upcoming sections dealing with region and text line segmentation.

dhSegment relies on adaptations of the ResNet-50 [125] architecture and uses its pretrained weights in order to increase generalization and robustness. It profits from already learned high level features obtained from training for the general image classification task ImageNet [77]. The training procedure is rather straight forward since the default parameters fit most areas of application and consequently only the desired dimension of the resized input image has to be specified. Basically, the required GT consists of masks of the same size as the input images providing a pixel-based labeling of the desired output classes, for example page frames, regions, or baselines depending on the specific sub task. During the actual training, rather standard techniques like the Adam optimizer [144], Xavier [108], and batch (re)normalization [134] are incorporated. In addition, on-the-fly data augmentation is utilized to ensure an efficient use of the training material.

Only a few basic image processing techniques are needed for postprocessing showing the effectiveness and genericity of the approach:

1. *Thresholding*, either using a fixed value or relying on Otsus’s method (cf. Section 2.2.1.1), allows to turn the prediction map into a binary map for each class.
2. *Morphological operations* [253] like erosion and dilation or their combinations opening and closing can be applied to binary images for example to perform clean up operations.
3. *Connected component analysis* to filter out small CCs.
4. *Shape vectorization* perform a path reduction in order to transform detected regions into a set of coordinates like polygons, their bounding rectangles, or baselines.

To evaluate the performance of dhSegment in the area of page frame detection the PageNet data set (cf. Section 2.1) was used to compare against the eponymous deep learning system [293] as well as against standard approaches, most prominently *GrabCut* [245]. The dhSegment network was trained on 1,635 pairs of page images and binary masks marking the page content to predict for every pixel of a document image whether it belongs to the fore- or background of the page. Contrary to other tasks, the input image was not cut into patches but passed into the network at once after being resized to a certain size while retaining its aspect ratio. After applying Otsu’s method to the probability output and cleaning the binary image using opening and closing operations, the final result is extracted by finding the bounding rectangle of the page CC. With

a mean Intersection over Union (IoU) score (cf. Section 2.3.1.1) of 0.980 dhSegment outperformed GrabCut (0.916) considerably, the task-specific PageNet approach (0.974) slightly, and got very close to the score achieved by human annotator agreement (0.983).

2.2.2 Tools

To conclude the preprocessing step we present several tools which tackle the various sub tasks described above. While image processing libraries like *OpenCV* [44] and *ImageMagick* [130] definitely provide powerful tools for a multitude of different tasks, adaptations of them more geared towards document analysis are quite rare [100]. Hence, we will focus on dedicated OCR solutions here as well as in the tool subsections of the upcoming steps. Unfortunately, it has to be said that, despite considerable research efforts in the area of preprocessing of document images and many publications resulting from it, the achieved results often do not find their way into practical applications [323].

2.2.2.1 ScanTailor

ScanTailor [21] is an interactive open-source [20] postprocessing software for scanned pages which offers a variety of tools and routines allowing to prepare scans for further processing. Among others this includes:

- Rotating scans that are available in landscape format into an upright position.
- Splitting pages that have been scanned together into two single pages.
- Deskewing.
- Removing the scan periphery and cutting out the print space.
- Converting the images into binary.
- Several smaller preprocessing techniques like despeckling and dewarping.

Most of the steps can be performed manually or by fully automatic routines whose effectiveness highly depends on the specific task and the material at hand. For example, while the deskewing works very reliably, cutting out the print space produces severe errors on complicated layouts on a regular basis. Nevertheless, the majority of steps can usually be performed with limited manual interaction and manageable human effort.

Unfortunately, the development of the project appears to be dormant and to the best of our knowledge the basic strategies and technical details of the mentioned routines were never published, neither were suitable evaluations. Consequently, we have to rely on the verdict of the community as well as our own experiences and conclude that *ScanTailor* is a powerful open-source tool that provides many helpful semi-automatic features for the preparation and preprocessing of scans but whose fully automatic routines sometimes lack the desired reliability.

2 Background and Related Work

2.2.2.2 Unpaper

Unpaper [306] is another open-source [307] tool that can be used to clean up scanned pages by applying several useful postprocessing tools:

- A variety of filters to reduce different kinds of noise.
- Skew and orientation detection and correction.
- Page splitting and merging.
- Several mask operations allowing to relocate the page content and to manipulate borders.

While Unpaper is missing the comfortable GUI of ScanTailor and its tools, its advantage is the easy integration into fully automatic workflows. Similar to ScanTailor there are no related publications or extensive evaluations available. A common opinion within the community states that compared to ScanTailor Unpaper overall is less powerful and not as reliable. Very brief evaluations in [80, 259] concluded that Unpaper regularly fails to distinguish text from textual noise if more than one column is present but does well when it comes to removing black clutter noise even if it is located in close proximity to the text regions.

2.2.2.3 Implementation in OCR Workflow Tools

OCROPUS 1 relies on an adaptive binarization technique using percentile filters [6] and on a robust brute force approach utilizing projection profiles for skew detection and correction. Since both methods are incorporated into OCR4all they will be explained in detail during the introduction of the OCR4all workflow in Chapter 5 (cf. Sections 5.4.2 and 5.4.3). The same holds for certain denoising techniques that are part of the OCROPUS 1 line segmentation algorithm. An explicit page frame detection is not part of the OCROPUS 1 tool set. To the best of our knowledge, no evaluation results for the OCROPUS 3 preprocessing have been reported, yet.

In [6] a brief comparison between the percentile filter binarization and Sauvola's algorithm is presented. While Sauvola performed slightly better in terms of F-measure on the utilized low quality scans, it was considerably outperformed when taking the resulting ATR accuracy into account.

Tesseract 3/4 uses Otsu's method as its binarization techniques and relies on the open-source library Leptonica [162] to internally perform other preprocessing sub steps. However, as indicated by a Tesseract wiki page dealing with ways to improve the ATR quality [296], Tesseract is not guaranteed to be able to deal with uneven illumination, large skewing angles, and some kinds of noise including very prominent page borders. Hence, in these cases the usage of external preprocessing methods and tools is recommended.

2.3 Segmentation

The segmentation step aims to use the preprocessed images as input and extract structural information which is then utilized to enable and improve the subsequent text recognition task. As stated during the first chapter, this step is multifaceted and the applied sub steps and techniques heavily depend on the task and material at hand as well as the demands of the user. This thesis treats the segmentation step as a combination of two consecutive and sometimes overlapping² sub steps, namely region and line segmentation, which we discuss separately in the following.

2.3.1 Region Segmentation

In general, we agree with the definition given in [261] that formulates the goal of page or region segmentation as “to divide the document image into homogeneous zones, each consisting of only one physical layout structure (text, graphics, pictures, ...)”. However, the definition of a “physical layout structure” is blurred, up for debate, and, again, highly material, task, and user-dependent. While the minimal solution simply consists of separating text from non-text regions the other end of the spectrum is considerably more complex and includes explicit markup and distinction of non-text elements, like images or decoration, as well as a fine-grained semantic distinction of text regions into sub types like running text, headings, or marginalia. In the following, we cover many approaches dealing with different manifestations and varying degrees of complexity of region segmentation. Again, we first introduce some fundamentals before discussing basic approaches, the state-of-the-art, and available tools.

2.3.1.1 Fundamentals

Many comprehensive surveys dealing with the task of page segmentation have been published over the last decades [66, 178, 189] but also quite recently [90]. Before selected examples of the proposed approaches are discussed in detail over the next three sections, the remainder of this section briefly introduces some basic definitions before focusing on the non-trivial topics of performance metrics and implicitly also the task of result and GT representation in the area of region segmentation.

Definitions In general, region segmentation approaches can be divided into two groups: *bottom-up* and *top-down*.

Bottom-up approaches start with local information, for example CCs or even individual foreground pixels, before they perform iterative grouping operations in order to progressively form higher-level descriptions like words, text lines, and regions. While iterating

²Some approaches first detect text lines and then combine them into text blocks or even skip an explicit region segmentation altogether.

2 Background and Related Work

through the different layout levels of a document can provide valuable intermediate results like identified text lines, typical drawbacks of bottom-up methods are computationally complex operations during early iterations and their vulnerability to error propagation. [161]

In contrast, top-down approaches rely on global information instead, for example by using black or white stripes as a starting point to then recursively split the page into regions, the regions into lines, and maybe even the lines into words and the words into individual characters. Top-down approaches tend to be faster than bottom-up ones but usually struggle with more complex layouts comprising non-rectangular regions and different font sizes. [161]

Finally, the term “Manhattan layouts” frequently occurs in the area of region segmentation and refers to layouts where different regions are separable by horizontal and vertical line segments [289]. Naturally, this property usually simplifies the segmentation task considerably.

Performance Metrics Several metrics assume each region in the segmentation result and in the GT to be represented as a closed contour like a rectangle or a polygon, containing a set of pixels. A simple way to measure the similarity between two sets is the so-called IoU score [163] stemming from the *Jaccard coefficient* [135] which is calculated by dividing the intersection of the two sets, in our case the number of pixels that occur in both of them, by their union.

Mao and Kanungo [176] proposed a method based on evaluating all text lines of a page, distinguishing between three types of errors: Text lines which were defined in the GT can be missed, split, or merged. The advantage of this approach is its independence from the shape of a text block and the fact that GT only has to be present at text line level. However, this method does not support a semantic distinction of text types and completely ignores non-text elements, for example images. The proposed approach was made publicly available within the *Page Segmentation and Evaluation Toolkit (PSET)* [177].

Shafait et al. [263] presented a methodology to represent and evaluate page segmentation results by assigning each foreground pixel the value of the label of the segment it belongs to. To tackle the problems discussed above, they use a distinct label for every individual segment independent of its semantic type. To achieve this, a 24-bit RGB can be applied enabling the use of $2^{24}-1$ labels. When both the GT as well as the proposed segmentation are encoded in this way the evaluation is performed by computing and analyzing a weighted bipartite graph called the *pixel-correspondence graph* [52]. In the graph, nodes represent segments in the GT and the proposed segmentation and edges describe their relation, including an edge weight that scales with the degree of overlap of the connected regions. The graph then allows to easily and precisely derive six performance measures for page segmentation similar to the true positives, true negatives, etc. metrics.

The most direct way to assess a segmentation result is to count the correctly and incorrectly classified pixels and compute the so-called *Pixel Accuracy (PA)*. Prerequisite

for this method is GT on pixel level. The strengths of this approach is its simplicity and presentiveness. However, the obvious drawback is that the effect of small regions can become almost negligible especially in the presence of very large regions like periphery or the page background. Furthermore, there is a high chance for a certain arbitrariness regarding the exact positioning of the outline of GT regions. For example, when a predicted region tightly fits a text block but the GT region was determined to be slightly bigger, this can heavily influence the evaluation result despite having no effect on subsequent tasks like line segmentation and ATR. To deal with this issue, the so-called *Foreground Pixel Accuracy* (FgPA) [317] only takes pixel into account that have previously been determined to be foreground, for example during a preceding binarization step. Naturally, this metric also has its drawbacks since, despite the fact that it is dependent on the applied binarization method, it does not measure if whole regions have been correctly identified but rather only if single pixels or CCs have been assigned the correct type which, depending on the task at hand, might not be sufficient, e.g. if two text columns are merged into one text region. These problems could be mitigated by assigning different labels not only to regions belonging to semantic classes like running text and marginalia, but also to a different region of the same semantic class, for example to two columns of running text. Of course, a more sophisticated evaluation scheme would then have to be applied.

Clausner et al. developed a very comprehensive framework for evaluating layout analysis methods [75] which has already been deployed in several challenges [17, 18]. In order to guarantee an efficient calculation of the results, the polygons, stored in the PageXML format (cf. [223] and Section 2.7.1), are first transformed into an interval representation. From the hereby obtained overlap information several types of errors can be derived: merge (a region overlaps more than one GT region), split (GT region is overlapped by more than one region), miss/partial miss (GT region does not or not entirely overlap a region), failed detection (region does not overlap a GT region) and misclassification (GT region is overlapped by a region of a different type). For every type of error different metrics like precision, recall, and F-measure are computed based on the degree of overlap. Furthermore, it is possible to create application-oriented scenarios by assigning varying weights to different types of errors. The framework is very flexible and can be configured thoroughly. However, the free of charge version does not support batch evaluation which limits the practical utility tremendously.

In conclusion, evaluating page segmentation results remains a challenging task, since even the most widely used pixel-based methods have clear drawbacks. The best way to evaluate results highly depends on the task at hand. Furthermore, combining several metrics, for example the method proposed by Clausner et al. and a PA-based metric, seems to be a sensible choice, which might, however, often not be easily feasible due to the various GT requirements of the different approaches.

2.3.1.2 Basic Approaches

There is a variety of approaches for the segmentation of scanned documents. First, some fundamental methods are introduced which served as a foundation for the development of several further approaches. The selection and description is based on [262] where a comprehensive performance evaluation of these approaches is provided that we summarize after the introduction of the individual methods.

The *recursive X-Y-Cut* Algorithm [190] is a top-down approach based on a tree representation of the document. The root of the tree represents the entire page and the leaves the resulting segmentation. Through a number of subsequent horizontal and vertical cuts the documents gets recursively divided into smaller, rectangular blocks, which correspond to the leaves of the tree. Each decision if and where to perform a cut is based on the horizontal and vertical projection profile of the background pixels computed for each node. To identify the cutting candidates, i.e. the valleys in the projection profile histograms, two thresholds for noise removal are defined and scaled on the size of the currently active segment. Then, after setting all bins with values below these thresholds to zero, the resulting valleys in x - and y -direction are evaluated against previously defined thresholds determining the minimum width of a valley necessary to justify a split. Finally, the widest eligible candidate is chosen and the node is split at the center of the valley. The algorithm terminates when no leaf node can be further divided.

The so-called *Run Length Smoothing Algorithm (RLSA)* [320] considers pixel sequences in a binary image (background: 0, foreground: 1) and changes zeros to ones, if the number of adjacent zeros is under or equal to a given threshold C . This step merges nearby black areas that are separated by C pixels or less and is performed independently for rows and columns using two different thresholds. The resulting images are then combined by applying a simple logical AND operation. After performing another horizontal smearing operation with a smaller threshold for smoothing purposes, CC analysis is performed to extract the foreground blocks from the obtained image.

Kise et al. [146] applied *Voronoi Diagrams* [25] to the task of page segmentation. These diagrams are calculated from an initial set of so-called center points and consist of a number of regions each one stretched around a center point. The regions are defined by the property that all points within a region are closer to the center point of their region than to any other center point. In order to utilize Voronoi diagrams for the decomposition of a document into regions, the aforementioned set of center points has to be determined. This is achieved by extracting sample points from the boundaries of the CCs in the page image and performing noise removal by applying several heuristics. Next, a Voronoi diagram is calculated from the sample points. After removing Voronoi edges that cut through CCs the final segmentation is obtained by removing surplus Voronoi edges. These are mainly located between characters, words, and lines and can be identified by simple rules based on the distances to other edges and the resulting area ratios.

The *DocStrum* method [209] clusters CCs by applying a K-Nearest Neighbours (KNN) approach. After a noise removal step the remaining CCs are classified either as belonging

to the dominant font size or to headings based on a character size ratio factor. Then, for each CC, the K nearest neighbours are detected and a histogram containing the distances and angles between each CC and its corresponding neighbours is computed. Next, the dominant skew in the document can be derived from the histogram by selecting the peak of the angle histogram. Next, the skew value is used to search for nearest neighbour pairs within suspected text lines. Finally, these pairs are combined to text lines by calculating their transitive closure and merged into text blocks by considering their vertical distances.

A variety of approaches is based on a simple description of the background of a page, also known as *Whitespace Analysis*. Initial appealing results were achieved by Baird [29]. In a first step maximal white rectangles (so-called *covers*) are detected whose union provide a complete cover for the background of the document image. For the comparison described below this step was performed by applying the refined method proposed by Breuel [53] that allows for an efficient calculation of a complete coverage of the background of a document. In the process not only the alleged optimal solution is returned but also a list of rectangles sorted by quality in descending order. The obtained covers are sorted according to a sort key using a weighting function that takes the covers' area, height, and width into account in order to assign higher weights to rectangles that are likely to be meaningful separators of text blocks. During the second step a sequence of segmentations, i.e. the remaining uncovered area to this point, is generated by combining the identified covers one by one using several heuristics including a trimming rule to prevent narrow blocks from being segmented too early and a stopping rule. The final segmentation is obtained by extracting the CCs within the remaining uncovered segments of the document.

The so-called *Constrained Text Line Detection* was proposed in [51] and also utilized in the skew and orientation detection approach presented in [310] and described above. It operates in three steps: First, a complete cover of the page background is calculated as described for the whitespace analysis approach. Second, column separators and gutters are identified by the evaluating the whitespace rectangles according to their width, aspect ratio, and proximity to CCs that are considered to be text according to their size. Third, a globally optimal solution for text lines is calculated respecting the gutters as obstacles and using the least square method described in [51].

Comparison The approaches introduced above were compared in [262] on the UW3 data set. As an initial error metric the aforementioned text line accuracy (see [176]) was used but then replaced by the pixel-based approach introduced in [263] and described above (cf. Section 2.3.1.1). After optimizing the required parameters on a separate training set, generally speaking, the Voronoi and Docstrum methods yielded the best results, followed by the constrained text line detection and whitespace approaches, while X-Y-Cut and RLSA performed worst. However, it was noted that the number of columns and the condition of the pages had a major impact on the quality of the segmentation. Furthermore, it was shown that the different methods revealed different strengths and weaknesses, leading to the conclusion that a combined approach might be promising.

2 Background and Related Work

After performing further extensive evaluation the authors arrived at the the following conclusions and recommendations:

- The X-Y-Cut algorithm represents a good choice when dealing with clean documents with next to no skew due to its simplicity and speed.
- When a parameter optimization is feasible, for example when dealing with large collections of similar documents with respect to resolution, layouts, font size, etc. but with a varying degree of noise, the Docstrum and Voronoi approaches are promising.
- For heterogeneous collections consisting of documents with different resolutions and font sizes the constrained text line finding algorithm is recommended since it is virtually parameter-free.
- To deal with non-Manhattan layouts the Voronoi algorithm can be a good choice or at least serve as a valid starting point.

2.3.1.3 State-of-the-Art

Connected Component Grouping In 2013, two competitions on layout analysis on historical newspapers [18] and books [17] led to the insight that the proposed methods all leaned towards a certain methodology where CCs (mostly letters) are combined bottom-up utilizing information about the page background. The approach that yielded the best results in both categories is described by Wei et al. [314]. After extracting the foreground CCs very large ones are filtered out and the remaining ones are assembled into horizontal chains (mostly text lines). Then, column separators are detected by calculating and filtering background rectangles between each CC and its right neighbor. The initially excluded large CCs are then classified as graphic, border or text, by mostly taking their shape and relative position to the detected text lines into consideration. Based on this classification the CCs are then either deleted or incorporated into the text. Finally, the detected lines are merged into text blocks by utilizing their own vertical and horizontal position as well as the one of their neighbors.

The *Tab-Stop-Algorithm* developed by Smith [275] relies on the whitespace boxes that serve as margins, column separators, and indentations. In more modern material these marks used during the formatting of a document are referred to as *tab-stops* giving the algorithm its name. First, very large and very small CCs are filtered out as they might hinder the upcoming operations. Then, CCs get extracted which were classified as left or right tab-stop candidates based on their neighborhood. Afterwards, these are first combined to lines, then to columns or regions. Finally, the detected regions are sorted into a reading order by using a top-down approach. Furthermore, each region can be assigned a type based on their horizontal overlapping with columns. For example, regions which entirely cover several columns are marked as *heading*.

The algorithm participated in the 2009 edition of the ICDAR competition on page segmentation [19] and was evaluated on a data set comprising contemporary material

with a focus on magazines and technical journals [15]. With a pixel-based F-measure result of 91.04% the method achieved comparable results to various other participating approaches, including ABBYY Finereader 8.1 (91.90%) which was used for comparison. Unfortunately, to the best of our knowledge no more up to date results have been published.

Fine-grained Semantic Distinction Thus far, the proposed methods mainly settle for a text/non-text separation and refrain from a fine-grained semantic distinction. This far more challenging task gets considerably less attention in the literature despite the semantic markup of regions being essential for many applications. The following two approaches were applied to already optimally segmented text blocks, consequently only allowing the evaluation of the type classification.

Wang et al. [313] introduced a statistical approach using a 25-dimensional feature vector to represent a text block. Among others, features like the background structure of the document, geometrical information about the text block itself, and the number and properties of its comprised glyphs were utilized. A decision tree classifier allowed distinguishing between nine segment classes: text with font size $\leq 18pt$, text with font size $\geq 19pt$, equations, tables, halftones, drawings, borders, logos, and others. On the UW3 data set the correct type was predicted for 98.45% of the regions.

A slightly better result of 98.56% was obtained by the method introduced by Keyzers et al. [141] who mainly used histograms in order to describe the text blocks. These histograms stored information about the run length of the fore- and background pixels, the width and height of the CCs and distance of a CC to its next neighbor. For classification a KNN classifier with $K = 1$ was used. Additionally, a slightly worse (2.1% error rate) but significantly faster maximum entropy [34] classifier was tested. For the most part, the same classes as defined by [313] were used. However, there was no distinction between font sizes and an additional noise class was introduced.

Naturally, a combined approach which first detects regions and performs a semantic classification afterwards poses a considerably harder challenge.

Pixel Classifiers often with deep learning components, present an approach which differs significantly from the methods described above. To evaluate the following approaches the PA or FgPA (cf. Section 2.3.1.1) were used.

An unsupervised method to learn features using convolutional autoencoders was proposed by Chen et al. [69]. The learning process can be divided into three steps: First, small image sections (5x5 pixels) are used to learn a feature mapping. Next, the obtained mapping serves as a basis to learn a feature representation of higher order from bigger sections (15x15 pixels). The third step is analogous to step 2 but introduces an overlap of 5 pixels. The method was applied to three handwritten documents [70] and distinguished between four classes: periphery, background, text, and decoration. Based on the learned

2 Background and Related Work

representation a Support Vector Machine (SVM) [43] was trained resulting in a PA of up to 97.66%.

In [68] Chen and Seuret combined feature learning and classifier training into a single step resulting in an end-to-end method. The utilization of superpixels, i.e. an arbitrary shaped but fully connected group of pixels that share common characteristics [228], led to improved results and a significant speedup.

Wick and Puppe [317] proposed a Fully Convolutional Network (FCN) adapted from the *U-Net* [244] consisting of several convolution, pooling, and deconvolution layers but refrains from using skip connections. The main advantage of processing the entire input image in one step is that the network can profit from all the available information, for example the absolute position of marginalia with respect to the whole page, allowing the approach to not only deal with a comparatively simple task of distinguishing between periphery, background, text, and decoration but perform a fine-grained semantic distinction between a wide variety of layout types, if properly trained beforehand. No preprocessing steps such as superpixels were used, resulting in an significant speedup of a factor of up to 10 and comparable or improved results compared to the approaches introduced above.

dhSegment The dhSegment framework (cf. Section 2.2.1.3) compared its approach on the DIVA-HisDB data set (cf. Section 2.1) in the context of the ICDAR2017 competition on layout analysis for challenging medieval handwritten documents [269], i.e. the pixel-precise labeling of scanned pages using four classes: text, decoration, comments, and background. The original images were not resized. Patches of 400x400 pixels were used for two of the three books, rising to 600x600 for the third one because of a higher initial resolution. In the postprocessing step binary masks for all predicted classes were obtained and cleaned by removing small CCs. In addition, the output of the page detection task described above was incorporated in order to remove false positive text region detection in the border area of the image. Despite only slightly adapting their generic method to the peculiarities of the task, the proposed approach achieved very competitive results compared to the highly specified systems participating in the challenge, beating six out of the seven submissions with an IoU score of 0.9435 only slightly behind the winning system (0.9490).

2.3.1.4 Tools

Compared to the variety of proposed approaches, only a small number of methods is available as ready to use tools which can be directly applied by the user, preferably via a comfortable to use GUI.

Aletheia [74] is a highly functional proprietary tool developed by the Pattern Recognition & Image Analysis (PRImA) research group [224] of the University of Salford. It offers a variety of tools for the supported manual distinction and adjustment of regions within a document. However, it simply applies the Tesseract page segmentation functionality (cf.

the Tab-Stop algorithm in Section 2.3.1.3) in order to obtain an automatically generated segmentation result. Its manual editing features are very extensive, but due to its closed-source nature there is no community contribution except by feature request, no guarantee of long term availability and it comes with a fee.

Garz et al. [104, 105] propose an interesting semi-automatic system to segment very challenging ancient handwritten documents by utilizing document graphs and pen-based scribbling interaction by the user. The initial graph for a page is created by first producing a sparse graph representation of the foreground by extracting nodes along gradient changes [103] and determining edges by triangulating these nodes. Then, a Minimum Spanning Tree (MST) is calculated using Kruskal’s algorithm [152]. As for the user interaction, the general idea is to allow the users rather imprecise but natural interactions and retrieve a precise segmentation from the graph. Interactions such as the insertion of edges between CCs of the same layout element and the deletion of edges that connect two different layout element are strokes drawn with a pen-based input device. Experiments showed that, despite being evaluated on very challenging documents, the initial graph representation already offers promising results and usually only a few user interactions are required to arrive at the desired output. Drawbacks of the method are the need for a suitable input device and the expensive computation of the MST which was reported to take up to 25 seconds for a single page.

The open-source software Agora [226] was developed as part of the PARADIIT project [227] of the University of Tours. The proposed approach utilizes information about the CCs, in order to construct a map for foreground and background respectively. Based on these maps, the user can interactively define so-called scenarios which are based on rules and allow to adapt to a given document. The software is Windows-only and the project appears to be currently dormant.

The SCRIBO module [157] of the OLENA platform [86] is an open-source layout analysis framework which finished second at the 2011 competition on historical book recognition [16]. While it clearly has its strengths because of its modularity and flexibility, it does not seem to be suited for the needs of the average user who is looking for a stand-alone tool which can be used right away.

N-Light-N [257] is a highly adaptable Java framework which allows the comfortable application of deep neural networks, especially convolutional auto encoders. It has already been used for the segmentation of historical prints by applying a pixel classifier. However, the necessary time expenditure for high resolution documents seems to be too high for practical use.

Implementation in OCR Workflow Tools OCRopus 1 offers no explicit region segmentation but the highly performant line segmentation incorporates some implicit text/non-text distinction capabilities as well as a decent column segmentation which will be explained in greater detail during the introduction of the OCR4all workflow in Section 5.3.

2 Background and Related Work

Tesseract’s page segmentation approach relies on the Tab-Stop-Algorithm developed by Smith [275] (cf. Section 2.3.1.3). Furthermore, the integrated table detection based on the algorithm of Shafait und Smith [265] is worth mentioning.

2.3.2 Line Segmentation

Not only because of the fact that modern text recognition systems operate on line-level, a precise and dependable line segmentation is a key ingredient for every OCR system. While the segmentation of modern printed documents like newspapers, journals, magazines and business letters has been tackled and mostly solved several decades ago [96, 209, 320] and modern commercial tools can be expected to yield almost perfect results on most contemporary printed material [16], the processing of historical printings or even handwritten documents presents a far more difficult challenge. Again, as discussed during the Introduction, this is mostly in consequence of aging effects like degradation including bleed-through, holes, and spots as well as the much more irregular layouts suffering from varying inter line distances and font sizes, marginalia, ornaments, swash capitals, and decorations [10, 167]. However, dealing with this demanding task represents an active field of research as shown by the publication of public data sets like the IAM-database [181] or DIVA-HisDB (cf. Section 2.1) and regularly organized challenges allowing researches to evaluate and compare their newly developed methods [269].

In this section we present and discuss different approaches proposed to deal with the task of line segmentation. Of course, our main focus lies on the processing of historical printings, but since the task of line segmentation on handwritten historical documents is quite similar, is tackled by a very comprehensive and active community, and can be considered even more challenging, we also include methods developed for the application on handwritten documents. Because the OCR4all workflow stops at line level, we mostly take methods specifically geared towards line segmentation into account. However, some of the mentioned approaches also provide word segmentations by default or can easily be extended to do so. We ignore methods that aim towards character segmentation since they are beyond the scope of this work (we refer to [65] for a very comprehensive survey).

In the following, we first provide a brief summary of important terms and concepts before we focus on different groups of text segmentation approaches. While doing so we mostly follow the concepts and grouping suggested within the highly recommended survey by Likforman-Sulem et al. [167] and build from there.

2.3.2.1 Fundamentals

Before dealing with different segmentation approaches we want to lay the groundwork by introducing some important definitions and ways to represent text lines.

Definitions According to [167] the following text line components can be defined (also cf. Figure 2.1):

- *Baseline*: A fictitious line touching the bottom of the character bodies (without descenders).
- *Median line*: A fictitious line on top of the character bodies (without ascenders).
- *Lower line*: A fictitious line touching the bottom of the descenders.
- *Upper line*: A fictitious line lying on top of the ascenders.
- *Overlapping components*: Descenders or ascenders of an adjacent line which overlap with the region of the current line.
- *Touching components*: Descenders or ascenders that connect consecutive lines.

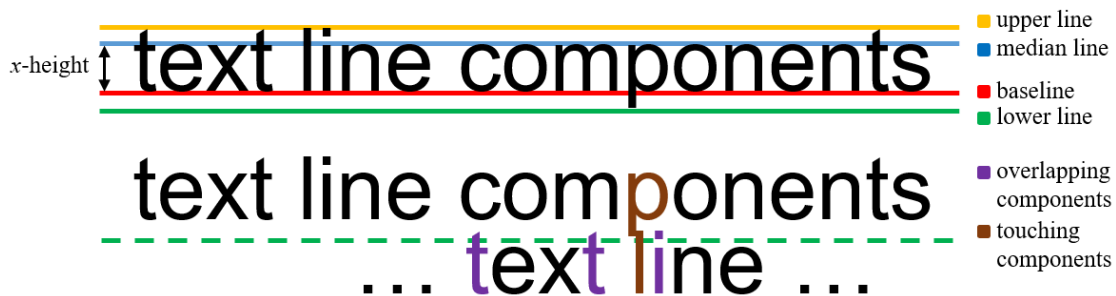


Figure 2.1: Visualization of text line components.

Furthermore, they define the process of text line segmentation as a labeling process that allocates spatially aligned units, e.g. pixels, CCs, or other characteristic points to a text line by assigning the same label.

Text Line Representations Not only identifying a text line and/or its components is a challenging task but also the representation. There are several ways with varying complexity and accuracy (altered and extended from [167]):

- *Separating paths, delimited strip, and polygons*: A separating path is a continuous line which separates the area of a text line from the vertically adjacent one. Two consecutive paths, one at the top and one at the bottom of a text line comprises the so-called delimited strip which represents the text line. When text lines span the entire page or previously defined segment horizontally this representation is sufficient and a polygon representation can trivially be obtained. If the left and right borders are required as well, the two separating paths are completed by two further paths resulting in a text line polygon.
- *Clusters* consist of units that were assigned to the same text line. Units may be individual pixels, CCs, or even blocks (rectangles, polygons, ...) that contain various amount of writing. Each cluster represents a text line encoded by the same label

2 Background and Related Work

assigned to each unit of the cluster/line. Again, deriving a polygon or bounding box representation is a straight forward task.

- *Baselines* only store information about the position of the lower parts of the character bodies of a text line as defined above. Naturally, these lines are easier to detect than for example entire polygons which fully surround the entire line but consequently also lack some information.

Since an OCR workflow does not stop after identifying the text lines, some representations require further processing before they can be passed into the component of the workflow, most likely an ATR engine. Modern character recognition approaches require a rectangular line image as their input which can quite easily be created from a polygon or cluster-based representation by simply copying the parts considered relevant into a blank rectangle of a fitting size. When a baseline representation was used, it is apparent that the extraction is not as trivial and cannot be as precise. This problematic and especially its effects on the ATR quality were examined in [243]. The experiment was performed on the C5 Hattem Manuscript [249], a 15th century single writer document with a relatively simple one column layout but also with small distances between adjacent text lines and consequently overlapping or even touching ascenders and descenders of consecutive lines on a regular basis.

Three different segmentation techniques resulting in three different line representations were applied:

- A tightly fitting *polygon*, representing the most informative approach, was created by applying the algorithm introduced in [170].
- The so-called *poly-baseline* was obtained by applying the method presented in [41]. Simply adding a few pixels above and below the localized baselines allowed for a straight forward extraction of the text lines.
- Analogously, the most simplistic representation, a (straight) rectangle was used to extract the text line again by just adding some pixels to the top and bottom of the detected *straight-baseline* as proposed by [40] and [37].

To ensure a fair comparison all results were manually corrected which of course is far more time consuming when dealing with polygons instead of baselines, especially straight ones. In this case, the authors estimated the required manual effort as one hour per page for the polygon solution, while correcting the poly-baseline (15 minutes) and straight-baseline (5 minutes) representations is significantly faster.

In order to determine the influence of different text line representations on the recognition accuracy a Hidden Markov Model (HMM) based approach was chosen and trained on the material at hand (for details we refer to the original paper [243]). Several cross-validation experiments resulted in quite similar CERs for the approaches polygon (15.8%), poly-baseline (17.3%), and straight-baseline (18.1%) with overlapping 95% confidence intervals. Combined with the vastly differing amount of human interaction required to obtain a perfect segmentation result, this led the authors of the study to the conclusion that the

poly-baseline method represents a good trade-off between effort and accuracy and that only a slight gain in ATR accuracy can be obtained if a tightly fitting polygon is used to represent the text lines. While this is mostly true for the experiment at hand further studies are required in order to allow a more general conclusion mainly because of two reasons: First, while, due to the overall high CER, the differences in recognition accuracy may be considered irrelevant or at least justifiable, it has to be seen how the different representations influence the results when dealing with material, for example historical printings, that allows to reach CERs below 5% or even 1%. Second, the influence of the manual correction step is not clear. Maybe a considerable portion of the time invested in optimizing the tightly fitting polygon had next to no influence on the recognition results despite being very costly.

We conclude that the described experiment does not provide enough insights to consider the modalities of the representation of text lines to be negligible. Consequently, we start from the premise that, especially when aiming for extremely low error rates, text lines should be represented as accurate as possible, namely as a pixel-based labeling or a tight-fitting polygon.

2.3.2.2 Basic Approaches

Before focusing on selected state-of-the-art methods we provide a brief overview over typical approaches [167] that have shown to be useful in the past and regularly provide a starting point for new methods.

Projection profiles techniques [175, 179, 327] sum up the pixel values from each column along the horizontal axis obtaining clues for gaps between text lines. Naturally, this works best with lines with next to no overlap or even none at all.

Smearing methods [159, 160, 267] rely on the RLSA proposed by [320] and introduced in Section 2.3.1.2. After smearing black pixels along the horizontal direction by flipping the white pixels in between them according to predefined thresholds the text lines can be extracted according to the resulting CCs.

Grouping-based techniques [93, 165] aim to create alignments of units (pixels, CCs, etc.) by following a bottom-up strategy. Usually this requires a quality measure to master the challenges of planting sensible seeds, define rules for the addition of further units, and to solve conflicts in the case that a unit may well fit several alignments.

As always when the detection of lines is required, the *Hough transformation* [129] represents a valid choice. Of course, this also holds true for the task of text line detection [166] and has even been successfully applied to the detection of considerably fluctuating lines [225]. In general, the Hough transformation is used to detect imperfect arbitrary shapes in a binary image resulting from an edge detection. The main idea is to perform a voting procedure to extract maxima from the so-called *parameter space* which marks parameter values that explain the evidence (an edge pixel) in the binary image.

2 Background and Related Work

The *Seam Carving* algorithm [26] was originally developed to enable a content-aware resizing of images and was successfully adapted to the task of text line detection. Basically the idea is to compute seams that separate two consecutive text lines with minimal cutting through the components of the two lines, or optimally even no intersections at all. To achieve this, an initial energy map has to be calculated treating the line components as high-energy regions and the space between the line as low-energy regions. After determining suitable starting spots for the seams, for example by using projection profiles, the separating seams can be calculated. The initial approach [248] performed a distance transform [39] operation on a binary image to generate energy maps. However, it has been extended to grayscale and color images since [23, 24].

2.3.2.3 State-of-the-Art

As expected, all three approaches selected for this section incorporate a substantial amount of deep learning. The first two have just been presented at ICDAR2019 and the third one is the already introduced dhSegment.

ARU-Net Grüning et al. [118] propose a two-stage method. First, a pixel labeling into three classes (baseline, separator marking the beginning and end of lines, and other) is performed by an extension from the U-Net [244] called ARU-Net which, due to data augmentation techniques, can be trained with a manageable number of GT images and whose training framework is open-source [22]. The second step uses the labeling from the first step as input and performs a bottom-up clustering to build baselines using superpixels³ in the process. Extensive evaluations showed the capabilities of the system as it considerably outperformed all submissions of the two latest ICDAR2017 competitions on layout analysis [269] and baseline detection [84]. Further evaluations proved the effectiveness of the data augmentation techniques and its applicability to curved and arbitrarily oriented text lines.

Labeling, Cutting, Grouping Alberti et al. [10] integrated deep-learning based pre-classification into their open-source [155] workflow and combined it with state-of-the-art segmentation methods. In a first intermediate step a precise segmentation of the RGB input images at pixel-level is performed which mainly acts as a denoising tool. The vanilla ResNet-18 [9] architecture was chosen and run within the DeepDIVA deep-learning framework [8]. After cleaning the output using simple postprocessing techniques like removing small CCs, all pixels labeled as non-text, for example pixels belonging to decorations, are discarded for further processing. Next, an optimized seam carving based algorithm is applied and a simple postprocessing step allows to extract the detected text lines represented by their enclosing polygons. Evaluations on the DIVA-HisDB data set (cf. Section 2.1) showed a nearly perfect IoU score of 99.42%, reducing the error of the

³Cf. the pixel classifier-based region segmentation approach of Chen and Seuret [68] introduced in Section 2.3.1.3.

earlier state-of-the-art system (97.86% [269]) by over 80%. Since closer investigation revealed that all of the few remaining errors were caused by misclassifications during the semantic labeling check, another experiment was performed using the pixel-level GT instead of the prediction, resulting in a perfect score of 100%. Further experiments showed a high parameter robustness and indicated the applicability to other data sets. Despite the very promising results the authors of the proposed approach pointed out some restrictions that have to be addressed: First, some assumptions on the to-be-processed documents have to be made including that the text lines on a page do not change their direction and the general structure is a standard one or two column layout. Otherwise a preceding text block segmentation would have to be performed. Finally, the method relies on a close to perfect pixel-level semantic segmentation which is expensive, both in terms of computational and also human effort, since books-specific training is most likely required.

dhSegment To evaluate the performance of dhSegment (cf. Section 2.2.1.3) in the area of line segmentation the method was compared to the submissions of the ICDAR2017 competition on base line detection using the cBAD data set (cf. Section 2.1). For this task the network was trained to label all pixels within a radius of five pixels of the GT baselines. The postprocessing step consisted of applying a Gaussian filter to the probability map produced as output and a subsequent hysteresis thresholding, before extracting the CCs from the binary mask and converting them into line polygons. For the two tracks treated in the competition the generic dhSegment approach once again achieved very competitive results placing second and first respectively among the six proposed methods [84].

2.3.2.4 Implementation in OCR Workflow Tools

OCRopus 1 incorporates a CC-based approach incorporating grouping and smearing techniques while applying the y -derivative of a Gaussian kernel while relying on several heuristics. A slightly extended version of it is utilized by OCR4all and consequently will be explained in detail during the introduction of the OCR4all workflow in Chapter 5 (cf. Section 5.5.2).

Tesseract 3/4 relies on a grouping-based technique introduced in [273] whose key steps consist of CC filtering and line construction [274]. Similar to OCRopus 1 it first uses a simple percentile filter that removes CCs that are considerably bigger (for example swash capitals or vertically connected characters of adjacent text lines) or smaller (for example punctuation, diacritica, or noise) than the median height within a text region. The filtering step then allows for a much easier assignment of the remaining CCs into unique text lines. After the baselines have been estimated by applying a least median of squares fit [246] the previously removed CCs are fitted back into the detected lines. Finally, a clean-up step deals with the optimization of the assignment of diacritical marks and the association of parts of broken characters.

2.4 Automatic Text Recognition

After introducing the historical development in the area of character recognition techniques, we briefly discuss advantages and disadvantages of mixed models compared to book-specific training and finally highlight the recognition capabilities of several available ATR engines.

2.4.1 Historical Development and State-of-the-Art

Text recognition can be considered as one of the earliest computer vision tasks [251]. Therefore, we begin our survey of related work with a short sketch and the historical development of the general recognition approaches.

2.4.1.1 Glyph-based Recognition

For a long time segmenting printed texts into single glyphs which are then classified individually was considered the go-to ATR approach. After identifying a glyph a feature extraction step takes place before the gathered information is utilized to assign a character class. This approach was used by all available ATR engines until very recently, for example by the open-source [295] ATR engine Tesseract [274] before version 4.0.

The main drawback of this method is the need to precisely identify every single glyph. This can be a very challenging task especially when dealing with older printings where the segmentation step leads to either splits or merges of glyphs on a regular basis due to various reasons: The glyph contours have lost their uniform ink impression and get segmented as individual pieces, or contours of neighboring glyphs have become fuzzy and tend to touch each other leading to segments containing several individual glyphs that cannot subsequently be classified. Furthermore, creating GT data for training a recognition model based on real printings (as opposed to train on synthetical images from existing computer fonts) is a cumbersome and time consuming task. Still, Kirchner et al. [145] showed that it is basically possible to train book- or rather type-specific models with Tesseract 3 using Aletheia [74] and Franken+ [99]. After manually identifying examples for each glyph class Franken+ supported the creation of the required Tesseract 3 training format. A model trained on an incunabulum was then applied to other books of the same print shop using the exact same type and resulted in CERs between 4% and 8%. However, due to the high amount of manual work required to produce such a model, this approach seemed only practicable if one desires to OCR a variety of works printed with the same letter types.

2.4.1.2 Line-based Recognition using LSTM Networks

In 2013 Breuel et al. introduced a segmentation-free approach (no segmentation beyond the line level) in their ground-breaking paper [46] which is capable to recognize entire text

line images at once. This is possible by utilizing recurrent neural networks with an Long Short-Term Memory (LSTM) architecture trained using the Connectionist Temporal Classification (CTC) algorithm. After resizing a line image to a fixed height, the image is cut into vertical stripes with a width of one pixel. The pixel values of these stripes (usually binary or grayscale) are fed into the neural network which produces a probability distribution over the entire glyph alphabet for each stripe, usually by processing the input sequence two times: from left to right and from right to left (bidirectional LSTM). Finally, the output sequence is generated by applying a CTC decoder.

The line-based ATR approach not only outperformed the glyph-based approach considerably, but also offers the advantage of a much easier GT production and training process. Lines chosen for training can simply be transcribed as a whole since a line image and the corresponding transcription completely suffice to serve as a training example without the need for any further information about glyph positions or bounding boxes. Those improvements also enabled an efficient high quality processing of even the earliest printed books as shown by Springmann et al. with individually trained models achieving CERs around 2% [279, 280].

2.4.1.3 Line-based Recognition using CNN/LSTM-Hybrid Networks

A further refinement of the LSTM approach was introduced in 2017 by Breuel [58] who added CNNs, which showed to be very effective in a variety of image processing task [174], as additional layers in front of the LSTM. Each CNN performs a convolution of the original line image using different filters whose parameters are learned during the training process producing a feature map that highlights the most descriptive parts of the input image. After a pooling operation the resulting images are then either passed into another CNN or vertically concatenated and passed into the LSTM layer. This CNN/LSTM-hybrid method has shown to be very successful in various application scenarios and therefore also represents the current state-of-the-art of modern ATR engines like Calamari, Tesseract (since version 4.0) and OCRopus 3. Since Calamari, as we will show below, is the most successful of the currently available open-source engines we introduce it in the following in greater detail before we discuss evaluations and comparisons in the next section.

2.4.1.4 Calamari

Calamari [318, 319] is still under active development [63]. Apart from its superior recognition capabilities compared to other open-source engines it offers a clean Application Programming Interface (API) and Graphical Processing Unit (GPU) support for rapid training.

Furthermore, Calamari natively supports three techniques developed or adapted during the course of this thesis, resulting in higher recognition rates which we will briefly explain in the following before discussing them in detail in Chapter 4: First, a *cross fold training*

2 Background and Related Work

procedure with subsequent confidence voting in order to reduce the CER on early printed books was implemented (see [234] and Section 4.1). By dividing the GT in N different folds and aligning them in a certain way, it is possible to train N strong but also diverse models which act as voters in a newly created confidence voting scheme. Second, the so-called *pretraining* functionality allows to build from an already available Calamari model instead of starting training from scratch which not only speeds up the training process considerably but also improves the recognition accuracy (see [236] and Section 4.2). In addition, *data augmentation* is supported using the routines of *ocrodeg* [203] for generating noisy variations of training material.

Moreover, Calamari provides several interfaces for more complex data representations than image/text pairs on line level, most notably PageXML (cf. [223] and Section 2.7.1). Combined with the highly modular structure, this ensures a straight forward integration into existing and future OCR workflows. Consequently, we decided to use Calamari as our primary ATR engine to implement and evaluate further developments and also rely on it as one of the core submodules in OCR4all.

Before discussing the performance of Calamari in comparison to several other open-source ATR engines we now go into detail regarding technical specifications and the provided helpful features for the end user.

Technical Specifications and Network Architecture The Calamari ATR engine is implemented in Python 3 and relies on the open-source machine learning library Tensorflow [1] for the underlying deep learning tasks. In the following, we briefly discuss its default network structure and core ATR functionalities.

Calamari implements CNN/LSTM hybrid networks whose exact topology can be easily specified by the user via a custom model description language. However, in our experience the default network architecture represents a reasonable choice for the vast majority of use cases and consequently was never changed during the experiments presented in this work. Figure 2.2 shows the default network and explains the task of the individual layers as well as their input-output relations.

Preprocessing For both applications, training and recognition, the networks expects the input to fulfill certain criteria whose adherence are ensured during the preprocessing step. Line images (color, grayscale, or binary) are automatically rescaled to a fixed height to fit the input layer of the network. The default height of 48 pixels proved to represent a sensible trade-off between simplicity (allowing the network to generalize well) and complexity (conserving all the important information) and therefore was always kept during our experiments.

As for the textual preprocessing of the GT before the training, Calamari performs some optional standard steps like the collapsing of whitespace chains and the removal of leading and trailing whitespaces. Additional regularizations like the splitting of ligatures can be applied by choosing from several predefined rule sets or by defining a customized one.

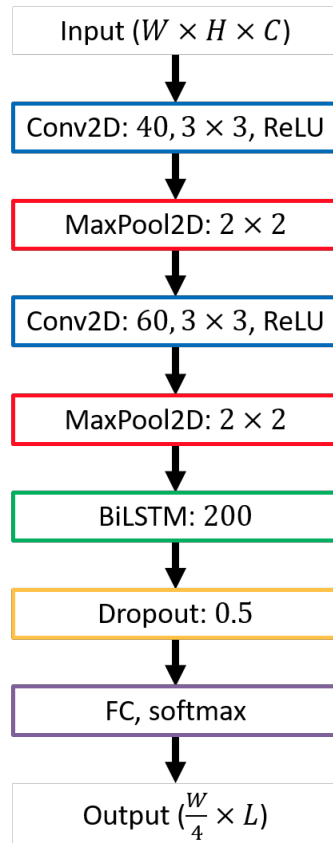


Figure 2.2: Calamari’s default network architecture: To begin with, the input image with width W , height H , and C color channels is passed into the first convolutional layer. Applying 40 filter operations with a kernel size of 3×3 results in 40 feature maps which are then reduced by a max pooling operation with a pool size of 2×2 . The two steps are then repeated but this time 60 features are used during the convolution. Next, the feature maps are vertically concatenated and passed into a bidirectional LSTM with 200 hidden nodes. A dropout layer with a dropout rate of 0.5 is introduced to reduce the effect of overfitting. Finally, for every horizontal position a fully connected layer with softmax leads to the final output probability matrix with $\frac{W}{4}$ columns (the original width of the input image is reduced by factor 4 because of the two pooling operations) and L rows, with L representing the number of labels, which is the alphabet size plus the blank label. Adopted from [229].

2 Background and Related Work

Training Calamari expects training examples, in the form of pairs of line images and their corresponding transcription, which can either be provided as raw image and text files on line level or as a combination of the preprocessed page images and an additional XML file, like PAGE or ABBYY XML [5], containing the required coordinate and textual GT information. After loading and preprocessing the data, which can be done at once before the training commences (preloading) or continuously during the training process (on-the-fly) the data can be divided into a training and a validation set. The latter is used to measure the recognition accuracy on unseen data, enabling Calamari to determine the best model and the correct moment to stop the training. This early stopping functionality as well as the optional data augmentation procedure are discussed in greater detail in context with the implementation in OCR4all in Section 5.6.2.

The actual training process uses the CTC loss function [112] which maximizes the probability of the given GT label sequence based on the probability output of the network. As a default the Adam solver [144] is used with the learning rate set to 0.001. Further standard techniques like dropout [283], to prevent the network from overfitting, and gradient clipping [218], to deal with the exploding gradient problem, are implemented.

Prediction To perform the recognition, Calamari requires line images, again provided either as raw line image files or as a combination of a page image and an XML file, as well as one or several models as input. If more than one model is used, confidence voting is automatically activated and the individual outputs are combined. Depending on the data input the ATR result is then output as a text file or placed at the corresponding position within the input XML file. Apart from the raw ATR result Calamari also allows the optional output of *extended prediction data* which we will make use of throughout this thesis: This data includes the position of each character which are identified solely based on the output of the network. Additionally, confidence values for the most likely character as well as for all noteworthy alternatives are provided.

2.4.2 Evaluation of ATR Engines

In the following, we compare existing ATR engines and identify missing features which we later address with our own solution based upon Calamari (recognition accuracy and speed), mixed models, training on historical data, and the necessary tooling for a complete workflow.

2.4.2.1 Book-specific Training on Early Printed Books

A thorough comparison of a shallow LSTM (OCROPUS 1) and a deep CNN/LSTM hybrid (Calamari) is given in [319]. Three early printed books, printed between 1476 and 1505 in German and Latin, were used as training and evaluation data. Book-specific models were trained using 60, 100, 150, 250, 500 and 1,000 training lines. The results showed, as anticipated, that the advantage of the deep network grew with an increasing number of

lines used for training, yielding an average improvement in CER of 29% for 60 lines and 43% for 1,000 lines. As for the training and prediction times the deep network, despite having a considerably more complex network structure and more trainable parameters resulting in a higher number of necessary operations, outperformed the shallow one when using four CPU threads or more. The reason for this are the pooling layers which reduce the dimensions of the image by a factor of 4 leading to a considerably speedup during the expensive LSTM and CTC operations. Running the training and prediction on a GPU led to further speedups by a factor of at least six and four, respectively.

2.4.2.2 Modern English and 19th Century Fraktur

In [318] Wick et al. compared Calamari, OCRopus 1, OCRopus 3, and Tesseract 4 on two public data sets: First, the UW3 data set consisting of lines from the late 20th century printed in Antiqua [308]. Second, the DTA19 data set which is a part of the GT4HistOCR corpus⁴ [119, 282] containing 39 German novels printed in Fraktur during the 19th century.

Again, book- or rather corpus-specific training was performed with each engine using the respective default parameters. On the UW3 data set Calamari achieved a CER of 0.155%, considerably outperforming OCRopus 1 (0.870%), OCRopus 3 (0.436%), and Tesseract 4 (0.397%). The inclusion of cross fold training and confidence voting (cf. Section 4.1) improved Calamari's result by another 26% to a CER of just 0.114%.

Evaluations on the DTA19 data set led to similar observations with Calamari reaching a CER of 0.221% (0.184% with voting) compared to the significantly higher 1.59% of OCRopus 1 and 0.907% of OCRopus 3. On this data set no Tesseract 4 results were reported.

Regarding speed the average time needed to train or to predict a single line of the UW3 data set was measured and compared. When using a GPU Calamari required 8 ms to train and 3 ms to predict a line which proved to be considerably faster than OCRopus 3 (10 ms and 7 ms), while OCRopus 1 (850 ms and 330 ms), and Tesseract 4 (1,200 ms and 550 ms) are far behind due to their lack of GPU support.

2.4.2.3 Conclusion

Based on the results presented above and our personal experience Table 2.2 sums up and rates the capabilities of the most important available ATR engines in terms of historical ATR. For a more in-detail comparison also taking model design options and hyper parameters into account we refer to [28].

As for the recognition the main criteria are accuracy and speed. Since we consider postprocessing using dictionaries and language models to be an individual step in the

⁴For further information please see Section 3.2.1

2 Background and Related Work

Table 2.2: Comparison and rating of the capabilities of four modern ATR engines. More ticks (✓, to a maximum of three) denote a higher ability/applicability of an ATR engine in the respective step. A cross (✗) shows that an ATR engine does not support a step at all.

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

workflow, we rate the raw recognition capabilities of the engines. Due to the results presented above, the best rating goes to Calamari.

Regarding the training, we rate the engines mainly based on their speed and effectiveness but also take into account the user friendliness when it comes to training on real data. OCROPUS 3, Tesseract 4, and Calamari in general allow pairs of line images and their transcriptions as training input which is very comfortable and straight forward for the user. While Calamari can deal with the image/text pairs directly, just like OCROPUS 1, OCROPUS 3 requires to create a .tar file comprising the data. As for Tesseract 4, the training of models on real historical data has been considered at least impracticable for several years until recently a solution was discovered and made publicly available [298]. However, this required an extension to the standard training tools. While it is basically possible to train single glyphs and consequently a book-specific model using ABBYY, this is a tedious and ineffective task which seems to be mainly geared towards the recognition of quite specific ornament letters. This effectively limits the recognition capability to the expensive existing historical models one has to licence from ABBYY.

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

2.4.3 Mixed Models

While the best results can usually only be achieved by training book-specific models, the out of the box application of existing mixed models represents a baseline option that may already be good enough and can be done automatically. However, even a fully automated approach becomes unattractive if it cannot yield satisfactory ATR results. Consequently, the applicability of mixed models considerably depends on two factors: the use case and the material at hand. In fact, the quality of an ATR result directly influences the possible areas of application, for example while critical editions aim for

perfection, tasks like key word search and word spotting can usually deal with (severely) flawed texts. Regarding the to-be-processed material it is apparent that mixed models can deal best with works which are printed in a similar type as the training material on which the models have been trained.

Before we take a closer look at related work concerning mixed models Table 2.3 provides an overview over the results achieved by various ATR engines on different material using mixed and sometimes also book-specific models. The results indicate that the difficulty level rises the older the material at hand gets, consequently increasing the importance of book-specific training.

Table 2.3: Summary of the results achieved by mixed and book-specific models sorted with respect to the used *Material* from older to newer. The early printings are selections from several sub corpora of the GT4HistOCR corpus (cf. Section 3.2.1), the 19th century material comes from different sources including the DTA19 corpus (cf. Section 3.2.1.5) and various works introduced in Section 7.1.2, and the 20th century data is the UW3 database (cf. Section 3.2.3). Apart from the *Source*, we provide the the ATR *Engine* utilized for training and testing as well as the obtained *CER* values.

Material	Source	Engine	Models	CER _{min}	CER _{avg}	CER _{max}
1476-1686	[279]	OCRopus 1	mixed	<1%	8.3%	21%
			book-specific	<1%	2.2%	5.3%
1476-1505	[319]	Calamari	book-specific	1.0%	1.2%	1.5%
1476-1572	[233]	OCRopus 1	book-specific	1.0%	2.1%	2.9%
1487-1870	[280]	OCRopus 1	mixed	<1%	5.0%	17%
			book-specific	<1%	2.1%	5.4%
19 th cent.	[46]	OCRopus 1	mixed	0.15%	0.83%	1.5%
19 th cent.	[235]	ABBY	mixed	0.01%	2.8%	26%
		Calamari		0.01%	0.61%	4.8%
		OCRopus 1		0.17%	1.9%	11%
19 th cent.	[318]	Calamari	mixed	-	0.184%	-
		OCRopus 1		-	1.59%	-
		OCRopus 3		-	0.907%	-
20 th cent.	[46]	OCRopus 1	mixed	-	0.6%	-
20 th cent.	[58]	OCRopus 3	mixed	-	0.25%	-
20 th cent.	[318]	Calamari	mixed	-	0.114%	-
		OCRopus 1		-	0.870%	-
		OCRopus 3		-	0.436%	-
		Tesseract 4		-	0.397%	-

2.4.3.1 Focus on Early Printed Books

Since the variance between printing types started off very high in the incunabula age and decreased over the centuries, it is clear that training wide applicable mixed models becomes the more challenging the older the target material gets. This effect as well as a general comparison of the effectiveness of mixed and book-specific models was investigated by Springmann et al. [279, 280] who relied on OCRopus 1 as their ATR engine:

First, they performed experiments on a corpus consisting of twelve books printed with Antiqua types between 1471 and 1686 with a focus (ten out of twelve) on early works produced before 1600. After dividing the corpus into two distinct sets of six books each a mixed model was trained on both of them. The evaluation of each model on the respective held-out books yielded an average CER of 8.3% with the individual CERs ranging from 21% to below 2%. Only two books scored a CER higher than 10%, both of them incunabula. As expected, training book-specific models and evaluating them on held-out data of the same book resulted in considerably better recognition results ranging from 5.3% to below 1% and an average of 2.2%.

Second, a similar experiment was conducted as part of a case study on the RIDGES corpus (cf. [207, 242] and Section 3.2.1.4) consisting of 20 German books printed between 1487 and 1870 in Fraktur. After applying the same methodology as mentioned above the mixed models scored an impressive average CER of 5.0% with individual results ranging from 17% to below 1%. Similar to the first case study the two oldest books performed worst with CERs over 10%. As a matter of fact, the individually trained models again performed considerably better, reaching an average CER of around 2.1% with the worst book still achieving 5.4%.

In both case studies the oldest printings proved to be the most difficult ones for the mixed models to recognize, as expected. This highlights the general difficulty of using mixed models: To yield high quality results the types at hand have to match (parts of) the material the model was trained on as much as possible. While this becomes more likely when adding more works to the training set, a perfect fit can never be guaranteed.

2.4.3.2 Modern English and 19th Century Fraktur

Breuel et al. [46] trained two OCRopus 1 mixed models to recognize modern English text and German Fraktur from the 19th century. The English model was trained on the UW3 data set and yielded a CER of 0.6% when applied to 1,020 previously unseen lines from the same data set. The training data for the Fraktur model comprised around 20,000 mostly synthetically generated text lines which led to a model achieving CERs of 0.15% and 1.37%, respectively on two books of different scan qualities. To the best of our knowledge the Fraktur (FRK) model is widely considered as the default OCRopus 1 model for Fraktur recognition and therefore, we will use it for comparison in our evaluations (cf. Section 7.1).

2.4.3.3 Multilingual Mixed Models

An approach not only mixing different types but also various languages was promoted by [305]. They generated synthetic data for English, German, and French and used it for training language-specific models as well as a mixed one. As expected, the language-specific models performed best when applied to test data of the same language yielding CERs of 0.5% (English), 0.85% (German), and 1.1% (French). However, recognizing a mixed set of text data with the mixed models also led to a very low CER of 1.1%. Despite being carried out exclusively on synthetic data this experiment indicates a certain robustness of OCRopus 1 (and arguably also of other LSTM-based ATR engines) regarding varying languages in mixed models.

2.5 Postcorrection

In this final main step of the workflow the goal is to further optimize the raw ATR output by identifying and correcting errors preferably without introducing additional errors by doing so. Since we did not deal with this step at all during the work leading up to this thesis, we only provide a very brief overview over the general related work and instead focus on two particularly interesting state-of-the-art approaches including an interactive one. Apart from that exception we only consider fully automatic methods and ignore raw transcription interfaces and crowd-sourcing [187] approaches.

2.5.1 Overview

There is a vast amount of related work available for the field of postcorrection of ATR output. Among others, extensive surveys are presented by Kukich [153] and Dengel et al. [78]. Kukich describes approaches which can be summarized into three main categories: non-word error detection (mostly approaches using n -grams), isolated word error correction (mostly techniques based on rules, dictionaries, or transition and confusion probabilities), and context-dependent word correction (mostly natural language processing related techniques). Dengel et al. treat voting techniques, lexical postprocessing as well as techniques that take the context of a word or even document into consideration.

Naturally, the applicability and effectiveness of an approach highly depends on the material at hand and the quality of the underlying ATR result. For example, methods that heavily rely on the surroundings of a correction candidate or even take the documents context into consideration can be expected to suffer considerably more from a severely flawed ATR result than an approach based on a combination of dictionaries and confusion probabilities. Another important factor is the degree of standardization regarding language and orthography within the document. Of course, a successful postcorrection of e.g. early German, where the lack of any standard can lead to the same word being spelled differently even within the same document, represents a particular challenging task [281].

2 Background and Related Work

As shown in Table 2.3 modern ATR engines can achieve results with very low error rates even on historical materials. On the contrary, the selected solutions presented in the upcoming section build from considerably worse ATR results. Unfortunately, to the best of our knowledge, there are no evaluations available dealing with very low CERs ($\lesssim 2\%$) on historical material. However, we think that the described approaches could well work in combination with considerably better base results.

2.5.2 State-of-the-Art

As mentioned above, an in-detail review of the state-of-the-art in the area of postcorrection would go beyond the scope of this thesis and consequently we focus on the *Post Correction Tool* (PoCoTo) and a language model-based method because they represent especially interesting approaches for our purposes.

PoCoTo The original PoCoTo introduced by Vobl et al. [312] is a system developed to support the efficient interactive postcorrection of historical texts by offering several advanced features: Suspicious tokens of the ATR result are identified by a special language technology which is aware of historical language variations represented by rewrite rules like $t \rightarrow th$ (modern spelling vs. historical spelling) and can be corrected by choosing a word from a list of generated plausible correction candidates. The user does not have to perform this for every single word but can *batch correct* entire error series which for example can consist of identically misrecognized words or words that suffer from the same ATR error, for example the confusion of “e” and “c”. In the batch correction view the system shows all suspicious tokens of an error series in a list of concordances, displaying their neighbourhood in the text and individual correction suggestions which can be applied to all list items in one shot. At any time it is possible to view the corresponding words within the scanned image. Evaluations performed in three major European libraries covering historical German, Spanish, and Dutch showed that even without the batch correction PoCoTo already helps to efficiently correct texts: In a first setting the users were only allowed to use the GUI which highlighted non dictionary words but could not access the batch mode. The full mode was enabled as a second setting. While the users on average were able to correct 3.8 errors per minute in the first setting this number almost doubled to 7.5 in the second setting. The real potential of the software became apparent when a user took advantage of some very productive error patterns, resulting in about 500 corrections in ten minutes.

The continuous development of the system resulted in several improvements on the original approach. In [94] Fink et al. added three major extensions: First, the system was made more adaptive to manual interventions of the user, which are used to simultaneously improve background lexica and estimate the probability of ATR errors and historical patterns. This refinement notably increased the precision with respect to identifying erroneous ATR tokens. Second, the linguistic background resources were extended by new historical patterns which led to a more successful discrimination of historical spelling

from real ATR errors. Third, tokens that could not be interpreted by the model were added to a list of conjectured errors and shown to the user despite no suggested correction being available, naturally resulting in a better error detection recall but also a better precision.

A fully automated extension of PoCoTo was proposed by Englmeier et al. [88]: *A-PoCoTo* is more geared towards the deployment in large-scale digitization projects, can take the ATR results of multiple engines into account, and uses sentence context for its decisions. This fully automatic first step is extended by an interactive postcorrection (resulting in *A-I-PoCoTo*) as a second, optional step in which the user can efficiently confirm, reject, or improve decisions made by the system. For evaluation purposes a postcorrection model was trained on a small corpus comprising 574 pages from 90 documents. Then, only two OCRopus 1 models, i.e. the ones produced in [279] (cf. Section 2.4.3.1), were used to recognize two previously unseen documents from the 16th and 19th century, resulting in quite high Word Error Rates (WERs) of 34.97% and 22.43%, respectively. Despite the manageable amount of training data, the challenging material, and the use of just two comparatively weak ATR models resulting in a low base recognition accuracy, the fully automated application resulted in improved WERs of 30.19% and 19.37%, respectively. Unfortunately, as next to no further results regarding these variables have been reported in the related publication, their influence remains unclear, at least for the moment.

Whereas the original PoCoTo client was a stand-alone Java application it has now been rewritten as a web-based tool⁵ which together with the interactive aspect and the focus on historical texts makes it a very interesting candidate for the integration into the OCR4all workflow.

Integration of Language Models Another interesting approach based on probabilistic n -grams was proposed by Wüthrich et al. in [324] who performed experiments on a manuscript from the 13th century, training not only a HMM recognizer but also a language model. While for modern languages it is possible and sensible to extract the required language information and statistics from comprehensive corpora like the Brown Corpus [98], considerably older language models have to be constructed from significantly smaller amounts of data. In this example, only 4,478 lines of GT were available and were distributed over three distinct sets for training (ca. 50%), validation (ca. 20%), and testing (ca. 30%). All sets combined contained close to 5,000 distinct word classes, of which almost 4,000 were covered within the training and validation set.

Since we want to focus on the language modelling aspect of the experiment, we refrain from describing the actual recognition system in detail and instead refer to the original paper [324] and the therein recommended supplementary material [180]. The used HMM-based recognition system allowed for an integration of the to-be-described language models during the decoding step.

In general, the usage of language information is considered very important in the context of OCR systems since, among other advantages, they allow to resolve errors that were

⁵<https://github.com/cisocrgroup/pocoweb>

2 Background and Related Work

inevitable for the actual optical recognition step, for example when dealing with highly degraded or even missing glyphs. The goal of language models is to imitate a human's capability to easily predict a word while reading a text. The described approach relied on n -gram language models which store statistical information about word sequences and whose task is to predict the n^{th} word based on the known $n - 1$ preceding words. As mentioned above, language models are usually built from large corpora, often by simply extracting word sequences and estimating their probability based on their occurrence. However, this straight forward approach yields a probability of 0 for word sequences that are not present in a corpus, making it impossible for the language model to recognize them later on. Since this problem is especially grave when dealing with small text corpora, like in this example, the authors of [324] incorporated a smoothing method [147] to deal with this problem.

For the experiments three initial systems were evaluated, using the language model and vocabulary created from the training set (System A), from the training and validation set (B), from the training, validation, and test set (C) to provide an upper limit for the recognition rate. Later on another system (B*) was introduced which shares the model and vocabulary with B but uses the optimized parameters from A to prevent overfitting. The result showed that A (word accuracy of 69.94% on the test set when using the most frequent 3,000 words from the training set as vocabulary) and B (68.59%) perform very similarly, while C (73.89%) performs best, as expected. Unfortunately, the recognition accuracy of the raw system without using any language modeling is not provided. Interestingly, B* performs significantly better (73.00% when using the entire vocabulary from the training set) than B (70.73%) showing that overfitting represents an actual issue and can be overcome by the proposed approach of using the vocabulary and language model from the training and validation set but the parameters optimized just on the training set.

Since the described procedure has shown that it can deal with a comparatively small amount of training data, it might be well suited to the current main area of application of OCR4all, that is iterative book-specific training using and creating medium amounts of GT either for a single book or a group of books with similar language properties.

2.5.3 Implementation in OCR Workflow Tools

ABBYY Finereader Since ABBYY's postcorrection functionality is said to be highly effective and since some actual details about the used method are available [2], we make an exception to our rule of ignoring ABBYY in this section due to its proprietary nature and the consequential lack of publicly available information. ABBYY provides language-specific dictionaries for over 50 languages, including six historical ones (English, French, German, Italian, Slavonic, and Spanish) [4]. The incorporation of custom dictionaries is supported as well.

After the ATR step the raw recognition result is combined with the dictionary information by calculating the *full confidence* which simply is the sum of the *recognition confidence*

and the *dictionary bonus* where the latter is the product of the *word length*, the *dictionary weight*, and the *word weight within the dictionary*. No further definitions of the terms are given but it is likely that a priori probabilities and *n*-grams based on word frequencies are used. The final comparison of two (or more) hypotheses then becomes relatively straight forward:

1. If both hypotheses do not occur in the dictionary, the raw recognition confidences are compared.
2. If both hypotheses do occur in the dictionary, the raw recognition confidences are compared
3. If one hypotheses does occur in the dictionary and the other one does not the calculated, full confidences are compared. If the confidences are very low, additional not further specified heuristics are applied.

OCROPUS 1 While the original OCROPUS 1 paper [56] introduced language modelling as a major component of the workflow the actual version does not incorporate an explicit language modelling step at all. In earlier versions OCROPUS 1 relied on the open-source *OpenFST* library [11] which utilizes weighted finite state transducers to represent statistical language models. Advantages of this approach are that it allows to separate the recognition from the language modelling step and its modularity allowing for example to extend a model providing a general approximation of English grammar by combining it with domain-specific dictionaries [56].

Tesseract 3/4 Before the switch from a glyph-based to a line-based recognition approach, i.e. before version 4.0, Tesseract relied on a linguistic module that is called into action every time the preceding word segmentation module looks at a new possible word segmentation. For the segmentation at hand the best available word candidate is determined for five categories, among others based on an underlying dictionary which can be adapted and provided during training: top frequent word, top dictionary word, top lower case word, and top classifier choice word [274]. The final output is then calculated by multiplying the probability for each of the five word candidates determined by the recognition step with a constant factor and choosing the best fitting one.

Since version 4.0 Tesseract does not support an explicit postcorrection step. However, several efforts to restore some of the lost functionality are currently underway⁶.

2.6 Tools and Projects Providing an OCR Workflow

Before discussing the various OCR workflow tools in detail, we first give an overview of their respective capabilities in Table 2.4. For our survey, we refer to the main steps

⁶No explicit source available. The assertion is based on information collected from the Tesseract GitHub repository [295] and from personal communications with developers related to the project.

2 Background and Related Work

defined during the introduction and for the most part only state whether a functionality exists or not. Regarding the ATR step, we mostly incorporated the ratings from Table 2.2 since, as shown above, there are several detailed comparative evaluations available which the other steps are lacking. As Calamari represents our main ATR engine, we adopted its ratings. There is a single exception: since OCR4all offers a line-based synoptic correction view including some user conveniences like a customizable virtual keyboard but currently lacking a dictionary or confidence based error detector, we adjusted the rating for *manual correction* accordingly.

In the following, we discuss the four tools from Table 2.4 but also briefly introduce other workflow tools as well as projects that deal with the OCR workflow.

Table 2.4: Comparison of existing tools providing an OCR workflow with OCR4all. In the *Historical ATR* row more ticks (✓, to a maximum of three) denote a higher ability/applicability of an ATR engine in the respective sub task. A cross (✗) shows that an ATR engine does not support a sub task at all. In all other rows we simply indicate whether a tool supports (✓) a sub task or not (✗).

Step	Sub Task	ABBYY	OCROPUS 3	Tesseract 4	OCR4all
Preprocessing	Deskewing	✓	✓	✓	✓
	Binarization	✓	✓	✓	✓
Segmentation	Image/Text	✓	✗	✓	✓
	Semantic Distinction	✗	✗	✓	✓
	Line Segmentation	✓	✓	✓	✓
	Reading Order	✓	✓	✓	✓
	Manual Correction	✓	✗	✗	✓
Historical ATR	Recognition	✓✓	✓✓	✓✓	✓✓✓
	Training	✓	✓✓	✓✓	✓✓✓
	Manual Correction ⁷	✓✓✓	✗	✗	✓✓
Postprocessing	Dictionaries	✓	✗	✗	✗
	Language Modelling	✓	✗	✗	✗
Open-Source	-	✗	✓	✓	✓

2.6.1 ABBYY

At least on contemporary material the proprietary ABBYY Finereader OCR engine⁸ clearly defines the state-of-the-art for preprocessing, layout analysis, ATR, and postcorrection. A wide variety of documents with considerably differing layouts can be processed by the fully automated segmentation functionality whose results can be manually corrected, if necessary.

⁷Of course it is always possible to manually correct results in some way if they are available in standard textual output formats (which all tools provide) but to get a tick here at least some kind of correction GUI within the tool is required.

⁸<https://www.abbyy.com/Finereader>

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

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

2.6.2 Tesseract

Just like ABBYY the open-source OCR engine Tesseract [295] provides a full OCR workflow including built-in routines for preprocessing like deskewing and binarization as well as for layout analysis⁹.

Tesseract's ATR training and recognition capability recently (version 4.0+) have improved considerably due to the addition of a new ATR engine based on LSTM neural networks which clearly outperformed the character-based approach during project internal experiments. The old glyph-based recognition method is still supported and mixed models trained for both recognition approaches and a wide variety of languages and scripts are openly available at the project's GitHub repository. Similar to ABBYY and contrary to OCRopus 1/2/3 and Calamari, Tesseract supported the use of dictionaries and language modelling until version 4, but this functionality has not been adopted to the line-based recognition approach, yet.

Tesseract has comprehensive strengths in the fully automatic out of the box processing of modern texts, its full support of right-to-left writing and vertical scripts, as well as the unmatched (at least by non-commercial tools) repository of freely available mixed models for a wide variety of (modern) languages and scripts. However, there are some drawbacks when it comes to historical material, e.g. the usage of Otsu binarization (cf. Section 2.2.1.1; although only for segmentation, not necessarily for ATR) and especially the lack of GPU support for training and recognition which considerably hinders the frequent use of book-specific training in an iterative training approach (cf. Section 5.6.2).

2.6.3 OCRopus

The open-source [206] toolbox OCRopus 1 [47, 56] comprises several Python 2-based tools for document analysis and recognition. This includes highly performant algorithms for deskewing and binarization as well as a segmentation module which extracts text lines

⁹For a comprehensive overview over the Tesseract OCR system cf. a relatively recent tutorial [297] by Tesseract's main developer Ray Smith.

2 Background and Related Work

from a page in reading order. While the segmentation can quite comfortably deal with modern standard layouts, that means text-only pages with clearly separable columns, it tends to struggle with typical historical layouts with marginalia, swash capitals, etc. When a page has already been split up into regions however, the line segmentation usually identifies the single lines very reliably and accurately at least when working with Latin script. This is not a trivial task since in historical printings the letters of adjacent lines can severely overlap vertically or even touch each other.

OCRopus 1 was the first OCR engine to implement the pioneering line-based approach for character recognition introduced by Breuel et al. [46] using bidirectional LSTM networks which allowed for considerably superior recognition capabilities compared to glyph-based approaches. Furthermore, this method significantly simplified the process of training new models since the user just has to provide image/text pairs on line level, which can be created by using a html-based transcription interface in a browser.

While the general line-based recognition still defines the state-of-the-art, the shallow network structure consisting of just a single hidden layer has to be considered outdated by now. The superiority of deeper architectures relying on a combination of CNN and LSTM layers has been shown well enough on different materials. Nevertheless, OCRopus 1 still proves to be a cornerstone for OCR workflows dealing with historical printings mainly for two reasons: First, the preprocessing usually achieves excellent results due to its robust deskewing approach (cf. Section 5.4.3) as well as its adaptive thresholding technique used for binarization (cf. [6] and Section 5.4.2) which can comfortably deal with pages even if they are in questionable condition. Second, due to the robust line segmentation (cf. Section 5.5.2).

After the comparatively disregarded OCRopus 2 [204], the third edition OCRopus 3 [205] was released in May 2018. It introduced a *PyTorch* [219]-backend which enabled the utilization of deep network structures and GPU support, resulting in better recognition rates and faster training and prediction. In the comparative study mentioned earlier [319], OCRopus 3 achieved recognition results similar to Tesseract 4 while being significantly faster but was nevertheless considerably outperformed by Calamari. Concerning the other steps in the OCR workflow like binarization, deskewing, and segmentation, OCRopus 3 almost exclusively relies on deep learning techniques. To the best of our knowledge, there no comparisons between the traditional methods of OCRopus 1 and the new approach by OCRopus 3 available, yet.

2.6.4 Kraken

The open-source [148] OCR software Kraken (see [143] for the initial paper) is originally based on an OCRopus 1 fork and has been significantly cleaned up as well as extended since. For example, Kraken seems to focus on the processing of Arabic text, resulting in an optimized line segmentation procedure which can deal with the specifics of Arabic script and a right-to-left text recognition support. The underlying OCRopus 1 architecture was

extended by a PyTorch backend enabling the training of deep networks consisting of a combination of CNN and LSTM layers.

In a very recent (July 2019) second publication [142] the present state of the software is briefly introduced. Apart from extensive recognition features like the support of right-to-left, bidirectional, and vertical writing, combined script detection and multiscript recognition are addressed. Moreover, a trainable deep learning line extractor is currently being implemented to allow dealing with the highly variant challenges of different scripts when it comes to line segmentation. Finally, results achieved on several publicly available data sets including historical ones are presented, mostly achieving above 98% or even 99% character accuracy. Unfortunately, neither details about the training and evaluation procedure nor a comparison with other open-source ATR engines are provided.

Despite the application to historical data the focus of the engine seems to lie on more recent printings. This is indicated by the fact that the results of the line segmentation are output as JSON files simply containing the line bounding boxes as straight rectangles. As mentioned above, this can be quite problematic when working on earlier printings, especially when considering that Kraken also got rid of OCRopus 1's deskewing functionality. Because of these shortcomings or rather these intended simplifications, Kraken does not seem ideally suited for an end-to-end application to historical printings.

2.6.5 Transkribus

A very comprehensive platform specialized on Handwritten Text Recognition (HTR) was developed within the Transkribus project¹⁰ [138] which provides a web service to store and share documents or perform layout analysis, recognition, and training tasks on the server. The main user interface is available as an open-source Java desktop application which allows the user to perform manual segmentation tasks or produce GT by transcribing lines. Unfortunately, large parts of the software are not open-source, preventing the users from adapting or extending the code and from running the advanced recognition tools on their own hardware. On the 1st of July 2019 a fee-based cooperative was founded that “serves as the basis for sustaining and further developing the Transkribus platform and related services and tools”.¹¹

The technical partner for the development of the layout analysis and training and recognition software is the CITlab¹² team at the University of Rostock whose approach performed best on the sub task of the detection of baselines at a competition layout analysis for challenging medieval handwritten documents at ICDAR2017 [269]. Several related publications are available (see for example [117, 118] for layout analysis and [184] for HTR) but to the best of our knowledge, the exact state of the software actually incorporated in Transkribus is not publicly known. Therefore, the best source for results seems to be a recently (May 2019) published talk [188] which briefly sums up some

¹⁰<https://transkribus.eu>

¹¹<https://read.transkribus.eu/coop>

¹²<https://www.mathematik.uni-rostock.de/forschung/projekte/CITlab>

2 Background and Related Work

evaluations: After training on close to 36,000 words corresponding to 182 pages a CER of 3.1% and a WER of 13.1% was achieved on a data set from the 18th century written by a single writer in German. For Latin and French medieval material from many different writers the system scored a CER of 6.4% and a WER of 22.1% after being thoroughly trained on over half a million words corresponding to close to 1,200 pages of GT. The application to printed text, more precisely to newspapers from the 18th century, led to a CER of 0.81% and a WER of 3.02% was achieved after training on 180,000 words corresponding to 345 pages.

2.6.6 DivaServices

With DIVAServices [321], Würsch et al. presented a fully open-source framework that allows to share and access document image analysis methods as a RESTful web service [241]. The idea is to allow the research community to simply provide access to newly developed methods via an unified API, independently from the used programming language, and therefore freeing the interested users from the burden of setting up and run a local instance after downloading the source code. DIVAServices supports various tools of different complexity, starting from smaller modules like binarization or line segmentation to more comprehensive tools like DIVAnnotation [256] which again can call other services themselves.

A recent publication [322] gives the latest updates concerning the execution environment (now using Docker just like OCR4all), the asynchronous execution of services, the output definition, and a planned workflow system that should allow the users to create their own workflows by specifying which modules, tools, and processes should be called in which order and with which parameters. Furthermore, there is a focus on building an ecosystem of tools and services providing further functionality to improve the usability of the system without being part of the core framework. This includes tools and services supporting experimentation, data and method management, programming libraries, and optimization.

While DIVAServices is a promising approach it cannot be considered a real workflow tool and is not meant to be one, yet. However, the available online collection of the document image analysis tool DIVAServices Spotlight¹³ represents a very helpful option to perform exemplary tests of existing methods on one's projects without the need for complicated setup operations. Unfortunately, to the best of our knowledge, not all showcased methods are available as open-source.

2.6.7 OCR-D

The OCR-D project¹⁴ [192] is funded by the *Deutsche Forschungsgemeinschaft - German Research Foundation* (DFG), initially for a period from 2015 to 2020. Its main goal

¹³<http://wuersch.pillo-srv.ch>

¹⁴<http://ocr-d.de/eng>

is to provide an OCR workflow for historical printings starting from the 16th century. The workflow defines a number of modules which are executed sequentially and whose development by different German research facilities is also funded by the DFG since 2018. Since the focus lies on mass digitization the aim is to keep the amount of manual user interaction to a minimum, ideally reducing it to zero resulting in a fully automatic workflow. Therefore, book-specific training or any kind of manual postcorrection, be it on layout or textual level, are currently neither envisaged nor desired. However, just as with Kraken, the high degree of modularity makes OCR-D an interesting project whose further developments should be closely followed. This is especially true since OCR-D also relies on PAGE and therefore has publicly released several wrappers for tools like Tesseract 4 to fully integrate them into their workflow. Since the project and therefore the developments of the submodules are still ongoing no evaluations of the overall workflow have been published, yet (December 2019).

Additionally to the efforts described above, OCR-D also aims to provide a GT reference corpus for German texts printed between 1500 and 1900. A description of the necessary formats and guidelines is given by Boenig et al. [38].

2.6.8 Nashi

The open-source [191] transcription environment Nashi [62] was created as a platform for the digitisation of the Arabic and Latin Corpus (ALC)¹⁵ at the University of Würzburg. Its main focus was to provide a group consisting of researchers and students with the opportunity to collaboratively segment, transcribe, and comment on scans of historical and modern printed editions in Latin, Arabic, and Greek language. Since the main goal is the creation of accurate citable digital editions, the web user interface for postcorrection provides the users with means to check and, if necessary, correct the ATR output for every single text line while also allowing to alter the coordinates of the line polygons. The transcription workflow is based on PAGE and can be considerably supported by ATR processes running in the background. Apart from Nashi the current ALC setup at the University of Würzburg relies on LAREX (cf. Section 5.5.1.1) for segmentation, Kraken for line segmentation, and has recently switched from Kraken to Calamari for ATR tasks.

Temporarily, Nashi served as the manual postcorrection module within the OCR4all workflow but meanwhile was replaced by an in-house development based on LAREX which we will discuss later on. However, some code snippets, for example for the PAGE-based line segmentation, have initially been taken over to OCR4all and have been considerably extended since.

¹⁵<http://arabic-latin-corpus.philosophie.uni-wuerzburg.de>

2.7 Data Standards and Formats

The storage of obtained results, both intermediate and final ones as well as physical and logical ones, represents an important and challenging task in the area of document analysis and specifically for OCR workflows [33]. In this section we introduce three widely used open formats and standards and briefly discuss their advantages and disadvantages with respect to our intended application scenario. We will focus on solutions specifically geared towards OCR workflows and ignore formats usable for deferred tasks, like for example TEI [292].

2.7.1 PAGE

The *Page Analysis and Ground-Truth Elements* (PAGE) format framework [216, 223] was developed at the PRImA Research Lab [224]. It is an open XML format¹⁶ that was and still is applied in numerous projects and institutions dealing with layout and text mark-up in the area of OCR workflows [192], including OCR4all.

PAGE requires one XML file per page, is very expressive, and can store a wide variety of information, most importantly concerning layout structure, ATR, references to and additional information about alternative images (for example deskewed binary images), and metadata, both bibliographical (author, title, ...) but also technical (processing steps, used software, ...). In this work we mainly focus on the layout and ATR functionality which consequently is described in greater detail in the following:

- A page can comprise an arbitrary number of regions whose reading order can be specified.
- Among others a region can store its enclosing polygon and its type.
- There are main types like image, text, or music. Text regions can be further classified into sub types like running text, heading, page number, marginalia, etc.
- A region can contain an arbitrary number of text lines whose reading order within a region can be specified.
- Each line stores its enclosing polygon as well as an arbitrary number of text elements which may contain GT, various ATR outputs, normalized texts, etc.
- Lines consist of words and words are made up of symbols.
- For each word and symbol the same attributes as for lines can be stored. In addition confidence information about the ATR output can be provided.

Due to its expressiveness and the ability to store and provide high quality GT for every step of the workflow, PAGE is regularly deployed in a variety of competitions dealing with layout analysis in general, including region and line segmentation, but also with textual recognition tasks.

¹⁶In the following we will rely on the term *PageXML* when referring to the actual files.

2.7.2 ALTO

While PAGE is widely used in the academic realm, institutions like libraries and archives mostly rely on *ALTO* [12, 14] which has to be considered the standard for data representation in document image analysis and is also recommended by the DFG [81] as the de-facto standard for digitization projects in Germany [82].

Developed during the the Metadata Engine (METAe) project [290], ALTO is usually used in combination with the Metadata Encoding and Transmission Standard (METS) to describe the digital object as a whole and also provide references within the ALTO file, for example to determine the reading order.

In general, again with focus on the layout and ATR functionality, an ALTO file is structured as follows (for more information we refer to the documentation [13]):

- An ALTO file consists of a list of top-level ALTO elements, like styles and layout, as well as their related attributes.
- Properties of layout elements are defined by style elements like text style (font properties) and paragraph style (formatting properties of text blocks).
- Pages are layout elements and divided in different areas, including the print space and various margins which may contain text or other elements not belonging to the main text body.
- Each area on the page may contain an arbitrary number of further elements, most importantly text and images blocks which are described by an ID and their respective bounding boxes and polygons.
- ALTO does not encode the reading order of elements on the page explicitly like PAGE does but instead relies on an implicit representation by the order within the XML document.
- Text blocks can refer to style elements and comprise text lines who can store their content and their enclosing polygon.
- Lines contain words and explicitly encoded whitespaces. Words store their position, content, confidence, and their contained symbols.
- Symbols can store the same information as words and in addition can provide alternative variants for the symbol and respective confidence values.

Overall, ALTO's main focus lies on storing the final results of digitization and document analysis pipelines, instead of describing and evaluating individual steps of the workflow in-detail.

2.7.3 hOCR

Another format geared towards representing both intermediate and final OCR result is the *hOCR* microformat. hOCR was initially created to be used within OCRopus

2 Background and Related Work

which specifically aimed to serve as an OCR system for all major languages and scripts. Therefore, OCRopus did not build from an a OCR-specific format but started with a format that already provided sufficient markup option. As suggested by the name, the format of choice was HTML which was then extended by several tags in order to represent some OCR-specific information. The resulting format uses a two-level typesetting model where the first one focuses on logical markup, while the second one represents the page layout using floats and boxes.

Since Tesseract version 3.0 introduced an option to output hOCR it can be considered a widely used format in the open-source OCR community, mainly due to its simplicity and flexibility. However, for our purposes PAGE still represents the go-to format, due to its expressiveness.

2.7.4 Converters

Naturally, it is possible to convert the formats introduced above, but also related ones like TEI [292] or ABBYY XML [5] into one another. However, due to the sometimes considerably different approaches a lossless conversion is not always feasible. Still, a wide variety of converters is freely available online (see [199] for a very comprehensive list) but many appear to be dormant and only support outdated versions. Notable exceptions are *OCR Fileformat* [202] which mainly focuses on the conversion from various formats into hOCR, the *PRImA PAGE Converter* [215] which supports a variety of transformations into PAGE, and the quite recently developed and active *PAGE 2 TEI* [214].

3 Data and Resources

In this chapter the data and resources used throughout the upcoming experiments are introduced. Table 3.1 gives an overview over the data used in course of this thesis.

Table 3.1: Summary of the data used during this thesis. Apart from the names of the (sub) corpora and the respective number of comprised works and lines, the use cases of the data are given.

(Sub) Corpus			(Partly) Used For
Complete Books	#Works	#Lines	
Early Printed Books			Evaluation: OCR4all workflow
• Narrenschiff	5	12k	
• Camerarius	17	17k	
• Miscellaneous	3	2.6k	
19 th Fraktur Novels	10	2.6k	Evaluation: OCR4all workflow
Sander’s Dictionary	1	1.4k	Training/Evaluation: case study typography recognition
Line-based GT only	# Works	# Lines	
GT4HistOCR			Training/Evaluation: voting, pretraining, AL
• ENHG	9	25k	Training: mixed models for 19 th century Fraktur
• Kallimachos	9	21k	
• EML	12	10k	
• RIDGES	20	13k	
• DTA19	39	244k	
Further 19 th Century Fraktur			Training/Evaluation: mixed models for 19 th century Fraktur
• Archiscribe	112	4k	
• CIS OCR Testset	2	0.5k	
• Newspaper Daheim	4	0.6k	
Modern Antiqua: UW3	?	90k	Training: mixed models for 19 th century Fraktur
19 th Fraktur Novels	13	3.5k	Evaluation: mixed models for 19 th century Fraktur

The data can be divided into two groups: First, there are full books which we mostly utilize to evaluate the complete OCR workflow comprised in OCR4all. Apart from various works from different centuries which were OCRed using standard ATR this category also contains a historical lexicon that we thoroughly indexed by making use of its comprised semantic information encoded using typographical attributes. Second, we

3 Data and Resources

also require line-based GT, which is exclusively used to evaluate the ATR steps, including the proposed methodical extension to the training and recognition tasks (cf. Chapters 4 and 7).

3.1 Complete Books for the Evaluation of the Entire Workflow

Our corpus of complete books can mainly be divided into (very) early printings from the 15th and 16th as well as German Fraktur novels from the 19th century. Additionally, we worked with a German dictionary from the 19th century.

3.1.1 Early Printed Books

Due to the focus of OCR4all on early printed books a large portion of our evaluation corpus consists of books printed before 1600, which are listed in Table 3.2 and can be further subdivided into three groups.

3.1.1.1 The Ship of Fools

The first group consists of editions of the *Narrenschiff* (Ship of Fools, see Figure 3.1 for some example pages), the second most popular book after the bible in the early modern period, and was digitized as part of an effort to support the *Narragonien digital* project [60] at the University of Würzburg. Despite their similar content these books are very different from an OCR point of view since their layout varies considerably and they were printed in different print shops using different typefaces and languages (Latin, German, French, and Dutch).



Figure 3.1: Example pages from different *Narrenschiff* editions.

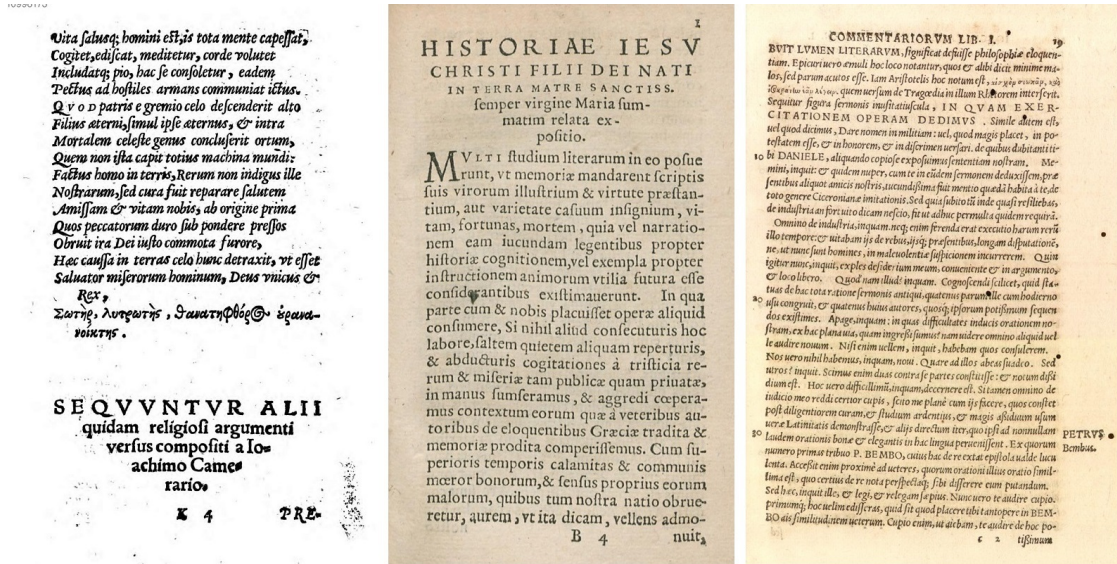
3.1 Complete Books for the Evaluation of the Entire Workflow

Table 3.2: Books of the early modern age used for our experiments including their *Full Title* and the *Languages* used within them. The *Identifier* encodes the group (Camerarius, Narrenschiff, Practical course) and the year of publication.

Identifier	Full Title	Languages
N1494	Das Narrenschiff	German
N1498	La nef des folz du monde	French
N1499	La grant nef des folz du monde	French
N1506	Nauis stultifera	Latin
N1549	Der Narren Spiegel	German
C1532a	Astrologica	Latin, Greek
C1532b	Ioachimi Camerarii Norica sive de ostentis libri duo	Latin, Greek
C1533	De theriacis et mithrateis commentariolus	Latin, Greek
C1535	Erratum	Latin, Greek
C1541	Elementa rhetoricae	Latin, Greek
C1552	Historia synodi nicenae	Latin, Greek
C1554	Versus senarii de analogiis	Latin, Greek
C1557	Libellus alter, epistolas complectens Eobani et aliorum quorundam doctissimorum virorum	Latin, Greek
C1558	De eorum qui cometae appellantur	Latin, Greek
C1561	Tertius libellus epistolarum H. Eobani Hessi	Latin, Greek
C1563	Dialogus de vita decente aetatem puerilem	Latin, Greek
C1566a	De Philippi Melanchthonis ortu, totius vitae curriculo et morte	Latin, Greek
C1566b	Historiae Iesu Christi Filii Dei Nati In Terra Matre Sanctiss. sempervirgine Maria summatim relata expositio	Latin, Greek
C1568	Libellus novus epistolas et alia quaedam monumenta doctorum	Latin, Greek
C1583	Epistolarum familiarum libri VI	Latin, Greek
C1594	Decuriae XXI symmikton problematon seu variarum et diversarum quaestionum de natura, moribus, sermone	Latin, Greek
C1598	De rebus turcicis commentarii	Latin, Greek
P1474	Das abenteürlich buch beweyset vns von einer frawen genandt Melusina	German
P1484	Histori von herren Tristrant	German
P1509	Fortunatus Eyne hystorye	German

3.1.1.2 Camerarius

In the second group we deal with printings related to the influential early modern universal scholar Joachim Camerarius the Elder whose numerous works have been identified and collected during the *Opera Camerarii* project [27] at the University of Würzburg (Figure 3.2 shows some example images).



Cum his Carolus [*Martellus scilicet. In Italico. Magnus est,*
 Cum his Carolus [Martellus scilicet. In Italico. Magnus est,
miffi atq; enucleati modi, græci Ἰανὸν uocāt, In quo
 missi atq; enucleati modi, græci @@@@ uocāt, in quo
γωγεύς uno uerbo græcis est, apte sinistra manu præ
 @@ @@@@ uno uerbo græcis est, apte sinistra manu præ

Figure 3.2: Top: Example images from different Camerarius works. Bottom: Lines showing the frequent change between Antiqua upright and italics (first line) and embedded parts of Greek (lines 2 and 3) as well as the corresponding transcription (green).

These works, which are now intended to be OCRed, are mostly written in Latin but frequently contain embedded parts of Greek, mostly scientific technical terms regarding the treated topics like astrology, medicine, and many more. A special feature of these books is that they often utilize two main fonts, i.e. Antiqua upright and italics, and also contain Greek sections directly within the Latin text. We focus on the ATR of the Latin parts and just ensure to mark Greek text for later processing by encoding

3.1 Complete Books for the Evaluation of the Entire Workflow

each letter using the character “@” (also shown in Figure 3.2). While this might seem counter-intuitive at first, ATR engines have been shown to be able to learn abstract representations of various scripts [304] or even different but very similar fonts (cf. our case study on typography recognition in Section 7.2).

3.1.1.3 Miscellaneous

Finally, the third group consists of various early modern printings that have been processed by students during a practical course on OCR4all for humanities scholars at the University of Würzburg (some examples can be seen in Figure 3.3). Camerarius and Narrenschiff books that have been processed during the practical courses are listed among the first two groups.



Figure 3.3: Example pages from various works processed by students during a practical course.

3.1.2 Corpus of 19th Century Fraktur Novels

The second part of our evaluation corpus consists of 19th century German novels (with one exception from the late 18th century) which are currently collected and OCR'd by the Chair for Literary Computing and German Literary History of the University of Würzburg (see Table 3.3).

Most of the books were scanned in 300 dpi and were provided by the Bayerische Staatsbibliothek¹. The overall quality of the material varies considerably as shown in Figure 3.4. This task requires a completely different OCR approach for various reasons: The

¹<https://www.bsb-muenchen.de>

3 Data and Resources

resulting corpus is intended to be used for experiments with quantitative methods which are usually quite robust with respect to OCR/ATR errors. Furthermore, from an OCR point of view, the material is considerably less complex compared to the early printed books we discussed before, due to its rather trivial layout, more standardized typography, and its superior state of preservation. The corpus is very extensive, currently comprising around 1,800 novels, and the project’s goal is to OCR all novels of this period (probably 10,000 to 15,000). These aspects make it neither necessary nor feasible to invest an extensive amount of manual work, which is why a highly automated workflow is intended instead.

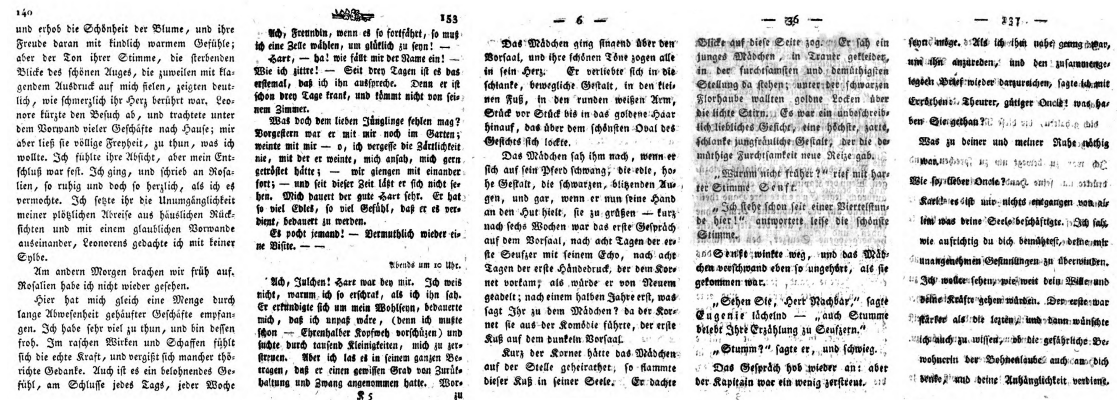


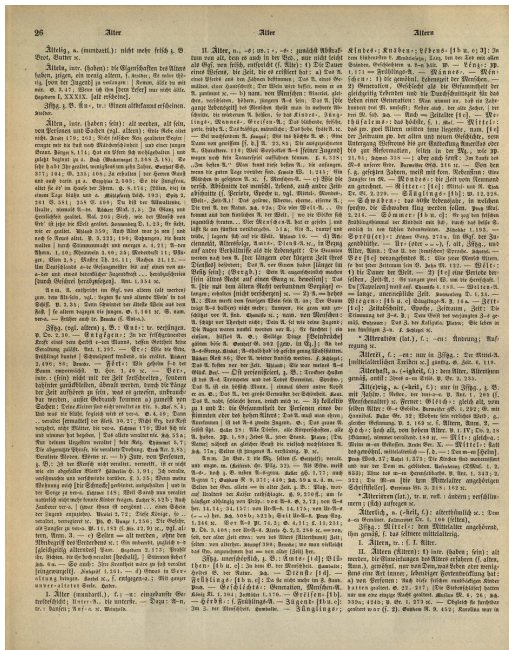
Figure 3.4: Example images from the German novel corpus. From left to right: F1870, F1781, F1818 (page in decent condition), F1818 (page in bad condition), F1803. Adopted from [229].

Table 3.3: German Fraktur novels used for our experiments given by their *Authors* and their *Titles*. The *Identifier* encodes the group (**Fraktur**) and the year of publication.

Identifier	Author	Title
F1781	Friedel, Johann	Eleonore
F1803	von La Roche, Sophie	Liebe-Hütten
F1810	Fouqué, Friedrich de la Motte	Der Held des Nordens
F1818	Lafontaine, August Heinrich Julius	Reinhold
F1826	Pichler, Caroline	Frauenwürde
F1848	Hahn-Hahn, Ida	Levin
F1851	Müller, Otto	Georg Volker
F1865	Hiltl, Georg	Gefahrvolle Wege
F1869	von Hillern, Wilhelmine	Der Arzt der Seele
F1870	Hiltl, Georg	Die Bank des Verderbens

3.1.3 Historical Lexicon: Sanders' Wörterbuch der Deutschen Sprache

The German dictionary of Daniel Sanders is one of the most important and most comprehensive historical German dictionaries and can still be considered an extraordinary linguistic “œuvre” of 19th century lexicography. Consisting of three part-volumes written between 1859 and 1865 it is a dictionary “completely rounded”. In comparison to our other evaluation data it stands out due to its particularly complex content consisting of lemmas as well as their definitions and references which are all encoded by the use of different typography classes (see Figure 3.5). The high quality scans at hand were provided by the University Library of Würzburg, whereas the text was gathered in cooperation with the Berlin-Brandenburg Academy of Sciences and Humanities².



Bäumeln, refl. : schwzr., sich emporrichten, namentl. vor Stolz = sich brüsten. *Gotthelf* 5, 280; 282 zc.; f. Bäumen I.
3ßßg. z. B.: *Äuf*: weidm.: sich auf die Hinterläufe setzen, vom Hafen.

Bäumeln, refl.: schwzr., sich emporrichten, namentl. vor Stolz = sich brüsten. *Gotthelf* 5, 280; 282 zc.; f. Bäumen I.
Zßßtg. z. B.: *Äuf*: weidm.: sich auf die Hinterläufe setzen, vom Hafen.

Figure 3.5: Example page from Sanders' dictionary (left) and an article extract demonstrating the complex use of typography to encode different semantic entities (color encoded on the right).

²<http://www.bbaw.de>

3.2 Line-based Ground Truth for the Evaluation of Individual ATR Tasks³

Line-based GT is a valuable resource needed to train and evaluate modern ATR engines. Since line-based GT is cumbersome to create we made use of freely available resources as much as possible. While some data has already been present in the required format, i.e. a line image together with the corresponding transcription, most resources had to be transformed first. By doing this we constructed the very comprehensive GT4HistOCR (Ground Truth for Historical OCR) corpus which we describe in the following. Apart from the data added to the corpus we collected further Fraktur data from the 19th century which we are going to introduce afterwards.

3.2.1 GT4HistOCR Corpus

In this section we describe and provide training material of historical GT which has been collected and produced by us over the course of several years and give information on its provenance. We specifically focus on early printings for which diplomatically transcribed text accurately matched against printed text lines is very scarce. As a result, we offer our combined GT resources to the scientific community for further training, research, and experimentation. The GT4HistOCR corpus comprises 313,173 lines of GT covering the incunabula period (German printings from the 15th century), Early Modern Latin printings (15th to 17th century), and German Fraktur printings (15th to 19th century). The corpus has been made available under a CC-BY 4.0 license in Zenodo [119]. Apart from the compressed corpora the repository contains various mixed models⁴ and a Perl script, which can be adapted to harmonize transcriptions using different guidelines in order to have a common pool of training data for mixed models.

In the following we introduce the five subcorpora of our *GT4HistOCR* corpus (see Table 3.4). As the transcription guidelines differ for each subcorpus in the amount of typographical detail that has been transcribed, we chose not to construct corpora according to language or period by mixing material from these subcorpora. However, the directory structure of our corpora encodes the metadata down to the publishing year and book containing the individual line images and transcriptions, allowing any users to construct new corpora according to their needs after an appropriate harmonization of the transcription. The transcription of these corpora was done manually (partly by

³This section contains a previously published article [282]: U. Springmann, C. Reul, S. Dipper, and J. Baiter, “Ground Truth for training OCR engines on historical documents in German Fraktur and Early Modern Latin,” *JLCL: Special Issue on Automatic Text and Layout Recognition*, 2019. [Online]. Available: https://jlcl.org/content/2-allissues/1-heft1-2018/jlcl_2018-1_5.pdf. The article was published under the Creative Commons Attribution-ShareAlike 4.0 International (CC BY-SA 4.0) License (<https://creativecommons.org/licenses/by-sa/4.0>), minor changes have been made. Permission to re-use content including text, figures, tables, and their respective captions has been given by the first author of the publication Dr. Uwe Springmann.

⁴All our models have been trained using *OCROPUS 1*.

3.2 Line-based Ground Truth for the Evaluation of Individual ATR Tasks

students) and later checked and corrected by trained philologists within projects in which we participated. The text line images corresponding to the transcribed lines have been prepared and matched by us using OCRopus 1 segmentations routines or, in the case of 19th Century DTA Fraktur Corpus (DTA19), the resulting segmentation of ABBYY Finereader.

Table 3.4: Overview of the subcorpora of *GT4HistOCR*. For each subcorpus we indicate the number of books (*# Books*), the printing *Period*, the number of lines (*# Lines*) and the *Language*.

Subcorpus	# Books	Period	# Lines	Language
Reference Corpus ENHG	9	1476-1499	24,766	German
Kallimachos Corpus	9	1487-1509	20,929	German, Latin
EML	12	1471-1686	10,288	Latin
RIDGES Fraktur	20	1487-1870 ⁵	13,248	German
DTA19	39	1797-1898	243,942	German
Sum:	89		313,173	

3.2.1.1 Reference Corpus Early New High German

The Reference Corpus Early New High German (ENHG) [89] is being created in an ongoing project which is part of a larger initiative with the goal of creating a diachronic reference corpus of German, starting with the earliest existing documents from Old High German and Old Saxon (750–1050), and including documents from Middle High German (1050–1350) and Middle Low German and Low Rhenish (1200-1650) up to Early New High German (1350–1650). The Reference Corpus Early New High German contains texts published between 1350 and 1650. From 1450 on prints are included in the corpus besides manuscripts. The last part, 1550–1650, consists of prints only. The texts have been selected in a way as to represent a broad and balanced selection of available language data. The corpus contains texts from different time periods, language areas, and document genres (e.g. administrative texts, religious texts, chronicles). From the Reference Corpus ENHG we received diplomatically transcribed GT for the incunabula printings in Table 3.5. Specimen of line images which give an impression of the fonts are shown in Figure 3.6. Full bibliographic details for these documents can be retrieved from the *Gesamtkatalog der Wiegendrucke* [106] via the GW number.

In principle we would like to have as large of a corpus as possible and reuse all transcriptions from 1450 up to 1650. However, the process of generating accurately segmented printed lines from scanned book pages and matching them to their corresponding transcriptions is still laborious. Since GT for periods later than 1500 is provided in other subcorpora, we just used the incunabula printings of the reference corpus.

⁵50% of the books have been printed before 1650.

Table 3.5: The Early New High German incunabulum corpus. Given are the printing Year, the GW number, the ((Short) Title), and the number of GT lines (# Lines) for training and evaluation.

Year	GW	(Short) Title	# Lines
1476	M51549	Historij	3,160
1478	04307	Biblia	2,745
1485	M09766	Gart der Gesundheit	2,520
1486	M45593	Eunuchus	3,403
1486	5077	Jherusalem	2,232
1490	10289	Pfarrer vom Kalenberg	2,503
1490	5793	Leben und Sitten	3,099
1497	5593	Cirurgia	3,476
1499	6688	Cronica Coellen	1,628
Sum:			24,766

**Op den burgeroyrten maendach
vnde nathan vnde zachariam vnde
die brust vnd den menschen in züfelligem
¶ Das ist ein entdeckung + oder lautere erkklärung in der
iagten sie vnd erflügen yr eyn teyl. welches
Wānyndert lebt vff erden kein man
stenget. So nymmet sy auch nicht ab. Den
molcken der geysß/ vñ werde gespiff
eme dat so gesacht wurde/ dat men eme gne**

Figure 3.6: Example lines from the Early New High German incunabulum corpus in chronological order (see Table 3.5). Adopted from [282].

3.2.1.2 Kallimachos Corpus

The Kallimachos corpus (see Table 3.6) comprises data from the identically named BMBF funded project [139, 252] including the 1488 printing of *Der Heiligen Leben*, which was gathered during a case study of highly automated layout analysis [230], and eight books from the *Narragonien digital* subproject [60] dealing with the *Narrenschiff* (*ship of fools*) by Sebastian Brant. There are four Latin printings (*Stultifera nauis*) translated by Locher and Badius, two Early New High German printings, one Early Low German work (*Der narrenschip*), and one Latin/English document (Barclay), which we just provide the Latin

3.2 Line-based Ground Truth for the Evaluation of Individual ATR Tasks

part of. Whereas the German documents use a broken script, some Latin works are printed with Antiqua types similar to our modern types (see Figure 3.7).

Table 3.6: The Kallimachos corpus.

Year	GW	(Short) Title	# Lines
1488	M11407	Der Heiligen Leben (Winterteil)	4,178
1495	5049	Das neu narrenschiff	2,114
1497	5051	Das nuw schiff von narragonia	1,197
1497	5056	Stultifera nauis	1,424
1497	5061	Stultifera Nauis	1,092
1499	5064	Stultifera nauis	721
1500	5066	Der narrenscip	2,500
1505		Nauis stultifera (Badius)	4,713
1509		The Shyp of Folys (Barclay)	2,990
Sum:			20,929

Diener. Da was der Keyser gar fro dz sein syn
 Et pudor: atq; fides/probitas/constantia
 hymel vmd fallen wider ab bis zu
 ir wißheit ist verschluckt/sie haben geirrt in
 nis nunq; satis laudata Nauis p Sebastia-
 té Florétinū atq; Franciscū Petrarcham
 Men diincker/men saincer/men specker op snaren
 Feria quinta post Martī post Saturnus per
 ui cui id dedicarē: tū quia saluberrima tua

Figure 3.7: Example lines from the Kallimachos corpus in chronological order (see Table 3.6). Both Antiqua fonts (Latin) and broken fonts (German) are present. Adopted from [282].

3.2.1.3 Early Modern Latin Corpus

In [279] Springmann et al. introduced a Latin data set of manual transcriptions from books produced during projects on ATR postcorrection funded by Common Language Resources and Technology Infrastructure (CLARIN) and DFG [277]. The Early Modern Latin (EML) corpus is essentially the same, but leaves out the 1497 *Stultifera Nauis* (belonging to the *Kallimachos* corpus) and adds the 1543 *Psalterium* of Folengo (see

Table 3.7). With exception of the *Speculum Naturale* of Beauvais the books are mostly printed in Antiqua types (see Figure 3.8).

Table 3.7: The Early Modern Latin corpus.

Year	(Short) Title	Author	# Lines
1471	Orthographia	Tortellius	417
1476	Speculum Naturale	Beauvais	2,012
1483	Decades	Biondo	915
1522	De Septem Secundadeis	Trithemius	201
1543	De Bello Alexandrino	Caesar	830
1543	Psalterium	Folengo	314
1553	Carmina	Pigna	297
1557	Methodus	Clenardus	350
1564	Thucydides	Valla	1,948
1591	Progymnasmata vol. I	Pontanus	710
1668	Leviathan	Hobbes	1,078
1686	Lexicon Atriale	Comenius	1,216
Sum:			10,288

Igitur apud antiquiores nostros & graecos eiusdem profus
gogicā dicit & vnificā. Vnificā quia dispersos ī
reuertit: & postea philippo cum eiusdem nominis filio fraude
tatum excepisti & quidem liberalissime! plura
his erant quinqueremes & quadriremes X. reliquæ infra
set in tabulis lapideis populo lapidibus ipsis duriori,
Et qui Pontificis maximi ad Arcana uocatus es,
quod tanquam circunstes propositam senten-
statim ab eo moto: sperans etiam fore
neratione taceam, illi maximè scriptis
te Actionis diuersas producit Apparitiones!
etiam ex Atrio nostro, defumta esse

Figure 3.8: Example lines from the Early Modern Latin corpus in chronological order (see Table 3.7). Adopted from [282].

3.2.1.4 RIDGES Corpus

The use of broken scripts dates back to the 12th century and was once customary all over Europe. It is therefore of considerable interest to be able to recognize this script in order to OCR the large amount of works printed in a variety of Fraktur. This data set collects Fraktur material from 20 documents of the *RIDGES* (Register in Diachronic German Science) corpus of herbals [208], which has been proofread for diplomatic accuracy and matched against the best available scanned line images (see Figure 3.9). ATR experiments on this corpus were reported in [280]. Note that the author of the 1543 printing was erroneously attributed to Hieronymous Bock and has been corrected to Leonhart Fuchs in Table 3.8.

Table 3.8: The RIDGES Fraktur corpus.

Year	(Short) Title	Author	# Lines
1487	Garten der Gesunthait	Cuba	747
1532	Artzney Buchlein der Kreutter	Tallat	504
1532	Contrafayt Kreüterbuch	Brunfels	366
1543	New Kreüterbuch	Fuchs	483
1557	Wie sich meniglich	Bodenstein	995
1588	Paradeißgärtlein	Rosbach	795
1603	Alchymistische Practic	Libavius	473
1609	Hortulus Sanitatis	Durante	696
1609	Kräutterbuch	Carrichter	677
1639	Pflantz-Gart	Rhagor	1,091
1652	Wund-Artzney	Fabricius	601
1673	Thesaurus Sanitatis	Nasser	733
1675	Curioser Botanicus	Anonymous	567
1687	Der Schweitzerische Botanicus	Roll	520
1722	Flora Saturnizans	Henckel	562
1735	Mysterium Sigillorvm	Hiebner	470
1764	Einleitung zu der Kräuterkenntniß	Oeder	916
1774	Unterricht	Eisen	562
1828	Die Eigenschaften aller Heilpflanzen	Anonymous	658
1870	Deutsche Pflanzennamen	Grassmann	868
		Sum:	13,248

3.2.1.5 DTA19 Corpus

The use of broken scripts in the 19th century and later was mostly restricted to Germany and some neighboring countries. What distinguishes the 19th century is the large amount of available scans and the variety of the printed material (newspapers, long-running

vnd trucken in dem zwayten grad
Cinamomum. Symetrolin/neuss sie fast/ist gut fur
igen feuchtigkeit/also das es kulet big vff den anfang
yll/oder Dyllkraut würdt zu Latein vnd auff Griechisch
rath der müter / treibt auß die an
Weins in das Erdreich sind gestelle/
er desto mehr ohl denn sonst.
langlechte schmable Wurzeln/
süchtig Wasser zun Solen herauf / also
gelegt / dem rechten Verstand der Worten nicht
cher vnder das Gebieth der Löblichen berühmten.
Grund mit dem Garne sitzen/darnach sich wol auß
fer ist kalt und trucken im
Mond verpfanget / man kan dise
ganz bläß-grünliche Farbe ; Sal com. kisset es unge
durch empfinden, inmassen der Jovialishe Geist, so
nach unten zu eingerollet ist, und währenden
Strünke selber, so weit sie zart sind,
Bilsenkraut, Saubohnen, Schlafkraut.
er über die deutschen Namen zu jeder Gattung hinzuffügt

Figure 3.9: Example lines from the RIDGES Fraktur corpus in chronological order (see Table 3.8). Adopted from [282].

journals such as *Die Grenzboten* [113] or *Daheim*, encyclopedias [328], dictionaries, novels, and reprints of classical works from previous centuries) to philologists and historians.

Because of this high interest, some prominent works have been converted into electronic form by manual transcription (keyboarding, double-entry transcription). Given the sheer amount of available material, faster and less costly alternatives are sought after and both commercial (ABBYY Finereader with a special Fraktur licence [3]) and open-source ATR engines (Tesseract and OCRopus 1) are capable of recognizing Fraktur printings. What motivated us to look at 19th century Fraktur separately were the questions whether we could beat the general recognition models of available ATR engines, which we address in Section 7.1.

It is tempting to use synthetic training materials, as a variety of Fraktur computer fonts is readily available on the internet. However, closer inspection shows that many are either lacking some essential characteristics of real Fraktur types (such as *f*, or *ch* and *tz* ligatures) or have obviously been constructed for calligraphic use and do not reflect the

3.2 Line-based Ground Truth for the Evaluation of Individual ATR Tasks

most frequently used historical types. For the best ATR results we again have to rely on transcriptions of real data.

In the following, we describe a collection of transcriptions from the *Deutsches Textarchiv (DTA)* [79, 107] for which line segmentations from ABBYY Finereader are available. The corresponding scans of these transcriptions are held by *Staatsbibliothek zu Berlin*⁶. One of the authors of [282] (Johannes Baiter) produced line images by cutting page scans into lines using the line coordinates contained in the ABBYY XML output. In this way a corpus of 63 books, some belonging to multi-volume works, could be assembled fully automatically. From these we selected just one volume of each multi-volume edition to provide a balanced multi-font corpus and did some quality checks on correct segmentations by hand.

The resulting *DTA19* corpus of 39 works is detailed in Table 3.9 and example line images are shown in Figure 3.10. To our knowledge a similar extensive collection of GT for German 19th century Fraktur does not exist.

da der Seelige eine sehr kleine, unles
und blies starke Dampfwolken von sich, daß wir
sich auch mit freudigem Muth zu vollbringen in kurz
mit einem Sprunge hinter dem Wagen, der Kutscher
ihr Schweigen immer mehr ausdrücken, als
zu leiden, denn die Nothheit und Gemeinheit, die Abform
Karten und Pläne zu zeichnen. Schon waren von
der mit diesem lästigen Leiden so fatal verwachsen war,
einem gewissen Stolz wies sie ihm das hochgethürmte
nach nicht, sondern ganz andere Dinge spielen

Figure 3.10: A selection of lines from the DTA19 corpus. From top to bottom: 1815, 1817, 1819, 1826, 1835, 1853, 1861, 1879, 1891, 1897. Adopted from [282].

3.2.2 Further 19th Century Fraktur Data

In the following we briefly describe some smaller corpora which are not part of the DTA19 corpus but also comprise Fraktur data from the 19th century.

3.2.2.1 Archiscribe Corpus

A prime obstacle for generating GT for ATR training purposes consists in the segmentation of textual elements on a printed page into text lines. To circumvent this problem one of the authors of [282] (Johannes Baiter) proposed a crowd sourcing approach by making

⁶<http://staatsbibliothek-berlin.de>

Table 3.9: The DTA19 Fraktur corpus.

Year	(Short) Title	Author	# Lines
1797	Herzensergießungen	Wackenroder	5,150
1802	Ofterdingen	Novalis	6,198
1804	Flegeljahre vol. 1	Paul	5,332
1815	Elixiere vol. 1	Hoffmann	8,008
1816	Buchhandel	Perthes	861
1817	Nachtstücke vol. 1	Hoffmann	6,578
1819	Revolution	Görres	6,178
1821	Waldhornist	Müller	2,343
1826	Taugenichts	Eichendorff	7,662
1827	Liebe	Clauren	6,724
1827	Reisebilder vol. 2	Heine	5,980
1827	Lieder	Heine	5,873
1828	Gedichte	Platen	5,103
1828	Literatur vol. 1	Menzel	8,124
1832	Gedichte	Lenau	4,446
1832	Paris vol. 1	Börne	5,329
1834	Feldzüge	Wienbarg	7,805
1835	Wally	Gutzkow	5,728
1852	Ruhe vol. 1	Alexis	9,314
1852	Gedichte	Storm	2,038
1853	Ästhetik	Rosenkranz	14,062
1854	Heinrich vol. 1	Keller	9,343
1854	Christus	Candidus	2,095
1861	Problematische Naturen vol. 2	Spielhagen	6,445
1863	Menschengeschlecht	Schleiden	1,788
1871	Bühnenleben	Bauer	12,008
1877	Novellen	Saar	6,354
1879	Auch Einer vol. 2	Vischer	10,492
1880	Hochbau	Raschdorff	661
1880	Heidi	Spyri	6,210
1882	Sinngedicht	Keller	11,209
1882	Gedichte	Meyer	6,262
1886	Katz	Eschstruth	6,601
1887	Künstlerische Tätigkeit	Fiedler	4,983
1888	Irrungen	Fontane	7,079
1891	Bittersüß	Frapan	7,008
1897	Gewerkschaftsbewegung	Poersch	1,476
1898	Fenitschka	Andreas-Salomé	4,753
1898	Erinnerungen vol. 2	Bismarck	10,339
Sum:			243,942

3.2 Line-based Ground Truth for the Evaluation of Individual ATR Tasks

use of several open APIs of the Internet Archive [132, 133] to directly retrieve line images from historical books that can be used as the image source for creating GT. The Internet Archive hosts a collection of over 15 million texts sourced mainly from Google Books and a growing number of volumes scanned by volunteers and cooperating institutions. For every scanned book an automated process creates OCR results with ABBYY FineReader which can then be downloaded by users. While the actual ATR output of this engine for text with Fraktur typefaces is of very low quality (presumably the default Antiqua model is used), the resulting line segmentation is usually fairly accurate.

To create GT from the Internet Archive corpus, a simple web application, Archiscribe [31], is provided. First-time users of the application have to read through a simplified version of the transcription guidelines of the DTA. Then they are offered the option to pick a certain year between 1800 and 1900 and set a number of lines they want to transcribe. In order to retrieve these lines from a suitable book, Archiscribe uses the publicly available search API of the Internet Archive to retrieve a list of 19th century German language texts and randomly picks a volume that has not yet been transcribed. To determine whether a given text is actually set in Fraktur a heuristic is used: The ATR text is downloaded and searched for the token *ift*, a common misinterpretation by ATR engines trained on Antiqua fonts of the actual word *ift* (is), which has a high frequency in any German text (of course, real books also contain quotations and other material in Antiqua). If this heuristic results in a false positive (there are some books printed in Antiqua employing a f), one can just start over. Once a suitable book is found, the desired number of line regions⁷ are picked at random from the OCR result.

To serve the images to the user Archiscribe uses the publicly available International Image Interoperability Framework (IIIF) [276] Image API endpoint of the Internet Archive⁸. Since the API allows the cropping of regions out of a given page image (hosted by the archive.org server), the application can directly use it for rendering the line images in the user's browser, without the necessity of processing the image on the Archiscribe server. Once a suitable volume has been picked and the lines to be transcribed have been determined, the user is presented with a minimal transcription interface consisting of the line to be transcribed, a text box to enter the transcription and an on-screen keyboard with a number of commonly occurring special characters not available on modern keyboards. To offer more context in difficult cases the user may opt to display the lines above and below the line to be transcribed. When all lines have been transcribed, they are submitted to the Archiscribe server, where they are stored alongside with their corresponding line images in a Git repository that is published to the corpus repository on GitHub on every change.

Currently, the application is restricted to 19th century German language books from the Internet Archive, but it is planned to add support for the transcription of books sourced from any repository that offers an IIIF API.

⁷user-defined, by default 50

⁸<https://iiif.archivelab.org/iiif/documentation>

3 Data and Resources

The *Archiscribe* corpus [30] of GT generated by crowdsourcing with the Archiscribe tool currently (August 2019) consists of 4,255 lines from 112 works published across 73 years, evenly distributed across the whole 19th century. All of the data is available under a CC-BY-NC-SA license.

3.2.2.2 JZE Corpus

As another source for Fraktur data from the 19th century we used the data made available by Jesper Zedlitz at GitHub [329]. It provides eight books comprising around 1,600 lines, an already trained OCRopus 1 Fraktur model, and a list of words which start with rarely occurring capital letters (ÄÖÜQY), which we use for the creation of synthetic GT later on.

3.2.2.3 Material from the CIS OCR Testset

The *Center for Information and Language Processing (CIS) OCR test set* [73] contains data of Latin, Greek, and historical German works. Apart from the high quality scans which have been binarized and despeckled using ScanTailor [20], it provides example outputs of several ATR engines as well as transcriptions on a page to page basis that had to be matched to obtain GT on line-level. For our purposes, we used the novel “Die Wahlverwandtschaften” and the journal “Die Grenzboten” that represent the German Fraktur part of the data set.

3.2.2.4 Newspaper Daheim

Daheim is a German journal published between 1864 and 1943. During a project of the Chair for Literary Computing and German Literary History of the University of Würzburg many issues have been scanned and OCRed using ABBYY Finereader by the University Library of Würzburg. The original printing quality is excellent as are the state of preservation and the scan quality (600 dpi). For our evaluation we randomly chose some sample pages from the volumes 1865, 1875, 1882, and 1892, segmented them into lines and corrected the ABBYY output.

3.2.3 Modern Antiqua Data: The University of Washington 3 Database

The youngest data we used for our experiments comes from the *UW3 database* [222] which contains scans and transcriptions of 1,600 pages from late 20th century English speaking sources, mainly scientific publications, all printed in Antiqua. Extracting the GT on line-level [309] led to a comprehensive data set consisting of over 90,000 lines which have been made publicly available [308]. Some clean-up was performed, for example text lines containing mathematical equations were omitted from the data. Due to the comparatively low variance of the typography and the overall high quality of the line

images corpus-specific experiments on the UW3 corpus are clearly the least challenging ones and consequently yield by far the best results.

3.3 Mixed Models

For some of our evaluations mixed models play an integral role. In the following we briefly describe the models we trained utilizing the GT introduced above and the two ATR engines OCRopus 1 and Calamari.

3.3.1 OCRopus 1

Our first model *LH* was trained on all twelve historical books from the EML corpus. The books are printed in Latin and with Antiqua types. The training was performed on 8,684 lines and was stopped after 109,000 iterations. We evaluated all resulting models on 2,432 previously unseen test lines in order to determine the best model which occurred after 98,000 training steps achieving a CER of 2.92% .

Additionally, we use the freely available OCRopus1 standard models for English (*ENG*) [48] trained on the UW3 data set and German Fraktur (*FRK*) [49] introduced in [46] and described in Section 2.4.3.2.

3.3.2 Calamari

After switching to Calamari as our first-choice ATR engine we produced several new default voting ensembles consisting of five individual models and made them publicly available [64]: The models “antiqua historical” (*AH*) and “fraktur historical” (*FH*) resulted from training on the works from the Early New High German (ENHG), Kallimachos, EML, and RIDGES printed in Antiqua or Fraktur (or more generally broken script), respectively. Since *AH* and *FH* were trained on the regularized data from the GT4HistOCR corpus they cannot output ligatures as single characters but will produce the resolved version instead. Since some projects aim for a highly diplomatic transcription we enabled the models to output ligatures and other previously regularized characters by “correcting” the GT of 100 lines per book according to the new transcription. Training these new lines, starting from the original *AH* and *FH* models, and using a high degree of data augmentation resulted in “antiqua historical ligs” and “fraktur historical ligs”.⁹ Finally, the “antiqua modern” model (*AM*) was trained on the UW3 data set.

During the processing of various works related to the Opera Camerarii project it became apparent that even the best fitting existing mixed model (in most cases *AH*) was not

⁹In the following we use the abbreviations *AH* and *FH* for both versions, with and without ligatures, since it never makes sense to use both versions at the same time as the choice of models depends on the user demands.

3 Data and Resources

ideally suited to the special challenges of the material at hand. Primarily, the frequent occurrence of Greek passages and words embedded in Latin sentences as well as the widespread usage of italic Antiqua fonts led to a rise of the error rate and also lessened the effect of using the mixed model as a starting point for book-specific training. To deal with this issue we bundled the GT produced during the OCR of Camerarius works and trained a model that is specialized on these type of printings and is able to comfortably deal with Antiqua upright and italics as well as capable of recognizing/marketing a multitude of Greek glyphs as the character “@”.

4 Methods

As discussed in previous chapters recent progress on ATR methods using recurrent neural networks with LSTM architecture [128] enabled effective training of recognition models for both modern (20th century and later) and historical (19th century and earlier) handwritten and printed documents [46, 95, 280, 281]. Individually trained models regularly reach CERs below 2% for even the earliest printed books. The need to train individual models in order to reach this level of recognition accuracy for early printings sets the field of historical ATR apart from readily available (commercial and open-source) *general* (also called *polyfont* or *omnifont*) models trained on thousands of modern fonts which yield CERs lower than 2% on 19th century printings and lower than 1% on modern documents. Training historical recognition models on a variety of typesets results in *mixed models* which may be seen as a first approximation to modern polyfont models, but their predictive power is considerably lower than that of individual models.

In view of the mass of available scans of historical printings we clearly need automatic methods for ATR which in turn require good historical polyfont models. As long as these models are not available and at present cannot be easily constructed (we lack the necessary historical fonts to be able to synthesize large amounts of training material automatically), our next best approach is to maximize the recognition rate of a small amount of manually prepared GT. Therefore, in the following we propose and evaluate several improvements regarding the training and recognition process that enable us to maximize the efficiency of the book-specific training and consequently minimize the required manual effort for the user.

First, we introduce *cross fold training* and *confidence voting* in Section 4.1 which enables the user to train several models at once and combine their outputs. Second, a so-called *pretraining* approach is proposed in Section 4.2 where the training process starts from an existing, not necessarily matching, mixed model instead of from scratch, i.e. a random parameter configuration. Third, we combine the voting and pretraining functionality and also add Active Learning in Section 4.3 which allows the user to purposefully select those lines for GT production and subsequent training, where the model is expected to learn the most from.

The proposed methods work with any given ATR engine that can be trained on real historical data, provides usable confidence information, and can be adapted to allow building from existing models. Still, for the actual experiments we used OCRopus 1 since, at the time, Calamari, OCRopus 2/3, and Tesseract 4 were not available, yet, Kraken still was very similar to OCRopus 1 and while it was possible to train individual models for early printed books using Tesseract 3 [145], the GT production and training process

was considerably more time consuming and led to less accurate results compared to using OCRopus 1.

4.1 Cross Fold Training and Confidence Voting¹

This subchapter introduces a method that significantly reduces the CERs for ATR results obtained from models trained on early printed books. Our goal is to improve the ATR accuracy with a given amount of GT by training different models and then apply a voting procedure to combine them. Most approaches use voting with outputs generated by different ATR engines like OCRopus, Tesseract, or ABBYY Finereader but due to the shortage of capable ATR engines, the absence of suitable mixed models for early printings, and consequently the need for book-specific training, the usage of several engines becomes impractical and therefore undesirable.

With the goal in mind to only use a single ATR engine, we still need to be able to generate variance between the trained models since the voters not only need to be strong individually but also have to be diverse enough to make the voting profitable. Hence, we propose a cross fold training approach on a given GT pool which leads to several models with different characteristics. To improve the voting result further we utilize the intrinsic confidence values produced by the engine. This enables the voting not only to take the top-1 output character into account but the top-n alternatives weighted by their confidences. In addition and for comparison, we use the Information Science Research Institute (ISRI) analytic tools [240] for alignment and majority voting.

The rest of the sub chapter is structured as follows: Section 4.1.1 introduces and discusses related work on voting in the context of ATR. The methods applied are described in detail in Section 4.1.2. In Section 4.1.3 the results achieved on seven early printed books are evaluated. These results are discussed in Section 4.1.4 before Section 4.1.5 concludes the subchapter.

4.1.1 Related Work

An overview regarding topics concerning the improvement of ATR accuracy through combination is given in [122]. Apart from different methods to combine classifiers string alignment approaches are discussed.

The voting method to improve ATR results obtained from a variety of commercial ATR engines is introduced in [238]. As early as 1996 Rice et al. [240] released a collection of command line scripts for the evaluation of ATR results called the ISRI analytic tools. These tools contain a voting procedure which first aligns several outputs using the Longest

¹This section is based on a previously published article [234]: C. Reul, U. Springmann, C. Wick, and F. Puppe, “Improving OCR Accuracy on Early Printed Books by utilizing Cross Fold Training and Voting,” in *2018 13th IAPR International Workshop on Document Analysis Systems (DAS)*. IEEE, 2018, pp. 423–428. [Online]. Available: <https://doi.org/10.1109/DAS.2018.30>

4.1 Cross Fold Training and Confidence Voting

Common Substring (LCS) algorithm [239] and then applies a majority vote including heuristics to break ties. The tools were used to evaluate the results of various commercial OCR engines in several competitions on modern prints (see [237], e.g.). This voting procedure was able to improve the CER of five engines on English business letters from between 9.90% and 1.17% to 0.85%.

A different approach to achieve variance among OCR outputs was proposed by Lopresti and Zhou [169], who simply scanned each page three times and ran a single OCR engine on them. Their consensus sequence voting procedure led to a reduction of error rates between 20% and 50% on modern prints resulting in a CER as low as 0.2%.

Boschetti et al. [42] achieved an average absolute gain of 2.59% compared to the best single engine (ABBYY, as low as 3% CER) by combining the outputs of three different engines on ancient Greek editions from the 19th and 20th century. They applied a progressive alignment which starts with the two most similar sequences and extends the alignment by adding sequences. Then, the character selection is performed by a Naive Bayes classifier.

In [172] Lund et al. trained maximum entropy models on synthetic data using voting and dictionary features. On a collection of typewritten documents from World War II their method achieved 24.6% relative improvement over the WER of the best of the five employed OCR engines.

Wemhoener et al. [315] proposed an approach for aligning and combining different OCR outputs which can be applied to entire books and even different editions of the same book. First, a pivot is chosen among the outputs. Then, all other outputs are aligned pairwise with the pivot by first finding unique matching words in the text pairs to align them using a LCS algorithm. By repeating this procedure recursively two texts can be matched in an efficient way. Finally, all pairs are aligned along the pivot and a majority vote determines the final result.

Liwicki et al. [168] tackled the task of Handwritten Text Recognition (HTR) acquired from a whiteboard by combining several individual classifiers of diverse nature. They used two base recognizers which incorporated HMMs and bidirectional LSTM networks and trained them on different feature sets. Moreover, two commercial recognition systems were added to the voting. The multiple classifier system reached a WER of 13.84% and therefore outperformed even the best individual system (18.74%) significantly.

Azawi et al. [7] used weighted finite-state transducers based on edit rules to align the output of two different ATR engines. Neural LSTM networks trained on the aligned outputs are used to return a best voting. Since the network has used plenty of training data similar to the test set, it is able to predict correct characters even in cases, where both engines failed. During tests on German Fraktur printings and the UW3 data set the LSTM approach led to CERs around 0.40%, while the ISRI voting tool and the method presented in [315] achieved between 1.26% and 2.31%. A principal drawback of this method is its reliance on fixed input-output relationships, i.e. each ATR token is mapped to a single “correct” token. But historical spelling patterns are much more variable than

4 Methods

modern ones and the same word is often spelled and printed in more than one form even in the same document. This method therefore not only corrects ATR errors but also normalizes historical spellings which may not be desired.

Our method shows considerable differences compared to the work presented above. Not only is it applicable to some of the earliest printed books, but it also works with only a single open-source ATR engine. Furthermore, it can be easily adapted to practically any given book using even a small amount of GT without the need for excessive data to train on (60 to 150 lines of GT corresponding to just a few pages will suffice for most cases).

4.1.2 Materials and Methods

The general idea of the proposed approach is to significantly improve the accuracy that can be achieved by using only a single ATR engine. There is a trade-off between adding more GT to the training pool (a costly manual process) and a considerable increase in the required computational effort. Here we take the second route with a given amount of GT, keeping the manual effort to a minimum. In the following, we briefly introduce the data and the workflow² before describing the confidence based voting and the needed adaptations in the OCRopus 1 engine in detail. The standard majority voting without confidence can be carried out by using any given ATR engine and the ISRI tools in their default configuration.

4.1.2.1 Data

The experiments were performed on seven early printed books (see Table 4.1 and cf. Section 3.2.1 for further details). Besides three editions of the *Ship of Fools* (1495, 1500 and 1505), 1476 from the ENHG Corpus and 1488 and 1572 from the Kallimachos Corpus, 1675 was specifically chosen from the RIDGES Corpus for its low quality and the resulting high error rate. To avoid unwanted side effects only lines from running text parts were used and headings, marginalia, page numbers etc. were excluded. Figure 4.1 shows some example lines.

²The corresponding code is available at <https://github.com/chreul/mptv>

Dem hoeghen marschalck sent quirtjn
 dienen. Da was der Keyser gar fro dz sein syn
 en vnd materien erlengert vnd sch
 Sonder beghinzel Volmæct in Dreughden.
 Stultoz ꝛc. Qm̄ inquā stultoz vt
 complexus sit: ut sub æquatore est
 Und angehängter besondern Tabell

Figure 4.1: Different example lines from the seven used books. From top to bottom: 1476, 1488, 1495, 1500, 1505, 1572, 1675. Adopted from [234].

Table 4.1: Books used for Evaluation including their *Year* of publication and *Language* as well as the available lines for *Training* and *Testing*.

ID/Year	Language	GT Train	GT Test
1476	German	1,000	2,000
1488	German	1,000	3,178
1495	German	1,000	1,114
1500	Dutch	1,000	1,500
1505	Latin	1,000	2,289
1572	Latin	1,000	541
1675	German	250	317

4.1.2.2 Workflow

The general workflow can be described as follows:

Input: Line-based GT consisting of line images and the corresponding transcription.

Output: Recognized text lines.

1. Divide the available GT in N distinct folds and set aside some held out data for evaluation.
2. Train N ATR models.
 - a) Declare one of the folds as test data and allocate the rest for training.
 - b) Run a training using $N-1$ folds as training data.
 - c) Choose the best model by testing on the remaining fold.
3. Apply the N trained models to previously unseen lines (the held out evaluation data) and determine the result by voting.

4.1.2.3 GT Allocation and Model Training

For example, if the number of folds N is 5 and there are 150 lines of GT available, each fold contains 30 lines. For the first training, fold 1 is used for testing and folds 2 through 5 for training. For the second training, fold 2 for testing and folds 1, 3, 4, 5 for training, and so on. So in this example each OCRopus 1 training is carried out on 120 lines, which represent 80% of the GT pool. The entire training process closely follows the approach described in [277] and is depicted in Figure 4.2.

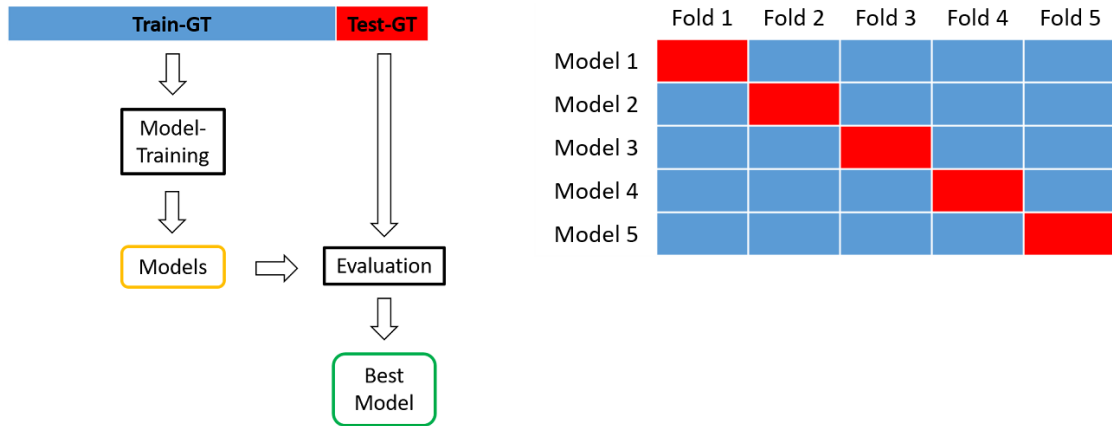


Figure 4.2: Schematic model training of a single fold (left) and the basic idea of the cross fold training (right). Please note that the allocation of lines to the training and test sets is visualized in a schematic way. In practise, an optimal distribution of lines, ideally stemming from different partitions of the book, is ensured to avoid unwanted side effects.

After the training process is finished, the best of the resulting models for each fold is determined by recognizing the test lines with each model and select the one with the lowest CER. These five best models are then used to recognize the unseen lines of the held-out data resulting in five text outputs for each line, which then serve as input for the voting tool.

4.1.2.4 Alignment

As an example we use the text line from Figure 4.3. The corresponding GT and the recognition results of five best models M1-M5 look like this:

in&e marien namen

Figure 4.3: Example line containing a strongly degraded “e”. Adopted from [234].

```

GT: inde marien namen
-----
M1: inide maricn namen
M2: inde maricn namen
M3: inde marien namen
M4: iade marien namen
M5: inde maricn namen

```

The alignment tool produces the following output using M1-M5 as inputs:

```

Aligned: i{1}de mari{2}n namen
-----
{1}: M1{ni}, M2{n}, M3{n}, M4{a}, M5{n}
{2}: M1{c}, M2{c}, M3{e}, M4{e}, M5{c}

```

The first line shows the aligned output. Curled parenthesis mark disagreements between two or more inputs. Afterwards, the different recognition results of the models are listed for each disagreement. Figure 4.4 shows a more descriptive and human-readable depiction of the alignment

$$i \left\{ \begin{array}{c} ni \\ n \\ n \\ a \\ n \end{array} \right\} de \text{ mari} \left\{ \begin{array}{c} c \\ c \\ e \\ e \\ c \end{array} \right\} n \text{ namen}$$

Figure 4.4: Alignment result of the individual output of the five voters.

4.1.2.5 Additional Information about the Recognition Confidence Values

During the recognition process the OCRopus 1 network determines the probabilities (represented by confidence values) of the output characters at each position in the text line as a distribution over the complete character set. The size of this set depends on the individual model and gets fixed at the start of the training process. To access this additional information some changes within the OCRopus 1 code had to be made. The confidence values are collected and stored for each line in an extended so-called *LSTM Location Of Characters (lloc)* file. See Table 4.2 for the first few llocs of M4.

From left to right the columns show the most likely character, the pixel coordinates of its start and end position, and its confidence. The rightmost column contains a list of alternatives with their respective confidences.

Two things are worth mentioning: First, the pixel positions of a character never exactly match the position of the actual CC in the line image as they are calculated directly from the network output. A character is only recognized if the probability of the epsilon

Table 4.2: Example llocs output from M4 for the word “inde” recognized as “iade” including the most likely character (*Char*), its start/end position (x_S , x_E), confidence (*Conf*) and its *Alternatives* including their respective confidences.

Char	x_S	x_E	Conf	Alternatives
i	120	123	87.54%	b=8.66%, f=2.94%
a	126	136	96.65%	n=45.78%, r= 23.65%, m=9.24%, k=8.32%, [...]
d	142	149	99.93%	ā=4.83%, V=4.17%, O=1.13%
e	155	160	99.15%	all alternatives < 1%

class (no character at this position) is below a certain threshold (OCRopus 1 default = 0.7). The start of a character is defined as the first pixel position after an epsilon where a character is recognized. The end of a character is the last pixel position a character was recognized before an epsilon is predicted. Usually the output positions are offset by a few pixels from the borders of the actual letter.

Second, the sum of all confidence values (representing a posterior probability distribution over all output nodes, i.e. all possible characters) per pixel position adds up to 100%. Because each glyph representing a character is several pixels wide, an alternative n-best recognition might occur at a different pixel position and the combined confidences of all alternatives will in general add up to values above 100%. For example, see the second row in Table 4.2: The recognized *a* has its confidence maximum of 96.65% somewhere between the start and end positions at 126 and 136 pixels, leaving only 3.35% to be distributed among all remaining characters. However, it is quite common that an alternative (in this example the *n*) is recognized at a different position than the top-1 character (*a*) between the start and end positions with a confidence lower than the maximum (96.65%) but significantly higher than the left over percentages at the position of the maximum. Naturally, the higher confidence at a different position is much more relevant to a confidence based voting.

4.1.2.6 Confidence Voting

After the alignment the confidence voting takes place. The aligned output is processed from left to right. Characters which could be matched for all inputs are accepted right away. To solve the disagreements between two or more inputs the corresponding llocs are identified and loaded. Since the disagreements can vary in length, first a majority vote takes place to determine the most likely length. Longer or shorter inputs are discarded (e.g. the output of M1 *ni* in the first disagreement shown in Figure 4.4). Of course, these inputs could still hold some valuable information, and therefore an alignment of these disagreements might make sense. But, preliminary tests showed that aligning the inputs

varying in length by applying a k -Means clustering on the recognition position led to a decline in accuracy for the voting result. There are two likely reasons for this: First, as explained above, the recognition position for a character can vary considerably along the glyph in the input image, and therefore lead to mismatches. Second, if a model already confused a single character as two or vice versa, the output information on the recognition output and the alternatives could be heavily flawed. For the unlikely case of a tie during the length voting, for example if the outputs have the lengths 1, 2, 2, 3, 3, the shorter option (2) is chosen to break the tie. This heuristic was implemented since it counters a common OCROpus 1 problem where the network outputs the same character twice because a blank (no character) is recognized within a glyph, and consequently enables the network to perform another output. During the final voting, the confidences for the actually recognized character and all relevant (confidence $> 1\%$) alternatives are summed up and the most likely ones are accepted.

In our example (see Table 4.3) the heavily degraded e at disagreement position $\{2\}$ got wrongfully recognized as a c by three out of the five models. Therefore, the simple majority vote leads to a c in the final output as does the confidence voting when only considering the actually recognized character. However, when incorporating the alternatives and their respective confidence values, the correct solution e is chosen.

Table 4.3: Confidences of the model outputs for the two characters in question (c and e) including the most likely character and its alternative as well as the confidence sum without (*Rec*) and with ($+ Alt$) including the alternatives.

	c	e
M1	<u>66.83%</u>	38.40%
M2	<u>93.27%</u>	19.77%
M3	-	<u>99.91%</u>
M4	7.56%	<u>98.02%</u>
M5	<u>90.31%</u>	50.07%
\sum Rec	250.41%	197.93%
$+$ \sum Alt	257.97%	306.17%

4.1.3 Experiments

In this section we describe the experiments and report the obtained results. Each model training was carried out until the recognition accuracy on the test set just showed variation due to statistical noise and no further improvement was expected.

4.1.3.1 Default Application (5 folds, 150 lines)

As a first experiment the number of folds N was set to 5 and 150 lines were used since this represents a magnitude which usually already yields good results without hitting the

4 Methods

point of diminishing return. Moreover, most modern PCs with multiple cores should be able to comfortably handle five parallel OCRopus 1 training processes. The experiment was conducted by following the workflow and example described in Section 4.1.2. Table 4.4 shows the CER achieved on a fixed evaluation set of previously unseen lines from the held-out data of each individual best model (1-5) and the combined results without (*ISRI Voting*) and with (*Confidence Voting*) confidence information. Furthermore, the relative improvement of the combined result with respect to the *best/average/worst* model is indicated.

The results clearly show that the cross fold training and voting process considerably reduced the CER for all books. The amount of errors corrected by combining the outputs varies from 16% for the best individual models to 62% for the worst ones. All improvements are highly significant on a better than 0.001 level with the χ^2 test [220]. Incorporating the confidence information approximately reduces the amount of errors by another 5% to 10%. This additional improvement is also highly significant except for 1572 and 1675 due to their relatively small number of lines used for evaluation (541 and 317 respectively).

4.1.3.2 Influence of the Number of Lines

In the next experiment the influence of the number of lines on voting was studied by varying them in six steps between 60 and 1,000 (see Table 4.5). Because of the previous results only confidence voting is considered. Furthermore, since in a real world scenario lacking held-out GT data there is no way to determine the best or worst of the five individual models, only average improvement is noted.

Cross fold training and confidence voting on 60 to 1,000 lines shows similar improvements as the previous experiments using 150 lines. Good outputs benefit more from voting than worse ones. However, the by far worst recognition result (1675, 60 lines) still shows a considerable improvement. For most books a medium amount of GT (150 to 250 lines) leads to the biggest decreases of the CER compared to the average of the individually trained models. It is especially noteworthy that voting is still very effective when combining the output of very high performance models: On all three books that surpassed a CER of 1% the average amount of remaining errors was reduced by at least one third.

4.1.3.3 Influence of the Number of Folds

To increase the degree of variety among the models even more the number of folds was set to 10. The size of the testing sets is given by the number of lines divided by the number folds. Consequently, a higher number of folds leads to less lines being used for testing and more for training. For example, when training five folds using 250 lines the train/test ratio is 200/50 for each model training. This ratio rises to 225/25 when increasing the number of folds to 10. Therefore, another experiment (5+) using five folds

Table 4.4: CERs and improvement rates of *Confidence/ISRI Voting* over the *best*, average (*avg*) and *worst* result of single models when using 5 folds and 150 lines of GT.

Year	CERs of the Best Model of each Fold					ISRI Voting		Confidence Voting	
	1	2	3	4	5	CER	Improvement over best/avg/worst	CER	Improvement over best/avg/worst
1476	3.93%	3.32%	4.07%	3.61%	3.41%	2.21%	35%	1.82%	45%
1488	2.87%	2.58%	2.54%	2.34%	4.23%	1.60%	32%	1.42%	40%
1495	3.97%	5.21%	6.16%	6.34%	4.34%	3.52%	11%	2.89%	27%
1500	3.10%	2.66%	2.87%	2.69%	2.82%	1.74%	35%	1.54%	42%
1505	5.29%	5.21%	4.93%	5.57%	4.67%	3.96%	15%	3.70%	21%
1572	1.62%	1.95%	1.74%	2.02%	1.99%	1.49%	8%	1.38%	15%
1675	10.93%	11.19%	11.81%	12.69%	11.26%	9.22%	16%	8.80%	20%
									27%
									44%
									62%
									44%
									44%
									29%
									26%
									20%
									20%
									27%
									44%
									45%
									51%
									44%
									45%
									29%
									26%
									20%
									20%
									27%

Table 4.5: CERs (top) and improvement rates (bottom) of confidence voting over the average result of single models when varying the number of lines.

	60	100	150	250	500	1,000
1476	7.55%	2.66%	1.82%	1.46%	1.20%	0.93%
	41%	50%	50%	44%	42%	35%
1488	4.76%	2.07%	1.42%	0.90%	0.74%	0.65%
	32%	53%	51%	45%	39%	34%
1495	8.05%	4.01%	2.89%	1.99%	1.57%	1.34%
	37%	47%	44%	42%	39%	37%
1500	2.81%	1.87%	1.54%	1.23%	1.06%	0.97%
	41%	43%	45%	39%	35%	33%
1505	5.82%	4.24%	3.70%	3.49%	2.63%	2.46%
	24%	27%	29%	28%	29%	27%
1572	1.90%	1.49%	1.38%	1.22%	0.98%	0.73%
	35%	29%	26%	22%	24%	31%
1675	14.48%	10.03%	8.80%	5.77%	-	-
	19%	27%	24%	32%	-	-

was conducted where fewer lines were added to the test sets in order to match the number of training lines during training with ten folds without altering the overall amount of GT lines. For practical reasons and since all books clearly showed the same tendencies, this final experiment was conducted using a subset of two books with comprehensive GT and varying CERs: 1476 and 1505. Table 4.6 shows the results.

Table 4.6: CERs when using 5 or 10 folds with different size of the training set (5+).

	1476			1505		
	5	5+	10	5	5+	10
150	1.82%	1.98%	1.78%	3.70%	3.71%	3.51%
250	1.46%	1.57%	1.24%	3.49%	3.31%	3.15%
1,000	0.93%	0.94%	0.86%	2.46%	2.34%	2.16%

Doubling the number of folds led to a decrease of the CER in all scenarios. This effect was smaller when using 150 lines. Adjusting the number of lines when using five folds always led to worse results compared to ten folds.

4.1.3.4 Time Expenditure

Table 4.7 shows the necessary computational time expenditure for different scenarios with a varying number of folds and a maximum number of training steps depending on the number of lines. All measurements were performed on a laptop with a quad core i5-6300HQ CPU @ 2.3 GHz and 8 GB RAM using multi-threading whenever possible. The speed of the OCRopus 1 training and prediction process depends on the length of the lines. Therefore, these measurements were performed on the book 1500, since its line lengths are closest to the average of all used books. The time expenditure for the training setup, alignment, and voting is negligible. The results show that the benefits of reduced error rates of our method can be reached by doing the necessary model training overnight.

Table 4.7: Required time expenditure in minutes for the processing (*Training*, *Recognition*, and *Sum*) of book 1500 (4,651 lines) using different scenarios in terms of training (*Fold x Lines*) and number of training iterations (*It.*).

	5x150 10k It.	5x250 20k It.	5x1,000 30k It.	10x1,000 30k It.
Training	177min	238min	381min	782min
Recognition	26min	26min	26min	52min
Sum	203min	264min	407min	834min

4.1.4 Discussion

Our experiments show that the proposed approach significantly improves the obtainable CER on early printed books. As expected, ATR results with a lower CER gain an even bigger boost compared to the more erroneous results. However, while not reaching the same improvement rates of up to over 50% even the worst ATR results still benefited greatly from gains of close to 30%.

The amount of available GT did not show a notable influence on the improvements achievable by confidence voting. Yet, a very high number of lines leads to a drop in voting gains for most books. This has to be expected for models that get closer to perfection as most of the remaining errors are unavoidable ones such as rare characters that could not be recognized because their corresponding glyphs were absent from the training set or highly degraded glyphs leading to misrecognition by any model. Both of these cases cannot be restored by voting.

Our first experiments on increasing the number of folds showed promising signs, especially when training with a large pool of GT. Adjusting the train/test ratio towards more training lines led to varying results depending on the number of GT lines. For small to medium amounts the CER stayed the same or even went up compared to the standard

4 Methods

80%/20% ratio. This indicates that choosing the best model based on a small number of lines can lead to models that perform well on the test data, but do not generalize well.

Moreover, our approach allows for a considerably more efficient use of the available GT. For example, in almost all cases the cross fold training and confidence voting based on 100 lines of GT significantly outperformed the default single model approach even when using 50% more lines.

Furthermore, it is worth mentioning that the CERs of the individually trained models vary strongly, producing up to 80% more errors. This kind of variance represents a big problem for the standard approach. Of course, in a real world scenario there is no way to determine if the training of a single model went well in terms of variance. Our robust approach does not suffer noticeably from a single flawed model, as is shown by e.g. 1488 in Table 4.4.

Finally, the experiment regarding time expenditure showed that even very comprehensive multi-fold training tasks can be performed by a standard system within a reasonable amount of time, e.g. overnight.

4.1.5 Conclusion and Future Work

In this sub chapter a method to significantly improve the CER on early printed books by utilizing cross fold training and confidence based voting was proposed. The results showed that our method works with any amount of GT which is typically used for training during the different stages of the ATR process on an early printed book.

Despite the very encouraging results there is still some work to be done: First, the alignment process of the llocs prior to confidence voting should be further optimized in order to allow an improved matching of disagreements with varying lengths. This is especially important since these variations happen due to insertions and deletions which are very common when dealing with historical printings. An initial solution approach would be to also consider blank outputs, i.e. positions where no character was predicted due to the lack of a likely candidate. Naturally, at these positions there are also alternatives available whose confidence information should be helpful for the voting. However, due to the particularities of blanks compared to real characters this integration is far from trivial.

Second, there are several parameters to be optimized during the cross fold training process like the optimal number of folds and the train/test split within a fold. The best choice mainly depends on the amount of available GT, but can also vary because of the accessible hardware or the overall ATR quality of the individual book. To be able to provide reliable recommendations for different scenarios further extensive tests are required.

Another promising option is to benefit from the diversity of the models obtained during cross fold training even further by implementing an AL approach. If additional training is required, the resulting outputs allow an informed instead of random selection of new

GT lines, e.g. by choosing the lines whose outputs differ the most, indicating a high degree of uncertainty across the models. We will follow up on this idea in Section 4.3.

Despite the focus on early printed books in this section the presented ideas can be applied to any given print. However, several adaptations might be needed since the ATR on newer books is usually not based on individual training but on highly performant so-called *omnifont* models. To investigate this area of application in greater detail, we perform a case study on 19th century Fraktur in Section 7.1, extending the proposed concept to the training and application of mixed models.

4.2 Transfer Learning Utilizing Pretrained ATR Models³

In this subchapter we present another method that significantly reduces the CERs for ATR results obtained from models trained on early printed books when only small amounts of diplomatic transcriptions are available. This is achieved by building from already existing models during training instead of starting from scratch. While the previous subchapter tackled the task of maximizing the quality of an ATR result by training several models and apply them as a voting ensemble, we now focus on the challenge of improving the effectiveness of individual models. Because real GT is both scarce and expensive to produce, it seems natural to look for other means to build workable models. As modern and historical font shapes (glyphs) are not totally different from modern ones, a simple idea is to reuse the models trained on modern fonts and use them as a starting point for continued training on some historical GT. Thus, one could hope to reach a certain level of CERs with less historical GT than if we trained a model from scratch. In the following we explore this idea and report some experiments that show if and to what extent this expectation is justified.

A note on terminology and a reminder regarding mixed and book-specific models: The alphabet on which a recurrent neural network is trained is also called *codec*, as the alphabet is internally represented by numbers to which the alphabet gets encoded and which at the end gets again decoded to alphabet characters. An individual or *book-specific* model is trained on a single book (which might contain different typesets, e.g. upright and italics) and is contrasted with a *mixed model* trained on GT relating to different books, which mostly also means different typographies (different fonts, different inter word distances). Models trained on synthetic material in different languages are also called mixed models by us even when all languages are represented by the Latin script using Antiqua fonts, as there are specific national typographic idiocracies leading to different codecs (e.g. the usage of accents in French texts, or different punctuation marks).

Section 4.2.1 describes related work, Section 4.2.2 gives details of the pretrained models and their respective GT used as well as our modifications of the OCRopus 1 code, Section

³This section is based on a previously published article [236]: C. Reul, C. Wick, U. Springmann, and F. Puppe, “Transfer learning for OCRopus model training on early printed books,” *027.7 Journal for Library Culture*, vol. 5, no. 1, pp. 38–51, 2017. [Online]. Available: <http://dx.doi.org/10.12685/027.7-5-1-169>

4.2.3 relates our experiments and their outcomes which are then discussed in Section 4.2.4. At the end follows Section 4.2.5 with the conclusions and ideas for future work.

4.2.1 Related Work

For a discussion of mixed models which are relevant to the application of pretraining an transfer learning in the area of historical ATR we refer to the corresponding section in the data chapter (cf. Section 3.3).

While to the best of our knowledge there is no suitable related work regarding transfer learning in the field of ATR available, it was applied successfully to a variety of other tasks. Yosinski et al. [325] performed experiments on the transferability of features in deep neural networks. They used the ImageNet data set [77], which at the time of the described experiments consisted of close to 1.3 million labeled training images and 50,000 test images, with each image labeled with one of 1,000 classes. After randomly splitting the classes in half they first performed a pretraining on one half before training and finally testing on the remainder. This approach yielded lower error rates compared to the default method, i.e. only training and testing on data with matching classes. So even after an extensive period of fine-tuning on suitable data, the features learned during the first steps still lingered and led to notably improved recognition accuracies.

Wick and Puppe [316] applied the same method using even more diverse data sets. In order to assign the correct species to images of leafs they first performed a pretraining on the Caltech-256 data set [114], consisting of over 30,000 images assigned to 256 classes like animals, tools, vehicles, or fictional characters. Afterwards, they built from the obtained network by training on real leaf images. Despite the diversity of the two sets of training data, the pretraining showed a significant positive effect on the classification accuracy.

Of course, these examples of transfer learning used far deeper networks than for example OCRopus 1 with only a single hidden layer, resulting in a dramatically increased number of parameters and consequently, more opportunities to learn and maintain useful low-level features. Nonetheless, we still expect a noteworthy impact of pretraining, since scripts in general should be expected to show a higher degree of similarity than for example oak leafs and Homer Simpson.

4.2.2 Materials and Methods

We first very briefly introduce our evaluation corpus consisting of books from various corpora. Our approach is expected to work best with models trained on data as similar as possible to these books. Hence, we use a fitting model trained on fitting historical data but distinct from the evaluation corpus. In addition, two less similar mixed standard models trained on newer types are utilized. Finally, some necessary changes regarding the OCRopus 1 code are described, which enable us to extend and reduce the codec available to a model in a flexible way.

4.2.2.1 Books

The experiments are performed on seven early printed books (see Table 4.8 and cf. Section 3.2.1 for further details). To avoid unwanted side effects only lines from running text parts are used and headings, marginalia, page numbers, etc. are excluded. Figure 4.5 shows some example lines. The books 1495, 1500, 1505, and 1509 are editions of the Ship of Fools. Despite their similar content, these books differ considerably from an ATR point of view since they have been printed in different print shops using varying typefaces and languages (Latin, German, and Dutch). 1488 and 1572 are part of the Kallimachos Corpus and 1476 belongs to the ENHG Corpus. All books above the horizontal line in Table 4.8 were printed in broken scripts (Fraktur in the wider sense), the rest used Antiqua types.

Table 4.8: Books used for Evaluation.

ID/Year	Language	GT Train	GT Test
1476	German	1,000	2,000
1488	German	1,500	2,678
1495	German	1,000	1,114
1500	Dutch	1,250	1,250
1505	Latin	1,500	1,789
1509	Latin	1,500	1,500
1572	Latin	791	750

Dem hoeghen marschalck sent quijn
 dienen. Da was der Keyser gar fro dz sein syn
 en vnd materien erlengert vnd sch
 Sonder beghinzel Volmaect in Dreughden.
 Stultoz ꝛc. Om̄ inquā stultoz vt
 milimus Alexander Barclay sui ipsius
 complexus sit: ut sub æquatore est

Figure 4.5: Different example lines from the seven books used for evaluation. From top to bottom: excerpts from books 1476, 1488, 1495, 1500, 1505, 1509, and 1572. Adopted from [236].

4.2.2.2 Mixed Models

As our first model we used LH which was trained on historical books printed in Latin. Additionally, we applied the freely available OCRopus 1 standard models FRK and ENG introduced in [46]. For further information we refer to Section 3.3

4.2.2.3 Utilizing Arbitrary Pretrained Models in OCRopus 1

OCRopus 1 in its original form already allows to load existing models and continue training from there. However, the default functionality only covers the case where a training is stopped (deliberately or not) and restarted using the exact same alphabet. While this suffices to ensure that the training process does not get lost, it cannot be applied to material with additional characters. Therefore, the following adjustments on code level had to be made. The corresponding source code is available at GitHub⁴.

4.2.2.4 Extending the Codec

While mixed models are usually trained on a variety of different books and therefore comprise a rather comprehensive alphabet it still is likely for them to sooner or later encounter previously unknown characters. For any (mixed) model it is impossible to recognize these glyphs it has never seen during training, so these glyphs constitute *blind spots* for the recognition process. Even worse, if a character is not part of the codec it can never be learned even if fitting training examples are available. Consequently, the model must be able to grow.

Figure 4.6 illustrates the extension and reduction (see next section) of the codec. The bidirectional LSTM-based network, among others used by OCRopus 1, consists of two layers. A single LSTM layer processes each pixel-wide slice of the text line, thus its input dimension equals the line height in pixel. The number of time steps T equates the line length. The LSTM layer produces a vector h for each time step. Its size H remains fixed for each model and is given by the number of states in the single hidden layer of the network (OCRopus 1 default = 100). The last layer represents a matrix multiplication where the weight matrix M is multiplied with the current h producing an output o for each character in the codec. Each character is represented in M by a vector of size H , containing the weights determined during the training process. Consequently, the dimensions of M are H times C , with C being the codec size. The predictions with probability $P(c)$ for each character c in the codec is generated by applying a softmax [59] function to o . Since each single character in the codec is given by a row in M , a codec extension can be achieved by adding additional rows to M . For each attached row, an additional entry in the output is appended. Yet, the application of the softmax function ensures that the output remains a valid probability distribution $P(c)$. The new weights in M are initialized randomly and have to be trained to produce meaningful results.

⁴https://github.com/ChWick/ocropy/tree/codec_resize

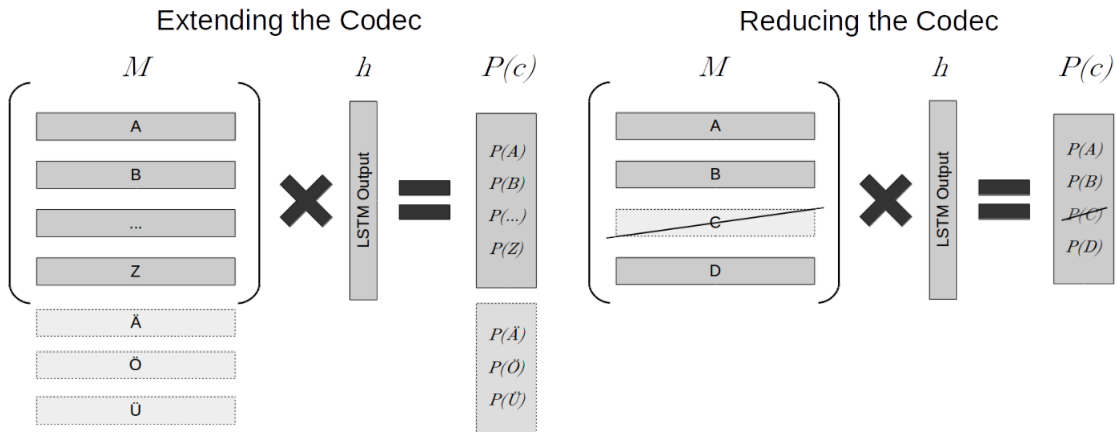


Figure 4.6: Schematic view of extensions (left) or reductions (right) of the output matrix of the network whose rows correspond to the codec. Adopted from [236].

4.2.2.5 Reducing the Codec

The just described problem regarding characters missing from the codec could of course be bypassed by simply bloating the codec. However, this is impractical for two reasons. First, the bigger the codec the slower the training and recognition process becomes. Second, when refining a mixed model towards an individual one for a single book the goal is to minimize the number of recognizable characters without risking blind spots. Naturally, a large codec also makes misrecognitions more likely, especially if it contains several very similar characters. For example, LH contains several *e* characters with various diacritica on top, e.g. *éêëë*, which are customarily employed in early printings. However, in books that do not contain these diacritics they only add potential for confusions.

The right sketch of Figure 4.6 shows the process of removing single characters from the output matrix M . By deleting a complete row the corresponding output probabilities $P(c)$ is removed, too. Retraining the network is not necessary since the softmax ensures that the output still is a valid probability distribution.

4.2.2.6 Defining a Whitelist Containing Immune Characters

Especially when working with small amounts of GT it is likely that these transcribed lines do not comprise all characters that occur throughout the entire book. In this case applying the approach described above will lead to blind spots. Therefore, we implemented a so-called *Whitelist* (WL) containing characters that will not be removed from the codec even if they do not occur in the GT used for training: a-z, A-Z, 0-9.

4.2.3 Experiments

In order to examine our hypothesis that building from an existing model holds clear advantages compared to training from scratch (from now on also referred to as the “default” model/approach/training/...) we perform several experiments whose outcomes are reported in this section. After explaining the general methodology of our training and evaluation procedure, we conduct the first experiment comparing the default approach with a training starting from the LH model. Next, since suitable models in terms of printing type, age, and language are often not available, we test the OCRopus 1 standard models ENG and FRK and still hope for improvements compared to the default training. Furthermore, we expect the gains of our pretraining approach to correlate with the number of lines used for training. Since more lines lead to stronger models, the room for improvement gets smaller and we therefore await smaller gains. In our final experiment we replace the mixed LH model by a model trained on a single similar book with the expectation to achieve even bigger improvements.

4.2.3.1 Setup and Methodology

The initial setup consisted of two main steps. First, for each book about half of the available GT was set aside for evaluation. The remaining individual GT was split up in five different training/test sets on which models were trained and their results averaged to reduce the impact of variance. To ensure maximal comparability, both, the initial training/test/evaluation split as well as the individual training sets were kept fixed for all experiments.

Analogously to the procedure mentioned in Section 4.1, the actual model training using OCRopus 1 was always carried out for a fixed number of iterations until no further notable improvements were observed. A number of 10% to 15% of the training lines were set aside before training to act as a test set in order to determine the best model, i.e. the one that produced the lowest CER on the test set. Finally, the best models are used to recognize the held out evaluation data to determine the final result.

4.2.3.2 Building from the Latin Mixed Model

In this first experiment we compare a training starting from the LH model with one starting from scratch. When using the LH model all trainings were performed twice, once with building the codec from the available GT and once with adding the WL as described in Section 4.2.2.6. All experiments were carried out for 60 and 150 lines of GT since usually 60 lines are a good starting point and 150 lines represent just enough lines to already train relatively strong individual models (150) without reaching the point of diminishing return. Table 4.9 sums up the results.

The achieved CERs show that building from a mixed model leads to superior individual models compared to using the available GT by itself. As expected, the improvement rates

4.2 Transfer Learning Utilizing Pretrained ATR Models

Table 4.9: Resulting CERs when using the raw Latin Hist model (*LH only*), models trained from scratch (*Def*) and by building from the LH model without (*LH*) and with (*+WL*) utilizing the WL. The last row shows the average (*AVG*) for training with 60 (52 Training, 8 Test) respectively 150 (130 Training, 20 Test) lines of GT. The books above the horizontal line between 1505 and 1509 are printed using broken script (basically Fraktur), the ones below utilized Antiqua types. All CERs and improvement rates are given in %.

Book	LH only CER	60 Lines of GT				
		Def CER	LH CER	LH Gain	+WL CER	+WL Gain
1476	31.12	8.21	5.35	35	5.17	37
1488	35.28	7.60	3.53	54	3.49	54
1495	42.79	12.67	6.26	51	6.14	52
1500	37.61	5.03	3.58	29	3.42	32
1505	17.23	6.19	5.32	14	4.79	23
1509	5.05	6.31	2.85	50	2.06	67
1572	10.40	2.43	1.58	35	1.61	34
AVG	25.64	6.92	4.07	38	3.81	43

Book	LH only CER	150 Lines of GT				
		Def CER	LH CER	LH Gain	+WL CER	+WL Gain
1476	31.12	4.00	3.11	22	3.04	24
1488	35.28	2.88	2.22	23	2.22	23
1495	42.79	5.83	4.03	31	4.04	31
1500	37.61	2.95	2.42	18	2.29	22
1505	17.23	3.70	3.43	7	3.40	8
1509	5.05	2.81	2.24	20	1.44	49
1572	10.40	1.72	1.27	26	1.26	27
AVG	25.64	3.41	2.67	21	2.53	26

decrease with more GT for training and increase with adding a WL of basic characters. The average gain when utilizing 60 lines of GT is 43%, from a CER of 6.92% without pretraining to 3.81%. This is nearly as good as using a considerable more expensive GT of 150 lines without pretraining, having a CER of 3.41%. With pretraining (including the WL), a CER of 2.53% is achieved using 150 lines of GT, with an average gain of still 26% over the default approach. Interestingly, the improvements do not necessarily correlate with the performance of the LH model on its own. For example, the book where the LH model did worst on (1495) still experiences one of the highest boosts among all books.

4 Methods

Since adding the WL shows a clearly positive effect (average gain of 5%) all remaining experiments were performed by including the WL.

The gained accuracies vary considerably. For example, book 1505 shows the least improvement over the default approach (but still 23% and 8%, respectively). Most likely this is caused by the fact that the distances between two characters in book 1505 are considerably smaller compared to all other books used for training and testing (see Figure 4.5, line 5). Another interesting result is that 1509 profits considerably from the inclusion of the WL, resulting in a CER drop from 2.24% to 1.44% when using 150 lines of GT. Apparently, in this case the training GT was missing characters that frequently occurred in the evaluation data.

4.2.3.3 Utilizing the OCRopus 1 Standard Models

The creation of high quality historical mixed models is a cumbersome task and there are not many publicly available. Therefore, we investigated the effect of pretraining on a mixed model trained on different but easily available data, in this case using the OCRopus 1 standard models ENG and FRK introduced in section 4.2.1. Table 4.10 sums up the results.

Although the gains of the ENG und FRK models are slightly lower than for the more similar LH model, they are still impressive: 33% on average for training with 60 lines of GT and 19% (ENG) and 17% (FRK), respectively for training with 150 lines of GT compared to the default approach. As expected, ENG outperforms FRK on the books using Antiqua types (books 1509 and 1572), while FRK has higher gains for Fraktur types (books 1476, 1488, 1495, 1500, and 1505).

4.2.3.4 Varying the Number of Lines

To further test the applicability of our approach, we repeated some of the experiments by varying the amount of GT in five steps from 30 to 60, 100, 150, and 250 lines. For reasons of clarity the results of only three representative books (1476, 1495, and 1572) are displayed in Figure 4.7. The remaining books showed the same tendencies.

As expected and in line with previous experiments, the achievable improvements decrease when increasing the amount of available GT. While for a small number of lines (30 and 60) the CER is reduced by at least one third and up to two thirds, this effect almost vanishes for most books when approaching 250 lines.

4.2.3.5 Incorporating Individual Models

Next, we want to examine if building from a model trained on an individual book similar to the new data can yield even better results than the mixed model approach we utilized thus far. We measured similarity by determining the CER obtained by models trained

Table 4.10: Resulting CERs from models trained by following the default approach (*Def*) compared (*Gain*) to building from the Latin Hist model (*LH*) and the Standard OCRopus 1 models *ENG* and *FRK*. The books above the horizontal line between 1505 and 1509 are printed using broken script (basically Fraktur), the ones below utilized Antiqua types. All CERs and improvement rates (*Gain*) are given in %.

Book	60 Lines of GT (52 Training, 8 Test)						
	Def	LH		ENG		FRK	
	CER	CER	Gain	CER	Gain	CER	Gain
1476	8.21	5.17	37	5.21	37	4.49	45
1488	7.60	3.49	54	4.32	43	4.12	46
1495	12.67	6.14	52	6.89	46	6.31	50
1500	5.03	3.42	32	4.11	18	3.49	31
1505	6.19	4.79	23	5.44	12	5.09	18
1509	6.31	2.06	67	2.94	53	4.09	35
1572	2.43	1.61	34	1.91	21	2.25	8
AVG	6.92	3.81	43	4.40	33	4.26	33
Book	150 Lines of GT (130 Training, 20 Test)						
	Def	LH		ENG		FRK	
	CER	CER	Gain	CER	Gain	CER	Gain
1476	4.00	3.04	24	3.21	20	3.12	22
1488	2.88	2.22	23	2.68	7	2.38	17
1495	5.83	4.04	31	4.12	29	3.89	33
1500	2.95	2.29	22	2.50	15	2.47	16
1505	3.70	3.43	7	3.45	7	3.53	7
1509	2.81	1.44	49	1.93	31	2.40	15
1572	1.72	1.26	27	1.25	27	1.57	8
AVG	3.41	2.53	26	2.73	19	2.77	17

on individual books and by the mixed models LH, ENG, and FRK on the GT data of the new book. For books 1476, 1505, 1509, and 1572 the LH model performed best and they were therefore excluded from further experiments. The 1488 model achieved the lowest CER on 1495 and vice versa and 1500 got recognized best by the individual model of 1476. Consequently, we trained new models for 1488, 1495, and 1500 by building from the models of 1495, 1488, and 1476, respectively. Of course, each individual model was excluded from the pool when processing the book it was trained on. Table 4.11 shows the obtained results.

The results do not show a clear tendency: in three cases, pretraining with the mixed LH model showed higher gains, and in the other three cases, pretraining with the best fitting individual model led to better results. Neither approach shows a significant gain over

4 Methods

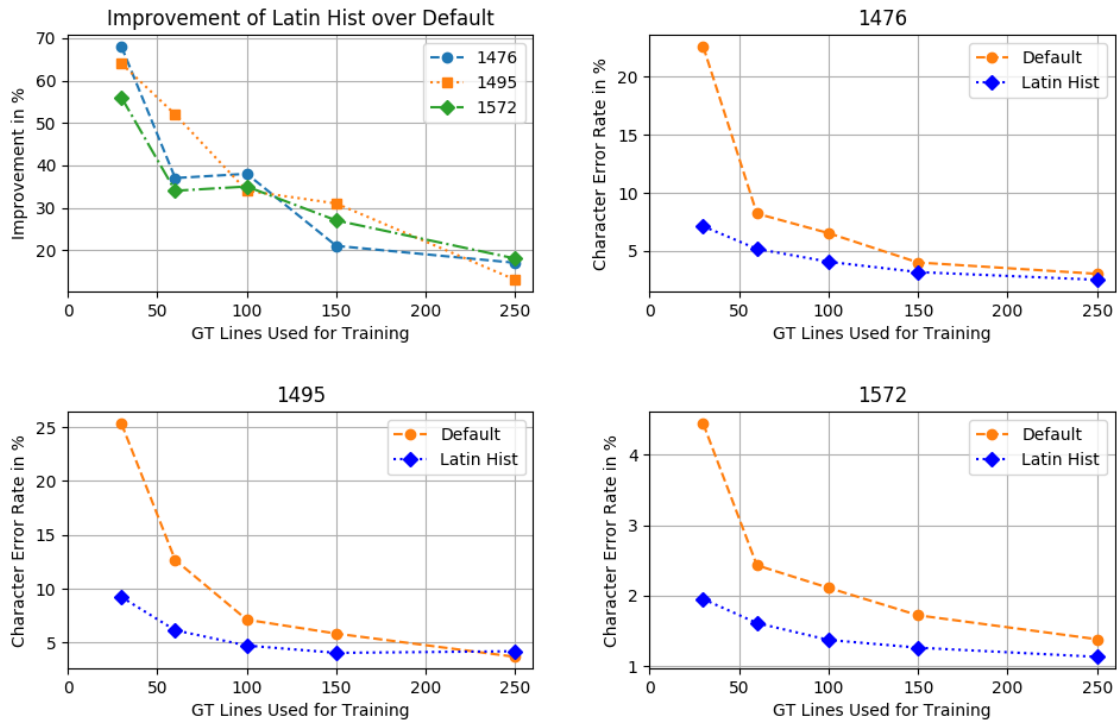


Figure 4.7: Effects of building from the LH model compared to the default approach for a varying number of lines showing the improvement rates for three different books (top left) and the resulting CERs for 1476 (top right), 1495 (bottom left), and 1572 (bottom right). Adopted from [236].

the other. From this experiment we cannot infer that it is worthwhile to incorporate individual models compared to the robust mixed model for pretraining. However, it has to be said that even the best fitting models only achieved CERs of around 16% or even worse. Therefore, higher gains should be expected when building from individual models, which already fit even better to the new data.

4.2.4 Discussion

Our experiments showed that building from a pretrained model can significantly reduce the obtainable CER compared to starting the training from scratch. The achievable improvement rates decrease with an increasing number of GT lines available for training. The effect of a WL used to prevent blind spots is reduced when adding more lines since the likelihood for missing characters in the training data goes down.

The evidence that even completely unrelated mixed models also lead to considerable improvement indicates that a pretrained model offers much more than an accurate description of the type(s) it was trained on. Despite the shallow structure of the OCRopus 1 network with only one hidden layer the training seems to benefit a lot from

Table 4.11: Resulting CERs from models trained by following the default approach (*Def*) compared (*Gain*) to building from the Latin Hist model (*LH*) and the best fitting individual model. All CERs and improvement rates given in %.

Book	Def	60 Lines of GT (52 Training, 8 Test)						
		Mixel Model (LH)			Best Fitting Individual Model			
		Raw	Trained	Gain	Model	Raw	Trained	Gain
1488	7.60	34.56	3.49	54	1495	15.58	3.23	58
1495	12.67	43.26	6.14	52	1488	16.26	5.82	54
1500	5.03	37.23	3.42	32	1476	27.67	4.58	9

Book	Def	150 Lines of GT (130 Training, 20 Test)						
		Mixel Model (LH)			Best Fitting Individual Model			
		Raw	Trained	Gain	Model	Raw	Trained	Gain
1488	2.88	35.42	2.22	23	1495	16.07	2.35	19
1495	5.83	42.95	4.04	31	1488	16.49	3.58	36
1500	2.95	37.60	2.29	22	1476	27.52	2.66	10

low level features that generalize well like general character shapes, different forms and severity of glyph degradation as well as an improved robustness against noise.

Not a single case occurred in our experiments where the proposed approach had a noteworthy negative impact on the recognition result. This seems sensible, since the weights of the network are initialized randomly when training from scratch causing the network to be unable to output anything during the beginning of training before slowly learning the most frequent characters like whitespaces, *e* and *a*. It seems that a pretrained model, which might not match the types at all but at the very least partly fits the required codec and is able to distinguish between character and non-character, benefits the training process more than a random initialization. Since the additional required effort when building from a model is negligible, our results imply that it is sensible to prefer the pretrained approach over training from scratch, especially when only a low to medium amount of GT is available.

4.2.5 Conclusion and Future Work

A method to significantly improve the CER on early printed books by building from pretrained models instead of training from scratch was proposed. Our experiments showed that adding fresh GT to an existing model outperforms the default training approach even if GT and model differ considerably, in particular if only a small number of transcribed lines is available. Despite our focus on very early prints using Latin script the experiments suggest that the proposed method should work with a wide variety of prints with diverse scripts and languages and different periods.

An obvious and very promising task is the combination of the proposed pretraining approach with the cross fold training and voting procedure introduced in Section 4.1. For the voting to be successful the participating models are not only required to be precise but also diverse. We have shown that a wide variety of mixed models is likely to have a positive effect on the training outcome. This makes the training of individual models by building from completely different mixed models a very attractive option to gather several powerful, yet highly variant models. A thorough evaluation of a combined approach is given in the upcoming sub chapter.

Furthermore, additional models would be very useful for real world application scenarios, since a suitable model to start training from can save hours of transcription effort. This includes several types: mixed models like LH which are created by collecting and combining real life data, as well as synthetically trained mixed models like ENG or FRK, but also book-specific models. Of course, it is also possible to combine several approaches, for example by taking a small subset of the LH data and train a new model building from ENG or FRK. With a growing repository of available models, it makes sense to narrow down the selection before testing on the available GT to find the best fitting model. This can be done by taking attributes like age, the printing type (Antiqua or Fraktur) or, if applicable, specifics like very small inter character distances into consideration. Thus, the gain of building from pretrained models can be further optimized.

4.3 Combining Pretraining, Voting, and Active Learning⁵

After showing the effectiveness of cross fold training and confidence voting in Section 4.1 as well as proving the apparent superiority of utilizing pretrained models instead of starting from scratch in Section 4.2, it stands to reason to combine both approaches. As suggested in Section 4.1, we suspect that voting ensembles can also be used in order to identify possible training lines the existing models can learn the most from. Therefore, we experiment with an Active Learning (AL) approach, ensuring that lines showing maximal disagreement among the available voters are included in the training set to maximize the learning effect.

Section 4.3.1 briefly summarizes the results obtained during the preceding voting and pretraining experiments and discusses related work regarding AL. In Section 4.3.2 we describe the printing material which the experiments of Section 4.3.3 are based on. Section 4.3.4 contains the discussion of our results and we conclude the subchapter with Section 4.3.5.

⁵This section is based on a previously published article [233]: C. Reul, U. Springmann, C. Wick, and F. Puppe, “Improving OCR Accuracy on Early Printed Books by combining Pretraining, Voting, and Active Learning,” *JLCL: Special Issue on Automatic Text and Layout Recognition*, vol. 33, no. 1, pp. 3–24, 2018. [Online]. Available: https://j1cl.org/content/2-allissues/1-heft1-2018/j1cl_2018-1_1.pdf

4.3.1 Related Work

In this section we first very briefly sum up the findings obtained during the previous experiments dealing with the application of cross fold training and confidence voting as well as transfer learning or pretraining in the area of historical ATR. Second, we give an overview over some basic AL concepts.

4.3.1.1 Voting

In Section 4.1 we implemented a cross fold training procedure with subsequent confidence voting in order to reduce the CER on early printed books. Our experiments led to the following observations:

1. For all experiments the cross fold training and voting approach led to significantly lower CERs compared to performing only a single training. Gains between 19% and 53% were reported for several books and different number of lines of GT.
2. ATR results with a lower CER benefitted even more than more erroneous results.
3. The amount of available GT did not show a notable influence on the improvements achievable by confidence voting. Yet, a very high number of lines leads to a drop in voting gains for most books.
4. Increasing the number of folds can bring down the CER even further, especially when training on a large set of lines. However, five folds appeared to be a sensible default choice.
5. The confidence voting always outperformed the standard sequence voting approach by reducing the amount of errors by another 5% to 10%.

4.3.1.2 Pretraining and Transfer Learning

In Section 4.2 we used existing mixed models as starting points for the book-specific training of seven early printed books. From our experiments we arrived at the following conclusions:

1. Building from a pretrained model significantly reduced the obtainable CER compared to starting the training from scratch.
2. Improvement rates decrease with an increasing number of GT lines available for training. While models trained on only 60 lines of GT gained over 40% on average over starting from scratch, this number went down to around 15% for 250 lines.
3. The incorporation of a WL for standard letters and digits which cannot be deleted from the codec even if they do not occur in the training GT showed an additional average gain of 5%.

4. Even the mixed models for modern English and 19th century Fraktur, which were completely unrelated to the individual books in terms of printing type and age of the training material, led to significant improvements compared to training from scratch.

4.3.1.3 Active Learning

Settles [254] gives a very comprehensive overview over the literature dealing with AL. Apart from introducing different usage scenarios and discussing theoretical and empirical evidence for the application of AL techniques they define a typical AL scenario as follows: A learner starts out with access to a (possibly very small) pool of labeled examples to learn from. In order to improve performance it is possible to send queries consisting of one or several carefully selected unlabeled examples to a so-called *oracle* (teacher/human annotator) who then returns a label for each example in the query. Afterwards, the learner can utilize the obtained additional data. Naturally, the progress of the learner heavily depends on the examples selected to be labeled. Furthermore, the goal is to learn as much as possible from as few as possible queried examples, keeping the oracle's effort to a minimum.

One of the most successful query techniques is called *query by committee* and was introduced by Seung et al. [255]. The basic idea is that a committee of learners/models/voters is trained on the current labeled set. Each member of the committee is allowed to cast a vote on a set of query candidates, i.e. unlabeled examples. The assumption is that the candidate the voters disagree most on is also the one which offers the biggest information gain when being added to the training set. This is called the *principle of maximal disagreement*.

Among others, the effect of this approach was demonstrated by Krogh and Vedelsby [150] who trained five neural networks to approximate the square-wave-function. They performed 2x40 independent test runs starting from a single example and using passive and active learning. While the next example was chosen randomly during the passive tests the networks always got handed the example with the largest ambiguity among the five voters out of 800 random ones. Evaluation showed that AL led to a significantly better generalization error and that the individual additional training examples on average contributed much more to the training process when chosen according to the principle of maximal disagreement.

As for ATR, Springmann et al. [279] performed some initial experiments on selecting additional training lines in an active way. After recognizing lines with a mixed model they tested several strategies according to which they chose lines for further transcription. The best result was obtained when using a mixture of randomly selected lines combined with lines with low confidence values. It is worth mentioning that after transcribing these lines they started their training from scratch since the pretraining approach introduced above had not been developed, yet.

4.3.2 Materials and Methods

In this section we first introduce the early printed books and mixed models we used for our experiments. Then, we show how the principle of maximal disagreement can be utilized in order to choose additional training lines in an informed way within an iterative AL approach.

4.3.2.1 Books

The experiments were performed on six early printed books (see Table 4.12 and cf. Section 3.2.1 for further details). The AL experiments were carried out on the three books above the horizontal line. We focused on these books as they provided a large amount of GT which is needed to perform the procedure. In a real world application scenario it would be sensible to choose the additional training lines by recognizing all lines without GT and choose the worst ones. Therefore, as many lines as possible are required to be able to evaluate this scenario. To avoid unwanted side effects resulting from different types or varying line lengths only lines from running text were used and headings, marginalia, page numbers, etc. were excluded. 1505 represents an exception to that rule as we chose the extensive commentary lines instead, as they presented a bigger challenge due to very small inter character distances and a higher degree of degradation. Figure 4.8 shows some example lines.

Table 4.12: Books given by their respective *ID/Year* used during the experiments as well as the number of GT lines set aside for *Training*, *Evaluation*, and *Active Learning*.

ID/Year	Language	Training	Evaluation	Active Learning
1476	German	1,000	1,000	750
1488	German	1,000	1,000	1,928
1505	Latin	1,000	1,000	1,039
1495	German	1,000	1,114	-
1500	Dutch	1,000	1,250	-
1572	Latin	1,000	1,098	-

It is worth mentioning that all the books, which will be used for evaluation, were disjoint with the training materials of the mixed models.

4.3.2.2 Active Learning

As explained in Section 4.3.1.3 the basic idea behind AL is to allow the learners, i.e. the different voters, to decide which training examples they benefit the most from instead of selecting additional lines randomly. Since in a real world application scenario there usually is no GT available for potential new training lines, we cannot just use the ones

Dem hoeghen marschalck sent quirtjn
 Sienen. Da was der Keyser gar fro dz sein syn
 en vnd materien erlengert vnd sch
 Sonder beghinfel Volmæct in Dreughden.
 Stultoz ꝛc. Qm̄ inquā stultoz vt
 complexus fit: ut sub æquatore est

Figure 4.8: Different example lines from the six books used for evaluation. From top to bottom: excerpts from books 1476, 1488, 1495, 1500, 1505, and 1572. Adopted from [233].

for which our current models give the worst results, i.e. the ones with the highest CER. However, we can still identify the lines we expect to be most suitable for further training by following the principle of maximal disagreement.

After recognizing a line with each model of an ensemble consisting of n voters, all outputs are compared to each other in pairs. As a measure for the difference between two ATR output strings a , b , we define the Levenshtein Distance Ratio (LDR) $LDR(a, b)$ as the Levenshtein distance [164] between a and b , divided by the maximum string length of a and b . These ratios are then summed up between all pairs of different ATR output strings and divided by the number of pairs $n(n-1)/2$ to yield the average LDR_\emptyset . The pseudo-code in Algorithm 1 details our calculations.

Algorithm 1: The calculation of a line’s average LDR (LDR_\emptyset). $Lev(a, b)$ denotes the Levenshtein distance between two outputs a and b . The length (= the number of characters) of an output a is given by $|a|$.

Data: line image without GT, ensemble of n voters ($n > 1$)

Result: the LDR_\emptyset of the line

$outputs \leftarrow recognizeWithAllVoters(line)$

$sum = 0$

foreach a_i, a_j with $(i < j) \in outputs$ **do**

| $LDR(a, b) \leftarrow \frac{Lev(a, b)}{\max\{|a|, |b|\}}$

| $sum \leftarrow sum + LDR(a, b)$

end

$LDR_\emptyset = \frac{sum}{n(n-1)/2}$

Finally, after processing various lines, they are sorted in descending order according to their LDR_\emptyset s. In a real world application scenario the lines are handed to a human

4.3 Combining Pretraining, Voting, and Active Learning

annotator who then produces GT by transcribing them or by correcting one of the individual ATR results or the output of the confidence voting, if available. While the idea of the whole process is to give the committee the lines it requested, it is still up to the human annotator to decide whether a line is suitable for further training or not. For example, a line might have been badly recognized due to a severe segmentation error or due to an unusually high degree of degradation making recognition pretty much impossible. These lines cannot be expected to make a noteworthy contribution to the training process and are therefore discarded.

4.3.3 Experiments

In order to evaluate the effectiveness of the methods described above we performed two main experiments: First, the voting and pretraining approaches were combined by performing the voting procedure with models which were not trained from scratch but started from one or several pretrained mixed models. Second, the voters resulting from the first experiment served as a committee during an AL approach following the principle of maximal disagreement.

Since the train/test/evaluation distribution of the GT lines has changed, the results can differ from the ones obtained from earlier experiments in sections 4.1 and 4.2. Based on the previous results we chose to implement the following guidelines for all of the upcoming experiments.

1. The number of folds during cross fold training and consequently the number of voters is set to 5.
2. ATR results are always combined by enforcing the confidence voting approach.
3. Whenever pretraining is used a generic minimal WL consisting of the letters a-z, A-Z and the digits 0-9 is added to the codec.
4. Each model training is carried out until no further improvement is expected (e.g. 30,000 iterations for 1,000 lines of training GT; analogously to sections 4.1 and 4.2).

4.3.3.1 Combining Pretraining and Voting

Naturally, the combination of voting and pretraining seems attractive and should be evaluated. The number of lines used for training was varied in six steps from 60 to 1,000. Each set of lines was divided into five folds and the allocation was kept fixed for all experiments. We used two different approaches for pretraining. First, we trained the five voters by always building from LH since it yielded the best results during previous experiments. Second, we varied the models used for pretraining. We kept voters 1 and 2 from the first setup (trained from LH). For voters 3 and 4 FRK was utilized as a starting point since it was trained on German Fraktur fonts which are somewhat similar to the

4 Methods

broken script of the books at hand. Only one voter (5) was built from ENG as it was the least similar one out of the available mixed models regarding both age and printing type of the training data. This setup was slightly adapted for book 1572 as it was printed using Antiqua types. Therefore, in this case one of the two FRK folds was pretrained with ENG instead.

The idea was to still train strong individual models while increasing diversity among them, hoping for a positive effect on the final voting output. The results are summed up in Table 4.13. For reasons of clarity, detailed numbers are only provided for three books, i.e. the ones which will be used for further experiments. The general behaviour averaged over all books can be seen in Figure 4.9 and an overview over the progress made by adding more training lines is presented in Figure 4.10.

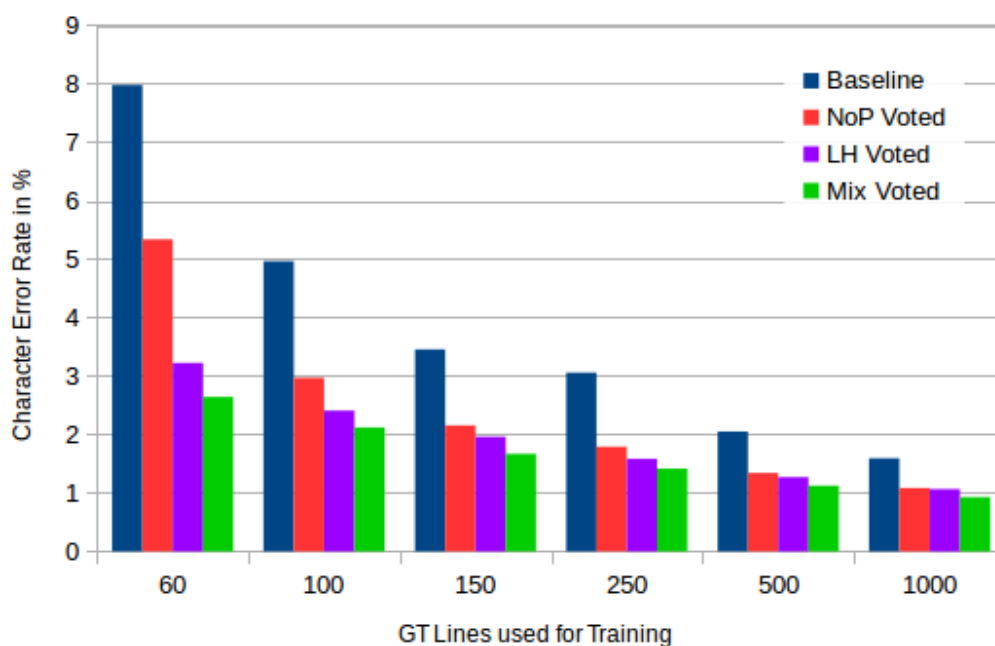


Figure 4.9: Comparison of the CERs (averaged over all books for each set of lines) of four different approaches: *Baseline* (no pretraining, no voting), *NoP* (no pretraining, voted), *LH* (all five folds trained from the LH model, voted), and *Mix* (mixed pretraining, voted). Adopted from [233].

In the majority of cases the combination of pretraining and voting considerably outperforms both the default voting approach showing gains of 14% (LH) and 26% (Mix) as well as the default pretraining approach showing gains of 29% (LH) and 39% (Mix) when averaging over all six books and lines. As expected, the best improvements can be achieved when using a small number of GT lines, resulting in gains ranging from 40% (LH) and 51% (Mix) for 60 lines to 2% (LH) and 14% (Mix) for 1,000 lines.

4.3 Combining Pretraining, Voting, and Active Learning

Table 4.13: CER in % of combining pretraining (*Single Folds*) and voting (*Voting Result*). *Single Folds* contain the baseline without pretraining (*Base*), pretraining with LH model (*LH*), and with a mixture of models (LH, LH, FRK, FRK/ENG, ENG) (*Mix*). *Voting Result* shows the results of different voters based on no pretraining (*NoP*), pretraining with *LH*, and and *Mix*. The *Improvement* columns show the voting gains of LH over NoP (*NL*), Mix over NoP (*NM*), Mix over LH (*LM*), and Mix over the base (*BM*). The underlined CERs represent the starting points for the upcoming AL experiment.

<u>1476</u> Lines	Single Folds			Voting Result			Improvement			
	Base	LH	Mix	NoP	LH	Mix	NL	NM	LM	BM
60	8.12	5.58	4.96	4.72	4.10	2.79	13%	41%	32%	66%
100	6.82	3.99	3.92	3.49	2.67	<u>2.23</u>	23%	36%	16%	67%
150	4.10	3.15	3.03	2.47	2.14	1.66	13%	33%	22%	60%
250	3.24	2.40	2.37	1.70	1.63	<u>1.47</u>	4%	14%	10%	55%
500	2.11	1.73	1.75	1.17	1.13	1.03	3%	12%	9%	51%
1000	1.55	1.30	1.22	0.97	0.88	0.75	9%	23%	15%	52%

<u>1488</u> Lines	Single Folds			Voting Result			Improvement			
	Base	LH	Mix	NoP	LH	Mix	NL	NM	LM	BM
60	7.28	3.97	4.28	4.38	3.05	2.50	30%	43%	18%	66%
100	4.19	2.85	3.20	2.73	2.06	<u>1.84</u>	25%	33%	11%	56%
150	2.96	2.26	2.33	1.81	1.51	1.24	17%	31%	18%	58%
250	2.59	1.82	1.89	1.29	1.21	<u>1.07</u>	6%	17%	12%	59%
500	1.50	1.40	1.38	0.91	0.95	0.79	-4%	13%	17%	47%
1000	1.17	1.06	1.13	0.71	0.72	0.61	-1%	14%	15%	48%

<u>1505</u> Lines	Single Folds			Voting Result			Improvement			
	Base	LH	Mix	NoP	LH	Mix	NL	NM	LM	BM
60	6.54	5.00	5.27	4.58	3.70	3.45	19%	25%	7%	47%
100	4.54	3.96	4.15	3.16	2.82	<u>2.68</u>	11%	15%	5%	41%
150	3.54	3.16	3.16	2.34	2.27	2.02	3%	14%	11%	43%
250	2.85	2.66	2.18	1.98	1.77	<u>1.60</u>	11%	19%	10%	44%
500	2.24	2.18	2.11	1.59	1.60	1.43	-1%	10%	11%	36%
1000	1.84	1.85	1.82	1.35	1.40	1.26	-4%	7%	10%	32%

<u>All</u> Lines	Single Folds			Voting Result			Improvement			
	Base	LH	Mix	NoP	LH	Mix	NL	NM	LM	BM
60	7.98	4.42	4.49	5.34	3.22	2.64	40%	51%	18%	67%
100	4.97	3.38	3.49	2.97	2.41	2.12	19%	29%	12%	57%
150	3.46	2.78	2.87	2.15	1.96	1.67	9%	23%	15%	52%
250	3.06	2.28	2.20	1.79	1.59	1.42	11%	21%	11%	54%
500	2.05	1.86	1.77	1.34	1.27	1.12	5%	16%	12%	45%
1,000	1.59	1.46	1.42	1.08	1.07	0.93	2%	14%	13%	42%
Avg.	3.85	2.70	2.69	2.45	1.92	1.65	14%	26%	13%	53%

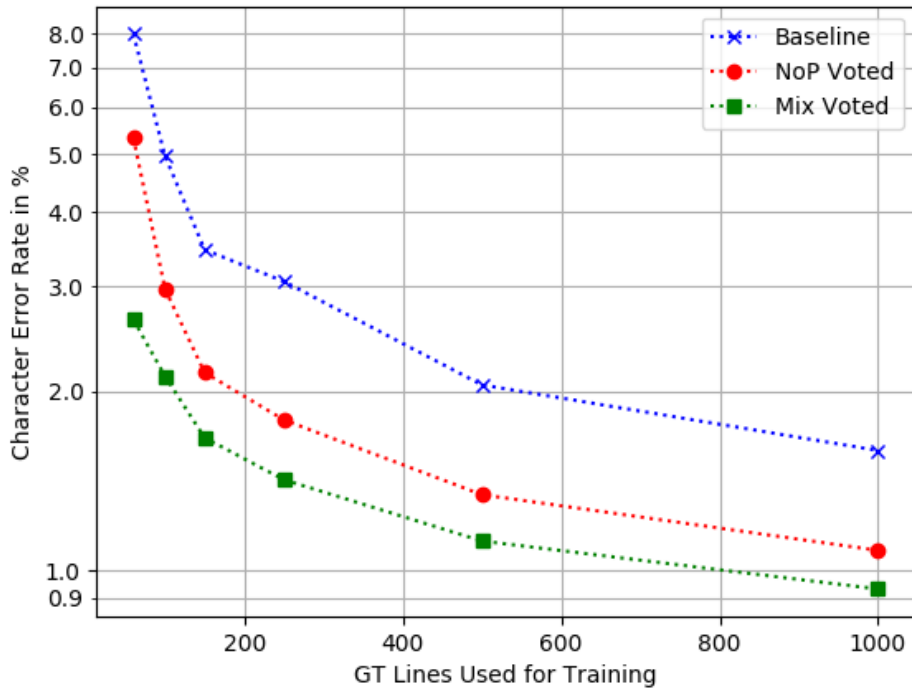


Figure 4.10: Influence of the number of GT training lines compared for the approaches *Baseline* (no pretraining, no voting), *NoP Voted* (no pretraining, voted), and *Mix Voted* (pretrained with different mixed models, voted) on a logarithmic scale for CER. Adopted from [233].

Overall, the average CER on individual folds pretrained with LH (2.70%) is effectively identical to the one achieved by building from a variety of mixed models (2.69%). However, in the case of applying the voting procedure over all five folds, the Mix approach yields considerably better results than just using LH as a pretrained model, leading to an additional reduction of recognition errors by over 13% without an apparent correlation regarding the amount of training GT.

Comparing the best method (Mix + Voting) with the baseline, i.e. the default OCRopus 1 approach (training a single model without any pretraining or voting), shows the superiority of the proposed approach yet more clearly. Even the error rates of strong individual models trained on a 1,000 lines of GT are reduced by more than 40% on average. In general, a substantial amount of GT (>250 lines) is required in the standard OCRopus 1 training (see “baseline” in Figure 4.9) to match the result achieved by a mere 60 lines when incorporating mixed pretraining and voting, indicating a GT saving factor of 4 or more. A similar factor can be derived when considering the average baseline CER for 1,000 lines of GT.

4.3.3.2 Incorporating Active Learning

To select additional training lines we utilized the models (voters) obtained from the previous experiment. Each model recognized the GT lines set aside for AL and the best candidates were determined by choosing the ones with the highest LDR_{ϕ} as explained in Section 4.3.2.2. Next, the candidates underwent a quick visual inspection in order to sort out lines where a positive impact on the training was considered highly unlikely due to very rare segmentation errors or extreme degradations. Indeed, this adds a bit of subjectivity to the task but we expected the decision whether to keep or drop a line to be trivial in most cases. However, in our experiments we never skipped a line proposed by the AL approach despite coming across several borderline cases which will be discussed below.

We performed two experiments starting with different numbers of (base) GT lines (100/250) which we kept from the previous experiment. For the passive learning approach we added an additional 50% (50/125) of randomly selected lines. This was performed five times and the results were averaged. As for AL we chose the lines by following the principle of maximal disagreement incorporating the LDR_{ϕ} . Since the number of characters per line may vary we made an effort to select only as many lines as necessary to match the average number of characters in the passive learning approach.

After selecting the lines the base fold setup was kept and we distributed the additional GT evenly over the five folds to ensure an effect on the training itself but also on the selection of the best model. Afterwards, the training was started from scratch/from the default mixed models while the voters were discarded.

Since the previous experiment showed the superiority of the mixed pretraining approach we decided to omit the pretraining using LH during the upcoming experiments. Table 4.14 shows the results.

Incorporating AL leads to lower CERs for four out of six tested scenarios. While important improvements with an average gain of almost 27% can be reported for 1476 and 1488, 1505 does not improve at all (see the discussion in the next section).

Moreover, it is worth mentioning that no clear influence of the number of the GT lines available for training can be inferred on the basis of these results. Even when starting from an already quite comprehensive GT pool of 250 lines AL yielded an average gain of 16% compared to randomly chosen lines.

Finally, Figure 4.11 sums up the results by comparing the baseline to the best pretraining (Mix) approach combined with confidence voting with and without AL.

4.3.4 Discussion

The experiments show that the combination of pretrained models and confidence voting is an effective way to further improve the achievable CER on early printed books. While the obtainable gain is highest when the number of available GT lines is small, a substantial

4 Methods

Table 4.14: Results of comparing active to passive learning. *Base Lines* is the number of lines used to train the voters of the previous iteration. The lines added (*Add. Lines*) randomly (*Rdm*) or by maximizing the LDR_ϕ (*AL*) correspond to 50% of the base lines. Compared are the resulting error rates (*CER*) after performing a confidence vote and (in case of *Rdm*) an averaging calculation. Finally, (*Avg. Gain*) shows the average improvement of the voters trained by AL.

Book	Base		Add. Lines Rdm/AL	CER		Average Gain
	Lines	CER		Rdm	AL	
1476	100	2.23	50	1.80	1.31	26%
	250	1.47	125	1.18	0.90	24%
1488	100	1.84	50	1.54	1.05	32%
	250	1.07	125	0.86	0.65	24%
1505	100	2.68	50	2.17	2.21	-2%
	250	1.81	125	1.60	1.57	2%

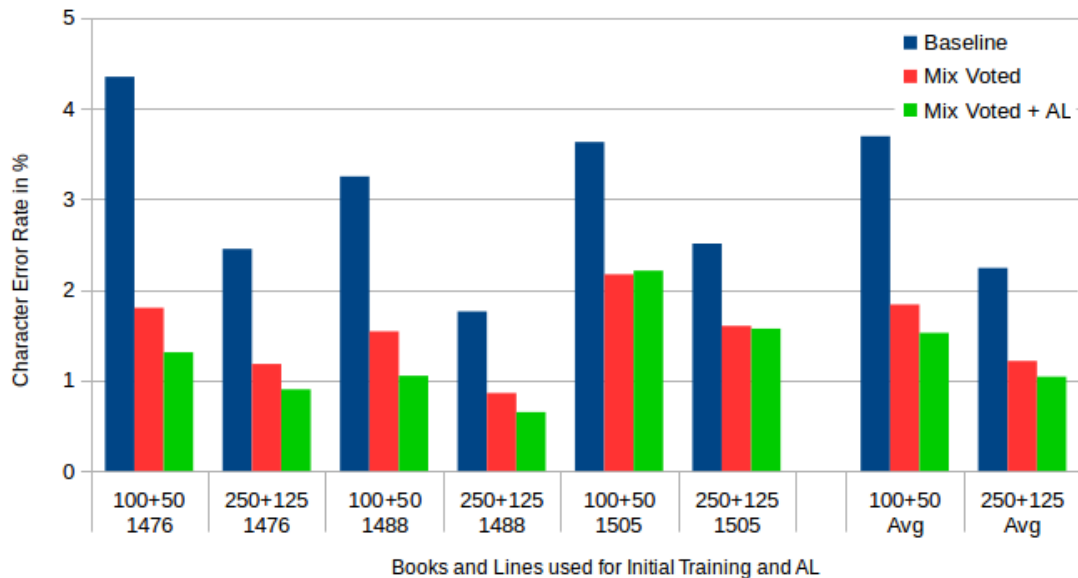


Figure 4.11: Results of the AL experiments for three books at two different sets of lines comparing the *Baseline* (no pretraining, no voting), *Mix Voted* (pretrained with different mixed models, voted) and *Mix Voted + AL* (Mix with additional lines chosen by AL). Adopted from [233].

reduction of ATR errors can still be expected even when training with several hundreds of lines.

An interesting result of our experiments is that the variability of the voters has a clear

4.3 Combining Pretraining, Voting, and Active Learning

influence on the voting result and can even outweigh a superior individual quality of the single voters. This explains why using a variety of models for pretraining considerably outperformed N -fold training from the LH model (as clearly shown by the last line of Table 4.13) even though it represented the best fitting one of the available mixed models.

Since training from an available model skips the random initialization of values in the weight matrix for pre-existing characters an important chance of introducing diversity to the training process is skipped, resulting in quite similar models even when trained on different but still heavily overlapping folds of training GT.

Following the proposed approach of combining a mixed pretraining with confidence voting allows for a substantially more efficient use of the available GT. On average we were able to achieve the same results as the standard OCRopus 1 approach requiring less than one fourth of the number of GT lines to do so. Moreover, a tiny amount of GT – only 60 lines – was enough to reach an average CER of close to 2.5%. Only two out of six books showed a CER greater than 3% but comfortably surpassed this value when adding another 40 lines of GT, raising the average to almost 2% CER, which is already considered good enough for many areas of application.

Despite these improvements, there are opportunities for optimizing the achieved results even further, e.g. in applications where the goal is to manually check and correct an entire book in order to obtain a CER of close to 0%. The experiments showed that our method also significantly outperforms the standard approach when training on a very comprehensive GT pool of 1,000 lines, resulting in an average CER of less than 1%. As an example, Figure 4.12 gives an impression of input (scanned page image) and ATR output for the 1488 printing with a CER of 0.60% reached by training on 1,000 lines combining pretraining using a variety of mixed models and confidence voting.

Even if the transcription of an entire book is intended, the goal still should be to minimize the CER by investing the least possible manual correction effort. Therefore, an iterative training approach makes sense and an efficient selection of further training lines is important. Our experiments on AL showed that choosing additional lines in an informed manner can offer an even more efficient way to use the available GT. Despite one of the books not responding at all to the proposed method, the three evaluated books still showed an improvement of 17% compared to the mixed voting approach when selecting additional training lines randomly for transcription. It is worth mentioning that during our experiments the number of lines presented to the committee of voters was considerably smaller than in a real world application scenario where it might be sensible to take as many as possible, if not all, yet unknown lines into consideration in order to choose the most promising ones in terms of information gain. This might have a positive effect on the achievable results of the AL approach.

A likely explanation for the lack of improvements of the 1505 book by AL is the degree and type of degradations of the lines queried for training by the voting committee which is illustrated by some example lines shown in Figure 4.13. The lines on the left are examples the voters fully agreed on and, as expected, got recognized correctly by the base models trained with 100 lines of GT. On the right some of the lines for which the

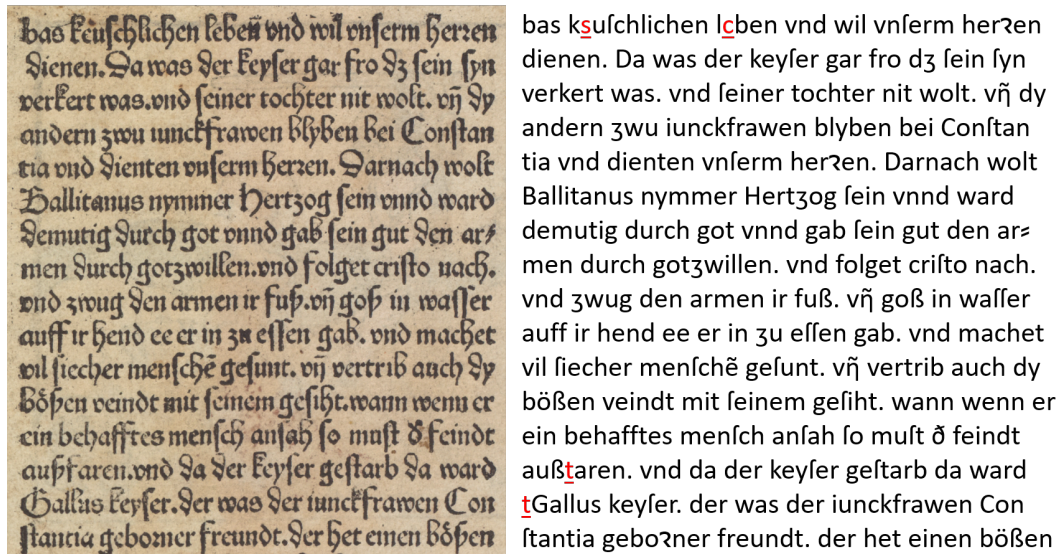


Figure 4.12: Snippet of a scanned page for 1488 and its best ATR output by combining mixed pretraining and voting. The resulting CER of 0.60% was achieved by training on 1,000 GT lines. The remaining four recognition errors are marked in red and underlined. Adopted from [233].

committee disagreed the most are shown, i.e. the ones with highest LDR_{ϕ} . For 1476 (top) the lines shown had a ratio of 0.45 and CER of 9.44%. There are some signs of degradation, mostly moderately faded glyphs. The worst lines selected from 1488 (middle, 0.72 LDR_{ϕ} , 33.68% CER) mostly suffered from noise while the glyphs of 1505 (bottom, 0.64 LDR_{ϕ} , 30.77% CER) frequently show severe deformations. This might be a sensible explanation why AL works very well for 1476 and 1488 but not at all for 1505. Despite the fading and the noise the glyphs of 1476 and 1488 look much more regular than the deformed ones of 1505, at least to the human eye. Therefore, the models trained for 1476 and 1488 using AL learned to see through the effects of fading and noise and earned additional robustness, resulting in a considerable gain in CER. In the case of 1505 the AL models were fed many lines showing severe but very irregular degradations which may have led to an increased robustness but probably did not improve the recognition capability of regular lines as much as the passive learning lines did.

4.3.5 Conclusion and Future Work

This section proposed a combination of pretraining, confidence voting, and AL in order to significantly improve the achievable CER on early printed books. The methods were shown to be very effective for different amounts of GT lines typically available from several hours' transcription work of early printings.

Dat dye nuysser tsamen moedych Der nuysser starcke moelen torn Garmhartzich gnedige frauw	U guedt ind oick v lijff freu wylch rieffē sy stait vast yr fromē riter ind Zo mynrebwoedē ind cloren
fagen ir solt Sye abgötter anbeten Des will er vnd bereydet Sy schar wol nach weyßheyt. vñ vmb Seinen Dienst Den Du mir gerhan hast. vñ	zu nam. gen freuden. Den boten gar ser. vñ embot sant Antoni hin
te aut inopia: z in sua versutia z opibus fensozē fugiet colonus. i. agricola sine vi. i. mutabile ac transitorio: iū se cōuertunt:	med s tenebris. Prouer. xx. et incl. quinto. curare mortalia aut nullā vindictā facturū

Figure 4.13: Example lines of the three books 1476 (top), 1488 (middle), and 1505 (bottom) which were presented to the committee. The left column shows perfectly recognized lines (LDR_{ϕ} of 0.0). On the right some of the most erroneous lines are presented (LDR_{ϕ} of 0.45 (1476), 0.72 (1488), and 0.64 (1505)). Adopted from [233].

In the future we aim to utilize the positive effect of more diverse voters even further. In general, a higher degree of diversity can be achieved in two ways:

1. By the inclusion of more mixed models for pretraining. Since there are only a few good mixed models freely available to date, there is a great need and sharing is key. To lead by example we made part of our GT data and models available online⁶.
2. By varying the network structure used for training representing a viable leverage point to increase diversity even further.

Clearly, the combination of both intended improvements also represents a very promising approach.

To optimize the achievable results, extensive experiments regarding parameter optimization are required. This includes the number of folds/voters, the kind of network they have been trained on, and the method of combination of mixed models used for pretraining, as well as the number of lines the training is performed on.

As for AL an important additional approach is to not only utilize it in order to choose new training lines before the actual training process but also to get involved during the training itself. The standard approach is to randomly select lines and feed them to the network. A more efficient method might be to decrease the chance to get picked for lines which already got perfectly recognized and consequently increase it for lines which still cause the current model a lot of problems.

Concerning the maximal disagreement approach for line selection, it would be interesting to experiment with other ways to determine those lines that offer a maximal information

⁶https://github.com/chreul/OCR_Testdata_EarlyPrintedBooks

4 Methods

gain. Utilizing the ATR engine’s intrinsic confidence values that we already use during our voting approach comes to mind but also measures like the *Kullback-Leibler-Divergence* [154].

Finally, despite our focus on early printed books the proposed methods are applicable to newer works as well. Especially 19th century Fraktur presents an interesting area of application. Since the typically used Fraktur typesets are more regular than those of the books used during our experiments the goal is to produce a mixed model with excellent predictive power and to avoid book-specific training at all. This topic will be addressed in greater detail during the corresponding case study in Section 7.1.

5 OCR4all

This chapter focuses on the OCR4all software. After explaining its general idea, we first introduce the data structure, describe the workflow and its individual modules, including their input/output relations, in detail before looking at the encapsulating web GUI which offers various possibilities to influence the workflow by manual corrections or configurations.

5.1 Goals, General Idea, and Historical Development

As discussed during the introduction the main goal of OCR4all is to enable non-technical users to perform OCR on even the earliest printed books completely on their own while still meeting the highest standards regarding layout markup and ATR quality. To facilitate this, the entire workflow, starting from the scanned images and ending with machine-actionable text, had to be made available to the users in a comprehensible and controllable way. As the goal was to utilize and build from existing state-of-the-art open-source solutions as much as possible, a suitable method to run and distribute the software had to be found which is discussed in detail in Section 5.8.1. Since fitting open-source tools often rely on the usage of the command line, which is unfamiliar and error prone for non-technical users, a GUI had to be provided allowing the users to comfortably operate each of the incorporated tools via a uniform interface and without having to deal with their individual particularities. In addition, powerful interactive correction tools allowing to interfere at basically any stage of the workflow in order to minimize consequential errors.

In addition, OCR4all was initially designed to support the OCR of (very) early printed books and was geared towards the production of quotable text, that can be used in digital editions and aims to be free of errors. Furthermore, the goal was to allow the users to pursue their very own ideas regarding the analysis and markup of complicated layouts, often including a fine-grained semantic distinction already on layout level in order to allow a complete reconstruction of the scan based on the collected information later on, if desired (see Figure 5.1 for an example). Combined with the, due to the highly variant typography and the desired perfectionism regarding ATR accuracy, imperative book-specific model training the processing of a book requires a certain amount of manual effort. However, we of course neither wanted to restrict the users to the processing of very early printed books nor wanted to force them to always invest manual effort into interactive layout markup and book-specific training, and therefore added further

functionality to ensure a fluent passage towards a fully automated approach when dealing with later and more uniform works.

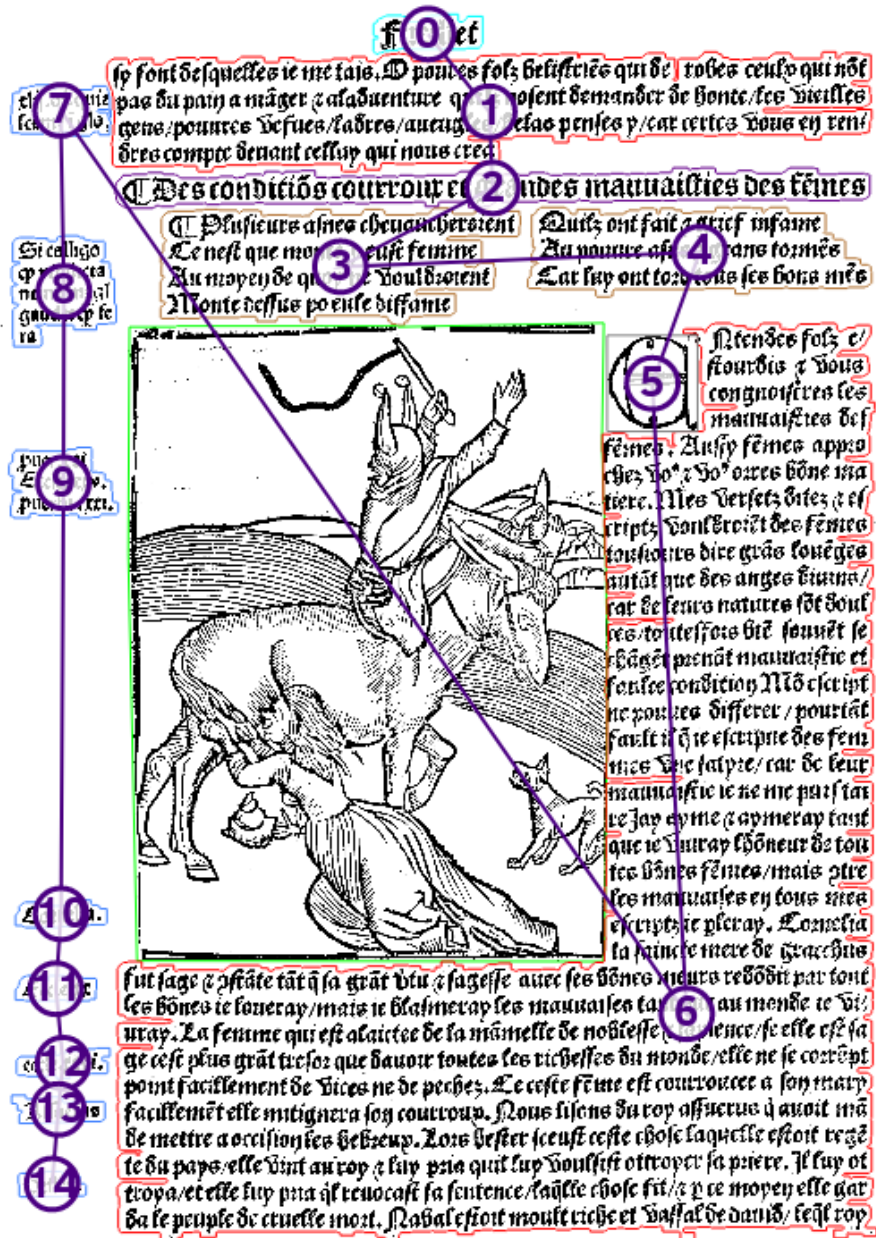


Figure 5.1: Example segmentation result of an early printed book page in line with very specific user requirements regarding the fine-grained semantic distinction of layout regions (indicated by different polygon colours) and the reading order (pink numbers and their connecting line).

5.2 Data Structure

Regarding our data structure, apart from different representations of page images we focus on PageXML (cf. Section 2.7) as the main carrier of information. This allows for a modular integration of the main submodules of OCR4all and sets up easy to fulfill requirements regarding interfaces, ensuring a reasonably straightforward addition of new submodules or replacement of existing ones. Additionally, defining a unified interface for all tools and modules enables the usage of a comprehensive post processing functionality. Another positive side effect of this approach is that submodules developed by us for OCR4all can be integrated analogously into other OCR workflows that use PageXML.

During the setup procedure a database containing two folders, *data* and *models*, is mounted from the host system into the docker container. While the first one allows to directly add new input files into the system and provides access to the final output as well as the interim results, the second one enables the user to import external ATR models or to extract trained models, for example in order to share them with fellow OCR4all or Calamari users. To add a new book the user simply has to create a new project folder within the *data* folder including an *input* folder containing the scans as single images or as one or several PDF files. During the processing of a step the resulting images and PageXML files (see the individual modules below for a detailed description) are kept in a *processing* folder before the final results can be generated as *output*. Figure 5.2 shows an example of the folder and data structure in OCR4all.

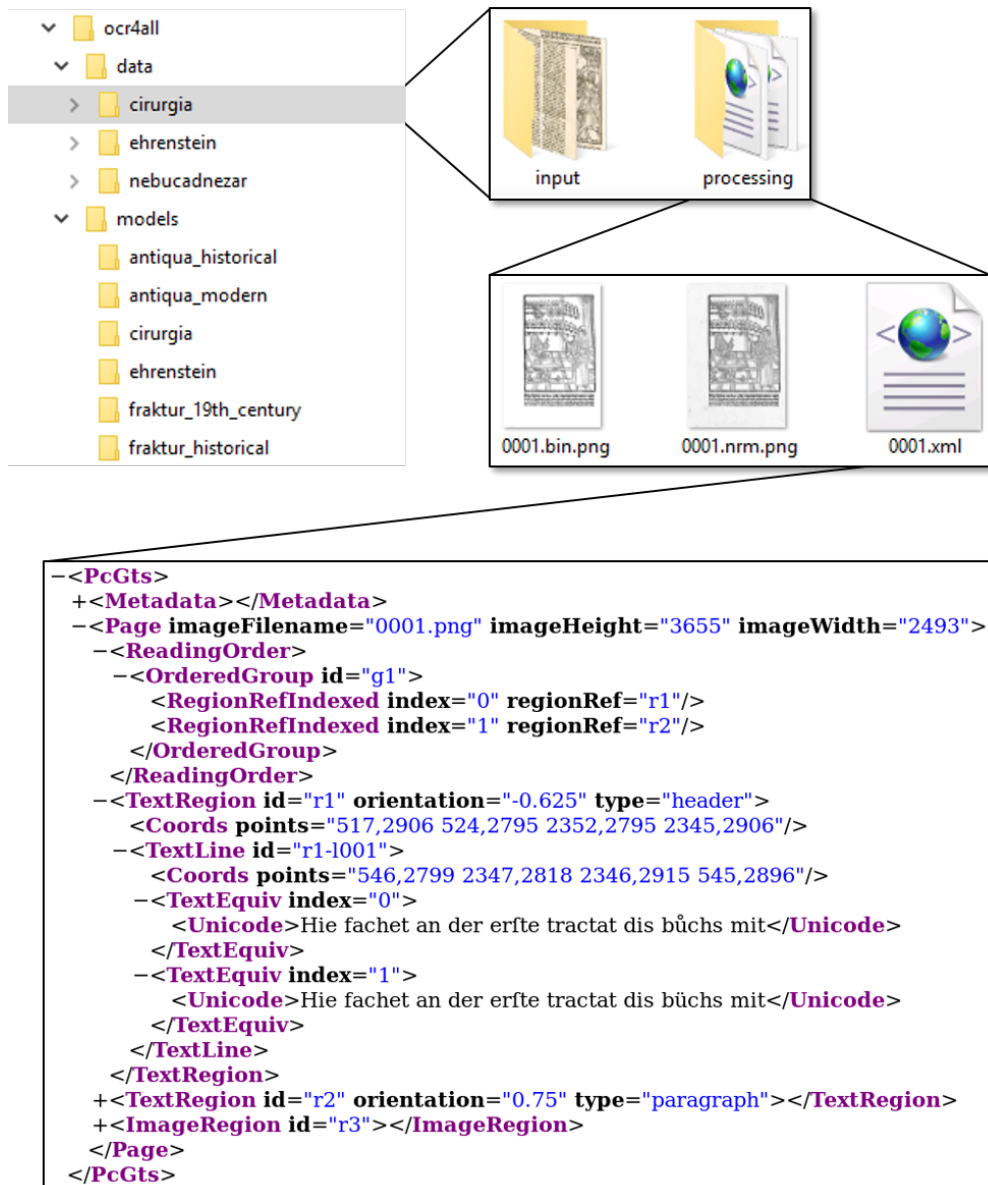


Figure 5.2: Example of the OCR4all folder and data structure. The *data* folder contains three books: *cirurgia*, *ehrenstein*, and *nebuchadnezar*. In the *models* folder several book-specific (e.g. *cirurgia*) and mixed (e.g. *antiqua_historical*) models are stored. Each book comprises an *input* folder that contains the original images. All intermediary results are kept in the *processing* folder, including a binary (*.bin.png*) and grayscale (*.nrm.png*) image as well as a PageXML (*.xml*) file for each page. All relevant intermediary and final results are stored within the PageXML: the *ImageRegions* and *TextRegions* with their semantic *types*, *ReadingOrder*, *orientation*, *coordinates*, and *lines* including their *coordinates*, ATR result, and GT (*TextEquiv 1* and *0*, respectively).

5.3 Workflow

Figure 5.3 shows the steps of the workflow implemented in OCR4all. After acquiring the scans and an optional preparation step the original images can be placed into the workspace. Next, image preprocessing is applied to the scans before several steps, like region segmentation and extraction as well as line segmentation, produce line images required as input for the ATR or GT production. The output of the character recognition can either directly serve as the final result or can be corrected by the user which enables a training of more accurate book-specific models, yielding better recognition results.

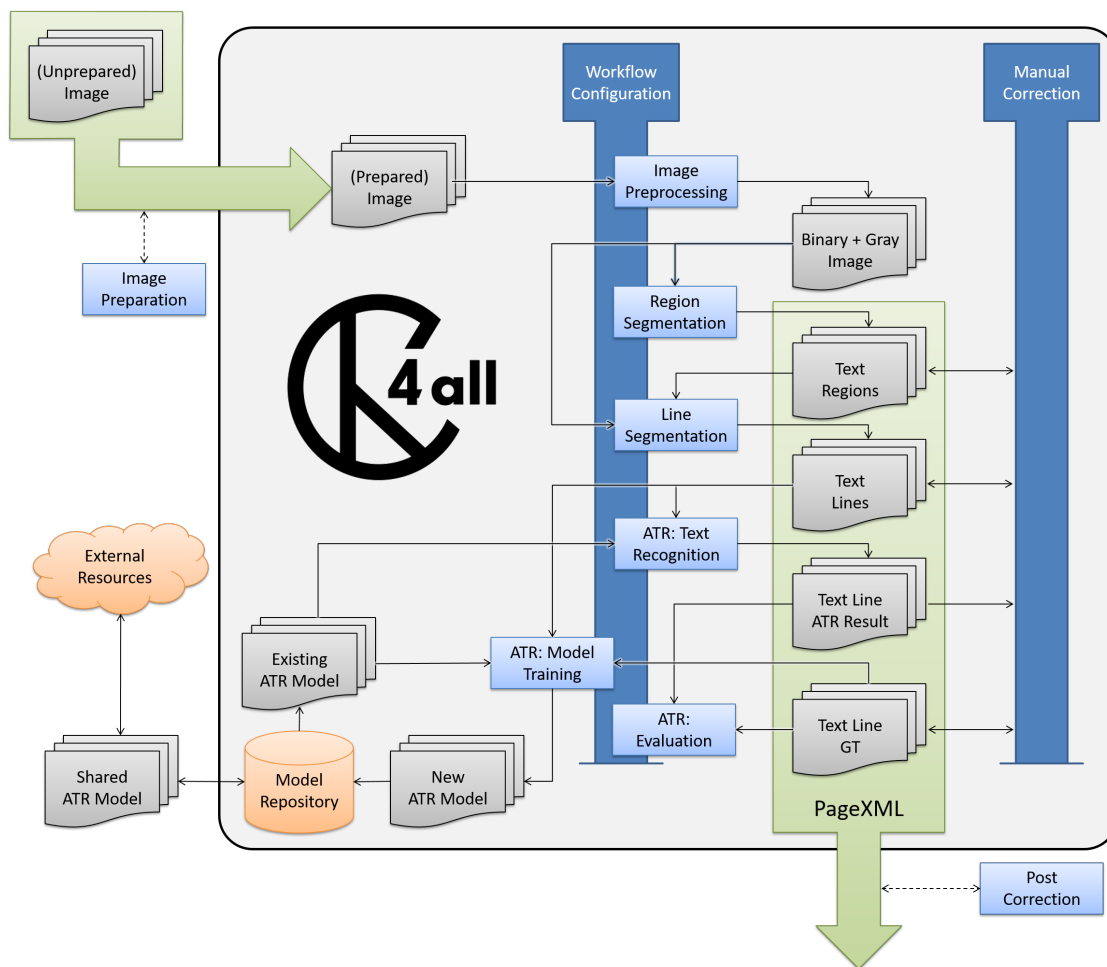


Figure 5.3: The main steps of the OCR4all workflow as well as the optional image preparation and post correction steps which are not part of the main tool (yet). Adopted from [229].

In the following we discuss a typical workflow by going through the first three main steps from Figure 1.1 (preprocessing, segmentation, and ATR) and discuss the corresponding

modules in OCR4all as shown in Figure 5.3. Furthermore, we always state the input and output relation of each module by describing the actual data each module works on, which is often produced by combining the information stored as PageXML with the preprocessed grayscale or binary image.

5.4 Preprocessing

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

5.4.1 Image Preparation

Input: unprepared image (image containing two scanned pages, a page rotated in an undesired orientation, ...)

Output: prepared image (single page in an upright position)

OCR4all expects the input images to be in an upright position and already segmented into single pages which can easily be achieved by using ScanTailor (cf. Section 2.2.2.1). Furthermore, it is recommended to remove excessive amounts of scan background, although this is not mandatory. Figure 5.4 shows an example of a valid and an invalid input, which also represents a possible input and output of ScanTailor, as well as the outcome of the upcoming binarization and deskewing sub steps. In fact, ScanTailor is not a true OCR4all submodule since it cannot be integrated due to the lack of a web-based user interface. However, we still decided to list it as a module since this step belongs to the workflow and the input images have to be added from external sources anyway. It is possible to deal with unprepared images like the ones described in the input completely within OCR4all but it certainly is not the recommended course of action. Incorrectly oriented scans generate additional effort during the upcoming deskewing step (cf. Section 5.4.3) and scans with a lot of periphery or containing two pages are harder to display and to work with in the interactive steps like region segmentation using LAREX and manual corrections (cf. Sections 5.5.1.1 and 5.8.2, respectively).

5.4.2 Binarization

Input: original image (color, grayscale, or binary)

Output: binary (and flattened grayscale) image(s)

During the binarization sub step the input image gets converted into a binary and (optionally) a flattened grayscale image. We use the OCRopus 1 *nlbin* script which we explain over the next two sections since it also performs the deskewing operation. The binary/grayscale conversion is performed by applying an adaptive technique proposed by Afzal et al. [6] which is explained in the following and depicted in Figure 5.5.



Figure 5.4: An example input and output of the image preparation and preprocessing steps. The image on the left represents an workable but undesirable input for OCR4all while the ScanTailor output in the middle is completely sufficient. During the preprocessing step the skewed color image in the middle is transformed into the deskewed binary image on the right. Adopted from [229].

Algorithm First the original image is converted to grayscale using a standard operation. After ensuring that it is neither completely black nor white, a normalization step is applied to ensure values between 0 and 1. In the subsequent so-called *flattening* step the local whitelevels are estimated for each pixel by first running a percentage filter with a vertically stretched kernel on the grayscale image and then processing the outcome with a horizontally stretched kernel. In both cases the local background intensity for a pixel is estimated by calculating the 80th percentile value (default) from the filter pixels which works as follows: First, all pixels under the kernel (default size 2x20 or 20x2; applied in an overlapping way) are sorted according to their grayscale values in ascending order. Then, the value representing the 80th percentile, that is the value below which 80% of the grayscale values may be found, is assigned to all pixels covered by the current filter kernel. After inverting the result and adding it to the normalized grayscale image, leading to a suppression of the background, another percentile-based operation is performed: Per default pixels with grayscale values lower than the 5th percentile of all values are set to 0, and pixels with grayscale values higher than the 90th percentile are set to 1. Next, clipping the pixel values to [0, 1] results in the flattened grayscale image which represents the first of the two final outputs of the algorithm. The second one is the binary image that is produced by binarizing the flattened image with a global threshold of 0.5 (default value).

Conclusion The approach described above is able to reliably produce high quality results even when facing difficult conditions, such as heavily degraded scans with considerable brightness variations and bleed through. In addition, it is reasonably fast and due to the local estimation very robust against scan periphery. The high degree of parametrization

theoretically allows for a precise adaptation to the material at hand, however, in our experience many parameters, and consequently the effects when changing them, are rather obscure.

5.4.3 Deskewing

Input: skewed image

Output: deskewed image

While the binarization substep is mandatory to enable and facilitate the upcoming image processing applications, the deskewing on page level is optional since its main purpose is to support the line segmentation process (see below) that operates on segmented regions which are individually deskewed beforehand anyway. Yet, depending on the degree of skewness of the original scans, it can be very beneficial to perform deskewing already on page level as it can considerably simplify the region segmentation.

Algorithm The deskewing algorithm implemented in OCRopus 1's *nlbin* script is a simple, yet effective brute force approach. Its general idea is that the CCs, i.e. mostly letters, within straight text lines form a certain pattern where the top and bottom end of characters and, if present, their ascenders and descenders as well as the inter line spacings are horizontally aligned. Naturally, this is not or at least much less the case for skewed lines. In other words, when looking at the row sums of pixel values in a deskewed image, there will be several sharp peaks where lots of black pixels line up, especially at the top and bottom of characters without any ascenders or descender. Additionally, there will be areas with next to no black pixel, i.e. the white areas between the lines. Apparently, these peaks will be much weaker or even non-existent when the image is skewed. To quantify this observation the variance is utilized since it is expected to be maximized when the image is perfectly straight. Figure 5.6 demonstrates this effect.

In practise the algorithm determines a suitable rotational angle by evaluating several solutions and picking the best one: First, a list of to be tested angles is defined using two parameters: the *maximum angle* and the *step size*. The respective default parameters 2 and 8 indicate that angles between -2 to +2 degrees are evaluated using eight steps per degree, so -2, -1.875, -1.75, ... , 1.875, 2. Then, for each angle in the list, the binary image is rotated by that angle, the algorithm counts the black pixels in each row of the resulting image, and calculates the variance among the collected values. The angle that yielded the highest variance is chosen and the corresponding grayscale and binary images are kept as output.

Conclusion Despite its simplicity the deskewing algorithm is very effective, robust, and reliably produces great results for the vast majority of pages. However, there are several drawbacks: First, due to its brute force nature, the required computational effort can be quite substantial. Second, the to be tested angle range has to be specified. While

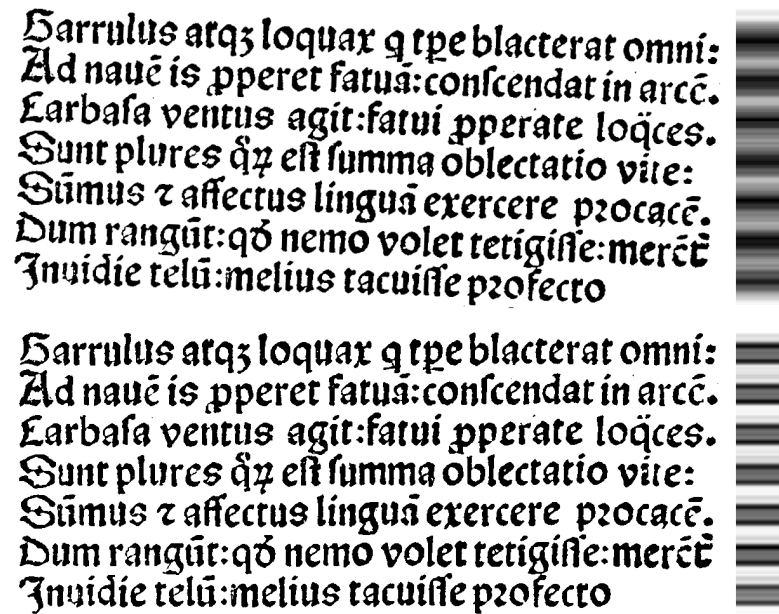


Figure 5.6: A skewed (top) and deskewed (bottom) version of the same region image. On the right the number of black pixels within the corresponding row of the image is visualized: the more black pixels present in a row, the darker the color.

the optimal deskewing angle cannot be found if a too low value is chosen, higher values directly increase the runtime considerably. A possible solution for this problem could be a two-stage approach that first performs a coarse computation to determine an approximate range where the “optimal” angle is located, before a subsequent more fine-grained search refines the result.

5.5 Segmentation

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

5.5.1 Region Segmentation

Input: preprocessed image

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

The general goal of this step is to identify and optionally classify regions in the scan. There are different manifestations which considerably impact the complexity of the task and entirely depend on the material at hand, the use case, and the individual requirements

of the user. For example, when the goal is to OCR a book for editorial purposes it is mandatory to obtain a flawless text and consequently a (close to) flawless segmentation result is advisable. Additionally, in these cases it is often desired to perform a semantic classification of text regions already on layout level. Therefore, a considerable amount of human effort has to be expended. On the contrary, the most simplistic approach would be to only distinguish between text and non-text regions in order to obtain a good ATR result, for example to be used in different Natural Language Processing (NLP) tasks which do not require flawless texts. Naturally, this is a considerably less costly process and can even be a trivial task, depending on the input material. For both scenarios OCR4all offers viable solutions which will be briefly explained in the following.

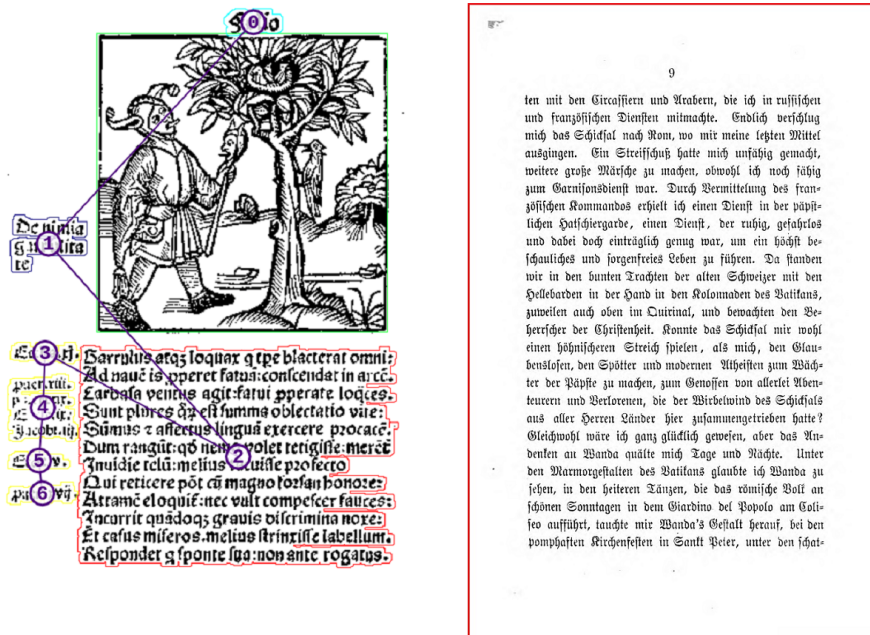


Figure 5.7: Left: a LAREX segmentation output consisting of an image (green), running text (red), marginalia (yellow), image caption (blue), and the page number or folio identifier (cyan), as well as the reading order. Right: output of the Dummy Segmentation for a standard 19th century novel layout. Adopted from [229].

5.5.1.1 LAREX

For complex layouts and especially when a fine-grained semantic distinction is desired (see for example Figure 5.7, left), the open-source [156] tool *LAREX* (Layout Analysis and Region Extraction) [232] is a good choice. It offers the user a variety of automatic, assisted, and fully manual tools which allow to gather a complex page layout with reasonable effort. The segmentation procedure is user-driven and relies on a simple CC approach for the fully automatic part of the workflow. It is based on two main

assumptions: First, it starts from the premise that related characters, words and text lines usually are closer to each other than unrelated ones. Second, it expects most pages within the same book to have similar layouts or at least to belong to one of only a few layout categories which is almost always the case, even when dealing with early printed books.

LAREX is not designed to be able to automatically segment any given document to perfection. In contrast, it aims to provide the user with a quick and easily comprehensible way to adapt to a given layout, get a segmentation suggestion, and then to manually correct it if necessary. Therefore, the correction operations should be as simple as possible and deliver an immediate feedback. The tool is programmed in Java. All required image processing operations were performed by using the OpenCV library [44].

The general workflow LAREX follows to deal with a single page is depicted in Figure 5.8 and explained step by step in the following.

1. Preprocessing: input \rightarrow resized binary ([A] in Figure 5.8).
 - a) Conversion to binary¹.
 - b) Definition of a region of interest (optional).
 - c) Resizing the image.
2. Image detection ([A] \rightarrow [B]).
 - a) Region Growing: many small regions \rightarrow fewer large regions (optional).
 - b) Region Classification: image \leftrightarrow no image.
3. Coarse text region detection.
 - a) Region Growing: many small regions \rightarrow fewer large regions ([C]).
 - b) Region Classification using constraints on attributes of regions: regions \rightarrow region types ([C] \rightarrow [D][E]).
4. Manual amendments and optional text sub region classification ([E] \rightarrow [F]).
5. Conversion to output format.

Preprocessing LAREX accepts color, grayscale, and binary images as input. However, the upcoming region growing and contour detection operations require the binary format. Therefore, a conversion is performed if necessary. Of course, the resulting segmentation still corresponds to the original input image. The user can work on the entire binary image or specify a region of interest by drawing a rectangle, e.g. to exclude artifacts of the input image due to scanning. Everything outside the specified area is ignored and

¹Of course, this step is not necessary if the input is already a binary image but first, LAREX also allows to operate on the flattened grayscale image produced during the binarization step (cf. Section 5.4.2) and second, LAREX can also be used as a stand-alone tool allowing the usage of arbitrary color and grayscale images.

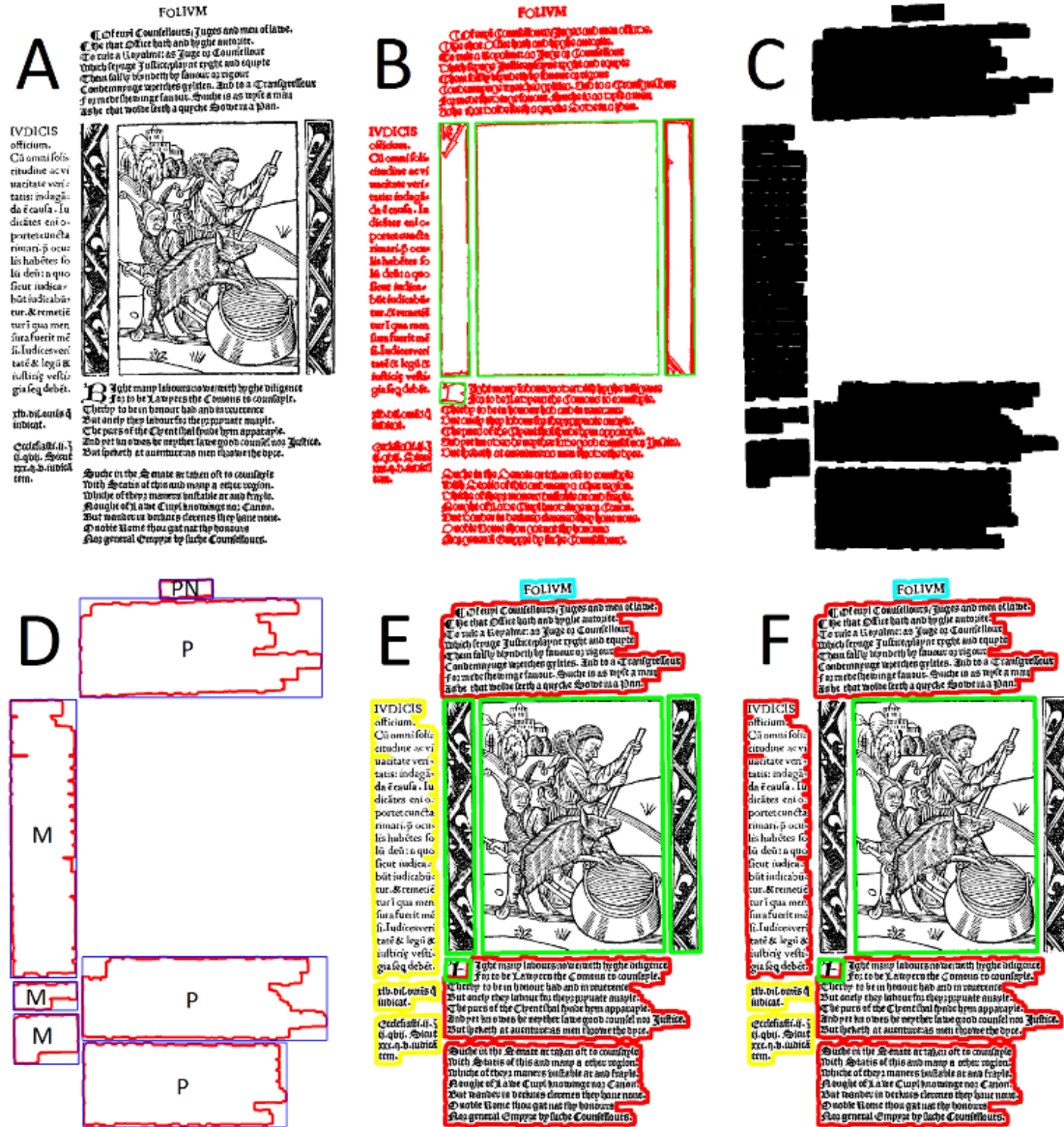


Figure 5.8: Typical LAREX workflow for a single image. A: binary, B: after image detection, C: after image removal and region growing, D: after text region classification (P: paragraph, M: marginalia, PN: page number), E: raw algorithm output, F: after user amendment. Adopted from [232].

will not have any effect on the rest of the segmentation process. A region of interest can either be specified for a single page or be applied for the entire book. It can be adjusted at any time. The image is resized to a given resolution (default: 1,600 pixels in height) in order to speed up the segmentation process and to normalize parameters like the upcoming dilation catchment areas or the required minimum size of the regions.

Image Detection Images are expected to have a certain minimum size and ideally to possess a border. First, a dilation operation is applied using small kernel values to prevent letters and words from merging as well. It may be imagined as a growing process during which black foreground pixels within a defined kernel grow together. All CCs bigger than the given image size threshold (default: 3,000 pixels) are marked as image components which, if desired, can represent the final output of this step. However, in most cases a postprocessing step is sensible: If the bounding rectangles of several image components overlap, they are merged and the resulting (straight or rotated) bounding rectangle is calculated. This step is repeated until no further merges are possible. For further processing either only the contours themselves or the bounding rectangles around them are removed from the image.

Coarse Text Region Detection Afterwards, another dilation process with a bigger kernel than before takes place (step 3a). Then in step 3b, the newly emerged regions are assigned a type by enforcing several constraints. The constraints for type classification use region attributes like size and position. Both can comfortably be specified and adapted by the user via the GUI. The position constraints are visualized as rectangles of different colors, indicated the region type (running text, marginalia, ...) they belong to. For all types it is checked if the area of the region is bigger than the type-specific minimal area. If this is the case and if the bounding rectangle of the region is completely covered by one of the rectangle positions of the type, the region is assigned that type.

Conversion to output format When the user is satisfied, the segmentation result can be stored as an XML file using the PageXML format. Thus, the segmentation result is recalculated to the true size of the input image.

Manual Adaptations In many cases the default parameters will not lead to a satisfying segmentation result. In the following, global and local operations to improve the results are shown.

Figure 5.9 shows an example outcome of the default setup compared to using adjusted parameters as well as the means necessary to derive a mask that precisely describes the individual layout of this individual page but also of almost all other pages within this book. The settings can be saved and applied to the rest of the book as well as to other books with similar layouts resulting in significant speedups with regard to the region segmentation process.



Figure 5.9: Example outcome of the LAREX segmentation when using the default (left) and slightly adjusted (right) parameters. The default setup consists of four regions (image: green, can occur anywhere on the page; paragraph: red, anywhere; marginalia: yellow, left and right 25%; page number: cyan, top and bottom 25%). Naturally, the obtained result is far from optimal and there is still some work to be done. The swash capital was correctly detected as was the page number. However, the marginalia regions got classified as paragraph, as their outer contour was not located entirely within the marginalia regions. Furthermore, there is a signature mark at the very bottom of the page. Lastly, the vertical spacing between text lines is relatively big, leading to the unwanted separation of text blocks. The default setup can be easily adapted by only a few clicks. First, the page number position at the bottom is deleted and the upper one is slightly adjusted, as in this book page numbers always occur at the top center of the page. Furthermore, the marginalia regions have to be expanded towards the middle of the image. The dilation parameters are slightly increased. Finally, another region of the type signature-mark (maroon) is added and located at the bottom center of the page. Adopted from [232].

For special cases the user can make amendments like adjustment of region parameters or the manual adaption of the obtained segmentation result by deleting segments, changing their type and splitting or merging them. For example, in Figure 5.9 the signature mark is connected to the bottom text block which is difficult to correct with global parameter optimization. There are several ways to deal with such problems which will be discussed in-detail during the *Manual Corrections* section (cf. Section 5.8.2). It is worth mentioning, that it is also possible to load existing segmentation results into LAREX and mostly use it as an editor by comfortably correcting the results, if necessary.

Conclusion The primary goal of LAREX is not to fully automatically achieve a decent standardized result but to enable the users to obtain their personal 100% result with manageable effort. This is particularly true for the desired complexity of the segmentation, that is the degree of semantic distinction within the result. Therefore, LAREX relies on a simple, yet effective rule-based approach which is very fast, easily comprehensible, and consequently easily adaptable for basically any given user.

Initially LAREX was developed to support the *Narragonien digital* project (cf. Section 3.1.1.1) where many editions had similar properties the ones shown in Figures 5.7, 5.8, or 5.9, i.e. relatively complex but regular layouts which can be projected to layout masks relatively well. Preliminary evaluations showed that LAREX provides an efficient and flexible way to segment pages of these early printed books: In a case study [232] on an edition of the *Narrenschiiff* with complex layout it took a human processor with some prior experience using LAREX 2 hours and 18 minutes to segment the entire book including a fine-grained semantic distinction of layout elements. For comparison, a processor with extensive experience using Aletheia (cf. Section 2.3.1.4) only managed 160 pages (28% of the entire book) during the same time frame.

Naturally, the relatively straight forward region growing-based segmentation approach has its shortcomings when the regions are not clearly spatially separated. For example, a quite common problem is that the marginalia are directly attached to the main text and the user has to manually interfere to obtain a precise segmentation. However, this can be done very efficiently due to the comprehensive manual tools and due to the fact that the automatic algorithm takes manual corrections into account and adapts the results accordingly, in this example by labeling all cut off regions as marginalia. The main drawback of LAREX is that, at least as of now, it expects the user to have a look at every single page and approve each result individually. Naturally, while checking each page is almost imperative for very complex layouts and high user expectations, it cannot be considered the preferred solution for considerably more moderate or even trivial layouts.

5.5.1.2 Dummy Segmentation

For this other end of the spectrum, OCR4all offers the so-called *Dummy Segmentation* which for the most part simply considers the entire page as a single running text segment (see Figure 5.7, right). Optionally, the user can activate a straight forward image detection

routine that operates analogously to the LAREX approach. While admittedly the Dummy Segmentation represents a highly simplified approach, which initially was only introduced due to technical reasons in order to basically skip the interactive region segmentation using LAREX without having to alter the general workflow, our experiences working with it have been very positive for several reasons: Most importantly, the upcoming line segmentation step actually comprises several additional segmentation capabilities which allow to process more complex inputs than already separated regions. For example, there is a rudimentary implicit text/non-text segmentation available as well as a highly performant column detection functionality. Furthermore, because of the additional capabilities of the line segmentation (cf. Section 5.5.2), the Dummy Segmentation can actually be applied to an unexpectedly wide variety of historical printings and, in fact, runs fully automatically and basically at no cost. On the downside, this approach does not perform any kind of meaningful semantic distinction of text parts.

Of course, we have been experimenting with more sophisticated approaches like pixel classifiers, but the lack of fitting GT, both in terms of quantity, but also quality, led to unsatisfying results, at least when targeting a mixed model as a generic solution. This topic will be addressed in greater detail during the Conclusion in Chapter 8.

Naturally, the user can decide on a page by page basis which segmentation approach to apply. For example, if a book starts with a quite complicated title page and a complex register, both in terms of layout, but apart from that consists of pages with a trivial one column layout, the user can easily segment the first few pages using LAREX and then switch to the dummy segmentation for the remainder of the book. Due to the well defined interfaces all preceding and subsequent steps can be applied to all pages in the exact same manner and without any further differentiations.

5.5.2 Line Segmentation

Input: text regions

Output: extracted text lines

The actual line segmentation operates on individual region images instead of the entire page. Therefore, the text regions identified during the segmentation step need to be extracted from the page images which is done by a region extraction sub step. We cut out the polygons stored in the PageXML file from the corresponding binary image. By extracting the exact polygon instead of just the bounding rectangle it is ensured that even complex alignments of several regions can be processed without any overlap of other regions. After the extraction, the region images are separately deskewed by applying the OCRopus 1 *nbin* script. Processing the regions one by one can lead to considerably better results than the standard deskewing on page level. Especially pages from very old printings frequently contain areas/regions which are skewed independently from each other, either because of inaccuracies during the printing process or due to physical degradation leading to deformed pages. Clearly, a “globally optimal” skewing angle on page level cannot deal with these situations. Since the deskewing operation can

be computationally expensive, the detected angle is stored in the PageXML and reused if the line segmentation is run again, for example with different parameters. Of course, the angle loses its validity and is consequently deleted, if the region outline is changed.

Since first impressions indicated that the performance of the OCRopus 1 approach and the one of OCRopus 3 are very similar, we focus on the traditional method since it is less of a black box approach and offers comprehensive parametrization capabilities. Hence, the actual line segmentation is performed by applying the OCRopus 1 line segmentation algorithm to the extracted and deskewed region images.

Algorithm In the following we describe the line segmentation algorithm implemented in OCRopus 1's *gpageseg* script. Figure 5.10 shows intermediate results from the individual sub steps using an example image.

After inverting the binary image the so-called *scale* value, which basically represents the *x-height*, i.e. the height of characters without ascenders and descenders (cf. Section 2.3.2.1 and Figure 2.1), is determined by calculating the median value of the *y* sizes of all CCs. Next, CCs that are considered unlikely to be characters, because they are too big or too small according to scale, are removed in order to eliminate noise and other unwanted elements like separator lines. Then, the algorithm optionally looks for columns by searching for vertical background separators of a certain length and width. In our example some candidates are found at positions where the inter word distances are relatively big but are then discarded due to insufficient size properties. Just like the rest of the line segmentation algorithm the column detection is highly parameterized and incorporates plenty of heuristics. For all parameters sensible default settings are available that do not have to be altered in the vast majority of cases. The column separation represents an exception to the rule as it requires the number of column separators to look for. For example, a standard two column layout has three separators: left and right border as well as in between the columns. Next, small CCs, mostly punctuation and diacritica, are temporarily removed to perform the next step: The top and bottom borders of the bounding boxes around the remaining CCs are horizontally blurred with some postprocessing including maximum filters returning the top and bottom boundaries of the so-called *seeds*. Expanding these seeds, again based on the scale value, then yields the final labeling by assigning the CCs to their respective text lines based on their overlap. CCs are mostly allocated as a whole but can also be split, if necessary, by again making use of heuristics.

Finally, a smearing algorithm is used to connect the CCs or their respective parts to a tight-fitting polygon in order to produce an optimal line segmentation result, exclusively containing the desired letters.

Conclusion Our adapted version of the OCRopus 1 *gpageseg* script has proven to be able to comfortably deal with the vast majority of line segmentation task. One of the main reasons for this is the fact that the output produced by the applied algorithm is

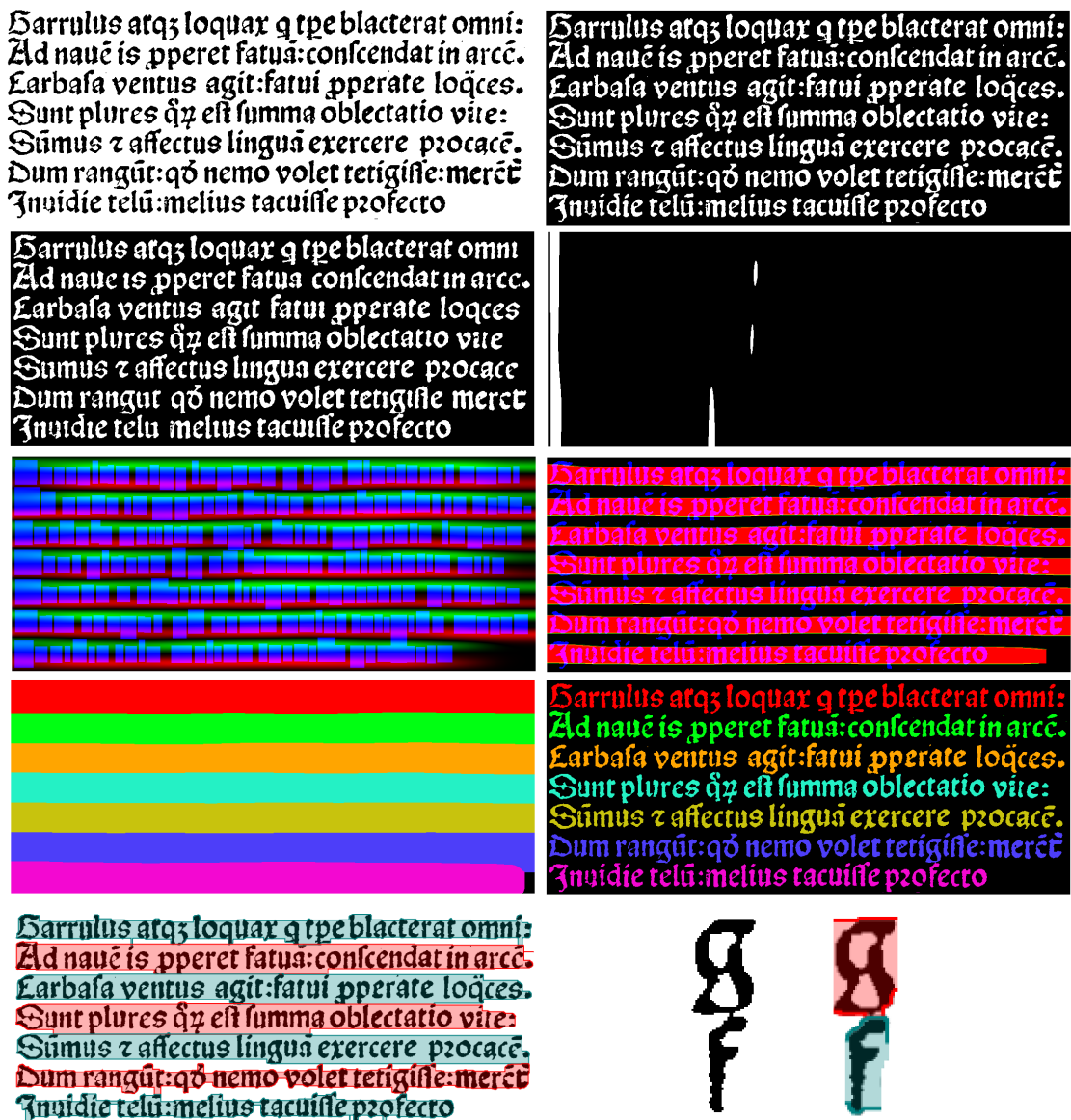


Figure 5.10: An example of the OCRopus 1 *gpageseg* functionality. First row: input binary (left); inverted image (right). Second row: cleaned image (left); column separator candidates (white, right). Third row: blurred tops (green) and bottoms (red) of the remaining CCs (blue, left); generated seeds (pink, right). Fourth row: spreaded seeds (left); resulting labeling of the CCs (right). Fifth row: generated line polygons (left, not part of *gpageseg*); magnification of the input and final output of the two touching glyphs on the bottom right of the text block.

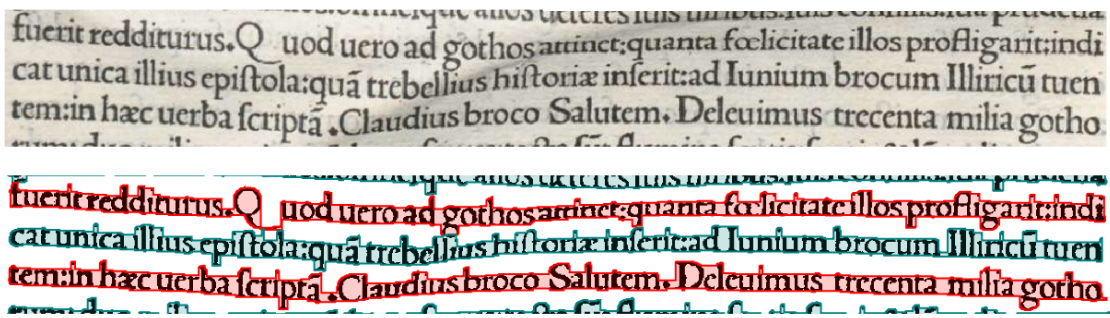


Figure 5.11: Example of the line segmentation step. Top: input image of a very challenging text snippet. Bottom: high quality line segmentation result showing the individual line polygons. For reasons of clarity, altering lines are depicted in different colors. Adopted from [229].

considerably more complex than just the bounding rectangle of a text line: Analogously to the region segmentation step the usage of exact polygons even allows to segment lines whose bounding rectangles considerably overlap. An extreme example of the segmentation capabilities is shown on the bottom of Figure 5.11. The subsequent ATR step had no difficulty with the recognition of each separate line. Other positives include the reasonable speed of the approach, and its basic text/non-text segmentation and column detection capabilities.

A major drawback of the method is its overdependence on and overreliance to the scale value which becomes especially apparent when dealing with pages or text blocks where the font size varies considerably, leading to lines split in half or merged together. An example of a quite common problem is shown in Figure 5.12. The scale value is oriented towards the size of the many smaller running text lines which leads to two seeds being planted within the running head line, resulting in a split. It is important to note that these effects can be heavily influenced in a positive way by either performing a region segmentation first and then run the line segmentation for each region, as it is done in OCR4all, or by manually adjusting the scale parameter.

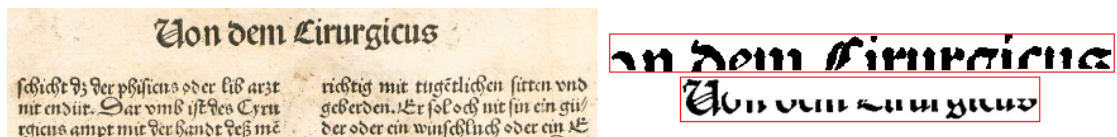


Figure 5.12: Example for a failed line segmentation due to varying font sizes. Left: Top snippet of a page with a large sized running head and several running text lines about half as high. Right: Severely flawed line segmentation result with the running head line mostly cut in half (detected lines indicated by red rectangles).

5.6 Automatic Text Recognition

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

5.6.1 Character Recognition

Input: text line images and one or several ATR models

Output: textual ATR output on line level

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

For the application in OCR4all we make use of Calamari's PageXML interface which cuts out the text lines from the corresponding binary or grayscale images according to their coordinates and passes them into the recognizer. In general, the recognition module allows to either apply self trained book-specific models, which we will address in the next section, or to resort to mixed models. These models have been trained on a wide variety of books and typesets and, depending on the material at hand, can usually provide at least a valid starting point to start off the manual GT production or even already provide a satisfactory final result. OCR4all comes with four single standard models [198] which are automatically incorporated and made available when building the Docker image: *antiqua_modern* (introduced as AM in Section 3.3.2), *antiqua_historical* (AH), *fraktur_historical* (FH), and *fraktur_19th_century* (outcome of the case study described in Section 7.1). Since voting ensembles have proven to be very effective (cf. Section 4.1), we additionally provide a full set of model ensembles [64] consisting of five models for each of the four single model areas mentioned above, which can be downloaded and directly added into OCR4all. As shown in Figure 5.13 the to be applied models can be comfortably selected.

Calamari supports the utilization of an arbitrary number of models. If only one model is applied, its output is directly considered as the final result and is consequently added to the corresponding line element in the PageXML file. The application of several models automatically triggers the confidence voting procedure where the final result is calculated from the single outputs of all the voters. Apart from the standard textual output it is also possible to enable an additional, extended output that includes information like the intrinsic ATR confidence values calculated by Calamari or the pixel positions of the detected characters within the line. Parts of this data could also be stored by using PageXML but since we want to keep as much information as possible, including detailed

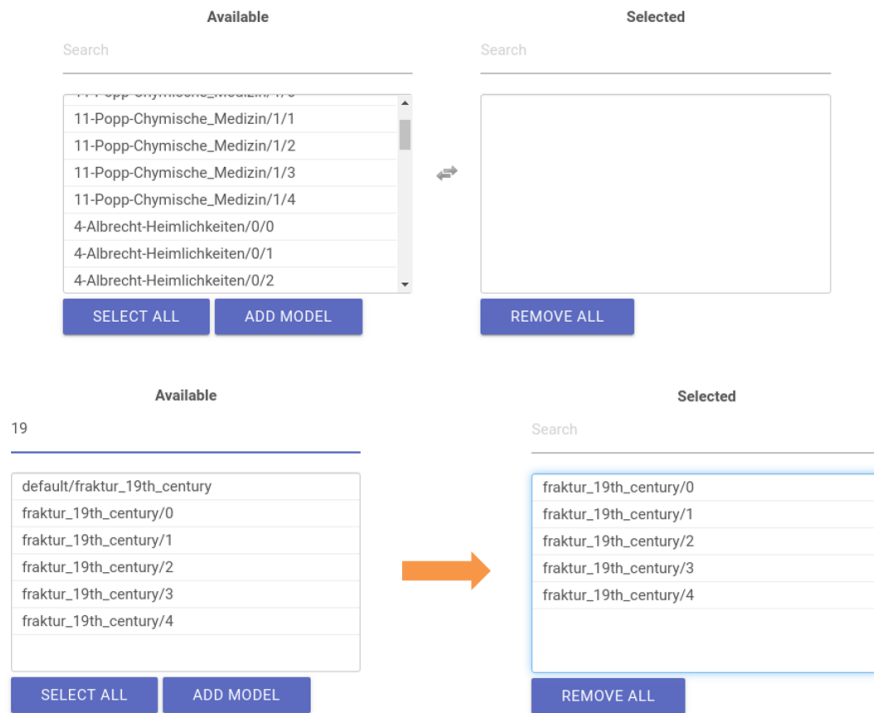


Figure 5.13: Example model selection process in OCR4all with the goal to select the 19th century Fraktur model ensemble for recognition. Top: Initial situation with a list of all available mixed and book-specific models (left) and currently no models selected for recognition (right). Bottom: Filtering result showing only models containing the search term “19” (left). Final model selection (right).

and comprehensive lists with character alternatives and their respective confidences, an additional storage format (JSON [45]) is required.

After transcribing lines or by correcting an existing ATR output, which we will cover in detail in the section on manual corrections below, the model training can take place.

5.6.2 Model Training

Input: line images with corresponding GT, optionally already existing models to build upon and a corresponding Whitelist (cf. Section 4.2.2.6)

Output: one or several ATR models

The model training which allows to train book-specific Calamari models is not only one of the most central modules of the entire workflow but also probably the most complex and challenging one when it comes to enabling non-technical users to utilize all available features. The training algorithm possesses a variety of hyper-parameters which can impact the procedure considerably [57] and therefore has to be treated with great care.

To begin with, one crucial aspect when dealing with the training of neural networks is the omnipresent challenge of determining when to stop the training process and choosing the best model, which is the one with the “optimal” weight configuration. To avoid overfitting we follow the established approach of setting aside a small chunk of the training material as a test set (cf. Section 4.1.2.3) which is realized by passing appropriate parameter settings into Calamari. Then, after a certain number of training iterations the current model is applied to and evaluated on this test set. The model which performed best is denoted as the best model and stored for further processing. Regarding the termination of the training we make use of the so-called *early stopping* provided by Calamari that basically observes the training progress and aborts as soon as no significant improvement can be expected anymore. This procedure requires several parameters which determine the stopping criterion and how frequently to evaluate the current model. To guard the users from having to deal with the underlying theoretical concept but still ensure sensible parameters, we developed a routine, that derives fitting values for all required settings from the available amount of training data, and was incorporated into Calamari. Naturally, experienced users are free to adjust the parameters directly at will.

Our main goal for the training module was to enable a non-technical user to comfortably make use of accuracy improving techniques like cross fold training (cf. Section 4.1), pretraining (Section 4.2), and data augmentation [203] (see Figure 5.14 for the implementation within the OCR4all web GUI):

- *Cross Fold Training*: A sensible arrangement of the available data into training and validation sets, in order to obtain diverse voters with different strengths and weaknesses, is carried out automatically by Calamari. All the user has to do is to determine the number of desired voters.
- *Pretraining*: Naturally, the usage of already existing models can and should be combined with the cross fold training. Consequently, the user can choose between three training approaches: training all models from scratch, training all models starting from the same existing models, or freely assigning arbitrary models to each fold. All this can be done by comfortably selecting the desired models from a list displaying all available options.
- *Data Augmentation*: To activate the generation of synthetically altered additional training data in OCR4all, the user simply has to determine the extent of desired augmentations, i.e. the number of augmented lines to be derived from each real line. Since nothing compares to real data the default training allows for a two step approach in which the models resulting from training on both real and augmented data are further refined by training exclusively on real data.

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

1. Transcribe a small number of lines from scratch or correct the output of a suitable mixed model, if available.
2. Train a book-specific model/voting ensemble using all available GT that has been transcribed up to this point, including earlier iterations.
3. Apply the model/voting ensemble to further lines.
4. Correct the output.
5. Repeat steps 2-4.

For example, let's assume that around 200 lines of GT are needed to reach a satisfactory ATR accuracy (e.g. a CER of 2% or less) for a given task and on a given book. Of course, the exact number of lines required is never known, which is another reason to not simply transcribe 200 lines from scratch before training the first model. Instead, we start out by recognizing a small number of lines, for example two pages comprising 60 lines, with a somewhat suitable available mixed model, resulting in a more or less helpful ATR output with a CER of, let's say 8%. While the recognition quality is not perfect, correcting this output is still faster than transcribing from scratch and it only needs to be done for 60 lines anyways. Next, a first book-specific model is trained by using the produced GT. The resulting model is then applied to four more page, resulting in a considerably lower CER of 3.5%. In fact, correcting this output can be done much more efficiently than before since the error rate was reduced dramatically by more than 50%. The resulting four pages of GT are added to the GT pool (now containing close to 200 lines) and used for another training process, resulting in a strong model yielding a CER of just below 2% on previously unseen pages. Since this final model fulfills the requirements set for this example, the iterative training process stops here. Naturally, the described steps can easily be repeated until a higher desired accuracy is reached or even until the entire book has been recognized and corrected.

Since the iterative training approach, especially when combined with cross fold training, can quickly produce plenty of ATR models, a certain amount of bookkeeping is required to stay on top of things. Therefore, OCR4all provides an intuitive automatic naming convention for the trained models. First, a separate folder is created in the models folder for each book. Then, each training iteration produces a folder in the books folder numbered incrementally according to the current iteration, starting with 0, 1, and so on. The resulting voting ensembles are then stored in the corresponding folders by simply

labeling the single models 0, 1, ..., allowing each ensemble to comfortably get selected by filtering for the desired book and iteration.

5.6.3 Error Analysis

Input: line-based ATR predictions and the corresponding GT

Output: CER and confusion statistics

To enable an objective assessment of the recognition quality achieved by the models at hand, we incorporated the Calamari evaluation script into OCR4all. For a given selection of pages it compares the ATR results to the corresponding GT and calculates the CER using the Levenshtein distance. Additionally, a confusion table (see Figure 5.1 for an example) displaying the most common ATR errors and their frequency of occurrence is provided.

Table 5.1: Example confusion table showing the desired output (*GT*), the actual output (*ATR*), the absolute number of occurrences (*CNT*), and the corresponding percentage with respect to all errors (*PERC*). Given are the five most frequent errors including substitutions (4), deletions (2,5), and insertions (1,3).

ID	GT	ATR	CNT	PERC
1		□	16	6.69
2	□		10	4.18
3		i	10	4.18
4	e	c	3	1.26
5	l		2	0.84

5.7 Result Generation

Input: GT and ATR results

Output: final output as text files

For the average OCR4all user PageXML most likely does not represent the desired output format that is needed for further processing with other tools. Consequently, we also offer a simple textual output where the line-based ATR results are concatenated in reading order and stored as a text file in two variants, one for each individual page and one for the entire book. If there is GT available for a line its ATR result is replaced by the corrected text in the final output.

Naturally, the conversion into raw text leads to the loss of all acquired additional information obtained during the complete workflow like semantic labels and coordinates of segments and lines. To preserve this data it is of course also possible to keep the PageXML files containing all information acquired during the workflow. Additional

output formats, for example TEI [292], can be added which will be discussed in greater detail in Section 8.2.

5.8 Web Application

One of the main goals of OCR4all is to allow anyone to perform OCR on their own on a wide variety of historical printings and obtain high quality results with reasonable time expenditure. Therefore, the tool has to be easily comprehensible even for users with no technical background. In fact, this includes the ability to comfortably control the entire process via a GUI. By making all submodules accessible from clearly structured and unified interfaces it ensures that the user for the most part has to learn only a single system. Furthermore, OCR4all equips the user with powerful tools to perform manual corrections on the produced output after most steps of the workflow and allows for a precise configuration, not only of the workflow in total but also of the sub modules. Before we discuss these options we briefly introduce some general aspects about the tool's architecture.

5.8.1 General Software Design

We chose to implement the workflow as a server application accessible by a web app because this allows a deployment as a true web app (using a web browser as a local client interacting with a remote server) as well as using OCR4all completely locally (both browser and server are locally installed). Because currently neither user administration nor resource management have been implemented and some steps in the workflow require a considerable amount of computational power, we only consider the local option in the following.

The core OCR4all web application is programmed in Java [110] and relies on the Java EE [136] compliant framework Spring [137]. It is then deployed on an Apache Tomcat [300] server. The user interface implements the Materialize CSS framework [182] to deliver a feature rich user experience. While most of the image processing tasks are delegated to the respective sub modules, the core application also requires some functionality and thereto uses the OpenCV library [44].

We rely on Docker [183] and VirtualBox [311] since both solutions ensure an easy installation process and keep the problems caused by dependency requirements of the sub modules to a minimum. Furthermore, the incorporation of Docker and VirtualBox effectively assures platform independency as they can be installed and run on basically all modern operating systems including Windows, Mac, and Linux.

5.8.2 Manual Corrections

Input: page images and their corresponding PageXML files

Output: corrected PageXML files

As emphasized before, a fully automated workflow is often not reasonable or at least cannot be expected to yield sufficient (depending on the use case) or even perfect results especially when dealing with early printings. Consequently, a potent, flexible, comprehensible, and easy to use option for manual correction is a must-have for every OCR workflow tool which relies on user intervention. In OCR4all this core task is covered by LAREX, initially introduced as a region segmentation tool in Section 5.5.1.1 and in its original release [232]. However, its functionality has been considerably extended since and will be explained in the following (see Figure 5.15 for an overview of the LAREX correction GUI).

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

5.8.2.1 Regions

LAREX offers a wide variety of tools and procedures to create new and edit existing regions. Regions identified during the earlier region segmentation step can be deleted and their type or sub type can be changed. If a region has not been found, the user can correct this by either manually drawing a rectangle or a polygon or by selecting the CCs belonging to the region and then activate an iterative smearing algorithm to automatically create the region outline. Additionally, the reading order can be freely adjusted by dragging and dropping regions in an additional view. Furthermore, it is possible to perform sophisticated polygon manipulation operations including deleting, adding, and moving points. Each operation can always be performed on an arbitrary number of points at once. Combined with the progressive zooming functionality this even allows for an pixel perfect segmentation, if desired. Figure 5.16 shows some example functionality.

5.8.2.2 Lines

The second view focuses on text lines. From an editing point of view lines are treated exactly like regions and therefore allow the same comprehensive set of operations with minor adaptations, for example that newly created lines are assigned to the active regions and not to the page and the reading order functionality is available for lines within a selected region.

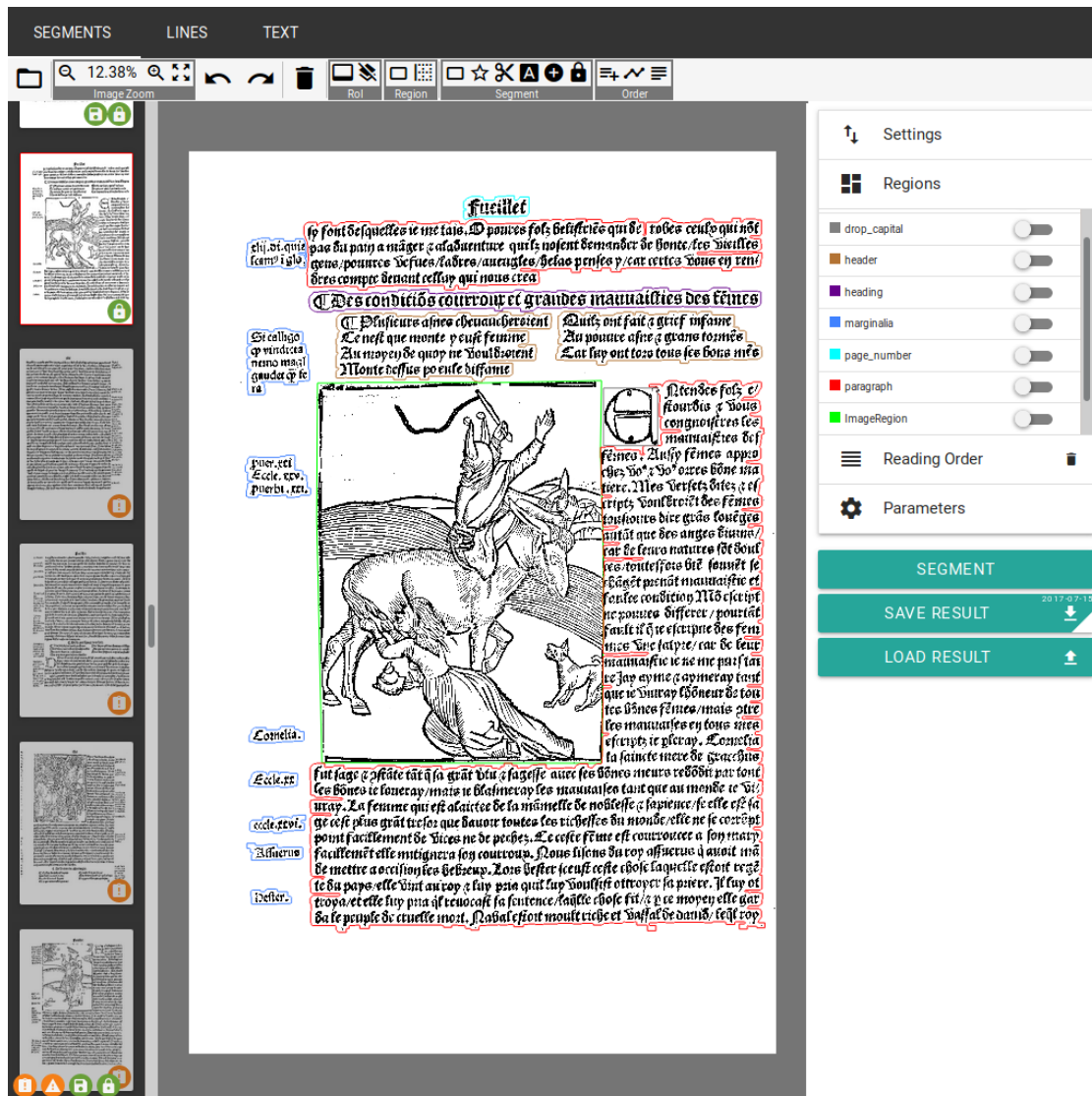


Figure 5.15: Compressed overview over the LAREX correction GUI with the actual page and its current data in the middle, the page selector on the left, the three correction tabs to switch between the segment, line, and text correction functionality as well as several tools on the top and the settings on the right.

5.8.2.3 Text

The text view (see Figure 5.17) is divided into two further sub views. In the first one the page image is still presented to the user with all text lines color-coded indicating the availability of corresponding GT. When selecting a text line, an input field is displayed directly below the line, showing the corresponding ATR result or GT if available. Like the

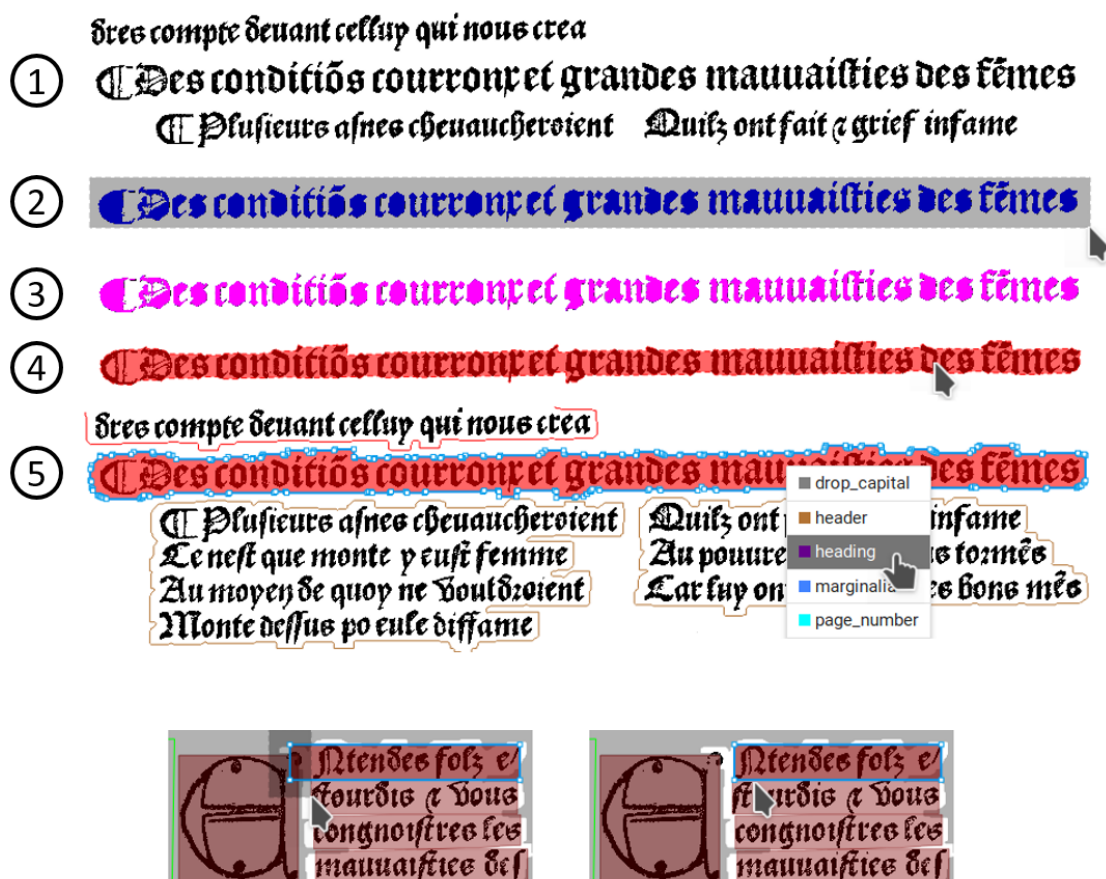


Figure 5.16: Example applications of the labelling and segmentation functionality on layout level in LAREX. Top: Semi-automatic segmentation of a complex layout (from top to bottom): (1) excerpt from the starting situation showing four lines belonging to four different segments with three different semantic types (see the top of Figure 5.17 for more context). (2) Manual selection of the CCs belonging to the subheading. (3) Marked depiction of the selected CCs which serve as input for the iterative smearing algorithm. (4) Output region of the smearing algorithm. (5) Adjustment of the region type; adjacent the remaining regions which have been automatically detected (remaining lines). Bottom: correction of a line polygon which wrongfully includes parts of a swash capital by moving two points.

rest of the visualization the displayed text line is zoomable and can be moved horizontally or resized separately to obtain a perfect alignment with the line image, since this cannot always be achieved automatically due to the narrow and often irregular typesetting. The user can then produce or edit GT by simply typing into the input field or by selecting characters from a customizable virtual keyboard which allows to define a set of non-standard characters, for example ligatures, which cannot be found on regular keyboards

but can easily be inserted this way by a single mouse click. The content and structure of the virtual keyboard can be customized directly in the web GUI and existing setups can be exported and imported. Keyboard shortcuts allow to temporarily fade out the input field so the users can get contextual information from the subsequent line if desired and they may quickly cycle through the lines in reading order.

While this view is a suitable solution for users that aim for a perfect text and consequently have to take a thorough look at each line anyway, it is not optimal for use cases where users just want to quickly scan the pages and lines for obvious mistakes. For this use case we introduced a second sub view which optimizes the correction process by providing a line-based view where an editable text field is placed directly under each line image. If there is already GT available for a line the corresponding text field is colored in green and the GT text is displayed. Analogously, if there is no GT but an ATR result available, it is shown in the text field. Otherwise, the transcription has to be performed from scratch. When a line is selected by clicking into the corresponding text field it is marked as active and can again comfortably be edited by typing regularly or via the virtual keyboard. When a line gets deselected, for example by activating the next line, its current content is automatically saved as GT.

Since all pairs are ordered one below the other in reading order it is possible to display the line image and the corresponding transcription for all lines at once allowing the user to get a quick overview. Despite the completely different arrangement of the lines the interconnection with the other views still exists. So if the users noticed a line which suffers from a serious segmentation fault they can simply switch to the line-based view, quickly identify the line since it is still marked as active, perform the necessary correction, and switch back to the text correction view to continue from where they left off.

5.8.2.4 Practical Integration into the Workflow

In theory, manual correction phases can of course be introduced at basically any step during the workflow. While it is clear that ensuring optimal results after each processing step minimizes the chance for and the effect of consequential errors, a comprehensive manual inspection and correction after each step is neither required nor sensible. For example, checking and correcting thousands of text lines one by one after the line segmentation step may well increase the achievable ATR accuracy during the subsequent recognition step. However, a much more efficient solution is to subject a few representative pages to a quick visual check and if any systematic errors are recognized, one may use the comprehensive set of parameters to optimize the output on a global level. In our experience with the currently available setup manual correction should only be applied directly after the region segmentation, which takes place in LAREX anyway if not performed fully automatically, or at the very end when all information including line coordinates and ATR results is available.

Fueillet


ly font desquelles ie me tais. **D** poures folz belistriés qui de robes ceulz qui n'ont pas du pain a māger & a l'adventure quilz nosent demander de honte/les vieilles gens/poures desues/labres/auengles/helas pense y/car certes vous en rendres compte deuant celluy qui nous crea

Des conditiōs courrony et grandes mauuaities des fēmes

**Plusieurs asnes cheuaucheroient
Le nest que monte y eust femme
Au moyen de quoy ne voudroient
Monte dessus po eule diffame**

**Quilz ont fait & grief infame
Au pouure asne & grans tormēs
Car luy ont toz tous ses bons mēs
Car luy ont toz tous ses bons mēs**

**Entendes folz e/
stourdis & vous
congnoistres les
mauuaities des
fēmes. Aussi fēmes appro
chez vo' & vo' ores bone ma
tiere. Mes versetz sitez & es
criptz voul broiet des fēmes**



*elij. di. que
scam' i glo.*

*Si colligo
q' vindicta
nemo magis
gaudet q' te
ra*

*puer. ecci.
Eccle. xxxv.
puerbi. xxi.*

Quilz ont fait & grief infame

Ouilz ont fait 7 grief infame

Au pouure asne & grans tormēs

Au pouure asne 7 grans tormēs

Car luy ont toz tous ses bons mēs

Car luy ont toz tous ses bons mēs

E
C

Entendes folz e/

Entendes folz e=

LOAD ↑ SAVE ↓

ā	æ	þ	ð	ç	ç	d	ð	
ē	ē	ff	ff	fl	g	g	h	f
ī	ī	ll	m	n	o	œ		
p	p	p	p	p	p	p		
q	q	q	q	q	q			
q̄	q̄	q̄	q̄	q̄	q̄	q̄		
r	r	r	s	f	ff	ff	ff	f
t	t	ū	°	°	đ			
v	v	v	w	x	y	z	&	γ
o	o	o	o	o	o	o	o	o

+ ☒ 🔒

Figure 5.17: Two text correction views in LAREX. Top: page based view where a selection of lines can be corrected. Bottom: corresponding line-based view (left) and virtual keyboard (left, available in both views). Adopted from [229].

5.8.3 Configurations

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

5.8.3.1 Process Flow

Most of the steps and modules introduced above give the user the chance to manually check the results and apply corrections, if necessary. However, there are certainly plenty of use cases where this is neither desired nor necessary, for example when dealing with relatively simple layouts. In order to enable the user to minimize the degree of manual intervention, we introduced the process flow functionality which allows to configure and execute several modules at once. For example, when dealing with a typical 19th century Fraktur novel as shown on the right in Figure 5.7, it may well be sufficient to fully automatically run the preprocessing, region segmentation (dummy), region extraction, line segmentation, and recognition (standard 19th century Fraktur model) steps, to obtain a high quality ATR result. Analogously, when a segmentation using LAREX is needed, which requires user intervention, the subsequent steps can still be run at once and fully automatically.

5.8.3.2 Settings

The open-source OCR tools we utilize in our workflow often allow influencing the results by passing parameters, usually via command line options. Optimizing the usage of these tools represented one of the bigger challenges during the implementation of OCR4all. On the one hand, it is imperative not to overwhelm inexperienced users by confronting them with a plethora of confusing options, settings, and parameters. On the other hand, it is equally vital to provide more experienced users to adapt selected settings in order to optimize the results. As a solution, our web GUI provides interfaces to set almost all parameters of the OCRopus 1 and Calamari submodules. Furthermore, we carefully split the available options for each submodule into *general* and *advanced* settings. Apart from the number of threads used for execution, which are by default set to the available maximum, the general settings usually only contain one or two parameters whose default settings normally completely suffice for the average user. The advanced settings comprise all remaining parameters and allow experienced users to maintain full control. Figure 5.18 shows an example.

EXECUTE >
CANCEL ✕

⚙️ Settings (General)

Maximum # whitespace column separators <small style="color: red;">Should be set to '-1' if no column separation is desired/required.</small>	<small>Default: -1</small> -1
Number of parallel threads for program execution	<small>Default: 1 Current: Available threats (Int value)</small> 32

⚙️ Settings (Advanced)

Error checking

Limits

Minimum scale permitted	<small>Default: 12</small>
Maximum # lines permitted	<small>Default: 300</small>

Scale parameters

Line parameters

Column parameters

Whitespace column separators

Output parameters

Other parameters

Figure 5.18: Overview over the settings and parameters view using the line segmentation step that relies on the corresponding OCRopus 1 script (cf. Section 5.5.2) as an example. The default view only displays indispensable parameters to the user, like the number of CPUs to use and, in this case, the number of whitespace separators to look for in order to control the column segmentation. If desired, the advanced settings can be expanded to take full control over each aspect of the respective step.

6 Evaluations

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

The main goals of our experiments are to evaluate the

- manual effort required to OCR a book using a precise segmentation and aiming for a very low error rate ($\leq 1\%$ CER) dependent on the complexity of the material and the experience of the user
- speedup when incorporating the iterative training approach
- potential speedup when considerably lowering the requirements regarding segmentation, especially considering the fine-grained semantic distinction of layout elements
- performance of OCR4all when applied to newer works with simpler layouts

6.1 Data

This section briefly introduces the books we used for our experiments comprising a variety of early printed books and 19th century Fraktur novels.

6.1.1 Early Printed Books

The first part of our evaluation corpus comprises all works introduced in Section 3.1.1 resulting in a selection of 25 books printed before 1600 (cf. Table 3.2 for an overview) including five editions of the *Narrenschiff* (used languages: Latin, German, French, and Dutch), 17 works related to the influential early modern universal scholar Joachim Camerarius the Elder (Latin and Greek), where we focused on the ATR of the Latin parts and just marked Greek text (embedded parts of Greek, mostly scientific technical terms) with a placeholder for later processing, and three further German early modern

printings worked on during a practical course. In the following, we refer to the individual books with a shortcut (N, C, or P) combined with the year of publication, for example C1566 for a Camerarius book printed in 1566. Figure 6.1 shows representative example images of some of the books used, as well as some desired segmentations. For reasons of clarity, we refrained from depicting the reading order.

6.1.2 Fully Automated Processing of 19th Century German Novels printed in Fraktur

The second part of our evaluation corpus consists of 19th Century German novels (with one exception from the late 18th century) as introduced in Section 3.1.2. The overall quality of the material varies considerably as shown in Figure 3.4. The less complex layout, the more regular typography, and the desired use for quantitative experiments make it neither necessary, nor feasible to invest an extensive amount of manual work. A highly automated workflow is intended instead.

6.2 Precise Segmentation and Trained ATR of Early Printed Books

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

6.2.1 General Processing Approach

Each book is always processed by a single user with the exception of 1494 which was divided into two halves and assigned to two users to be handled independently. Clear guidelines for the processing of any book had been specified beforehand. The most important ones can be seen in the following:

- The entire book is segmented by the user and the required time is recorded.
- A fine-grained semantic classification of layout elements level is required, including the distinction of images, running text, headings, page numbers, marginalia, signature marks, catchwords, and swash capitals.
- After segmenting down to line level the GT production and iterative training approach starts. For evaluation purposes only the transcription of whole pages was viable.
- To get notable improvements during the iterative training approach the amount of added GT should rise considerably during each step. The suggested approach was to start out with two to four pages, then add three to five pages during the next iteration and so on.
- Ideally, representative pages with respect to their state of preservation, print quality, and used fonts should be selected for training and evaluation. Especially for the Camerarius project the font aspect is particularly important, since many books comprise large sections printed in italics while the bulk of the text is printed in an upright font.
- Since most of the books are meant to be transcribed in full later on in their respective projects, a very comprehensive training was performed in most cases with the iterative training process stopping only when a CER of 1% or below was reached.

6.2.2 Overall Time Expenditure and ATR Accuracy

In this first experiment we evaluate the two main criteria for a workflow with considerable human interaction: the time that had to be invested both for obtaining a sufficient result

regarding segmentation and ATR as well as the achieved ATR accuracy. These criteria are heavily influenced by several factors which have to be taken into consideration:

First, the experience of the user: An experienced user can be expected to be more efficient, both during the segmentation and the GT production phase. In our experiments we differentiated between two groups of users: On the one hand, there were several first time users with no experience with OCR4all and no or next to no experience with other OCR-related tools and processes. On the other hand, we had users with a solid general OCR background and an extensive history of using OCR4all, LAREX, and various transcription tools. In the following we assign the labels 1 and 2 to the users of the respective groups, with a higher number indicating a more experienced user. To distinguish the individual users from each group we assign additional labels (A, B, ...). With the exception of one experienced user (Digital Humanities) all users are classical humanities scholars. Before starting to work on their respective books on their own, all participants were introduced to the tool by one of the experienced users.

Second, challenges due to the book: Different books can vary considerably, mainly regarding the number of pages, the complexity of the layout, but also the print or scan quality and overall state of preservation. Since the books utilized during our experiments did not show large discrepancies concerning the latter criteria we just provide the number of pages and distinct semantic layout classes. Table 6.1 sums up the results.

6.2.2.1 Results

For the less experienced users an average segmentation expense of slightly more than one minute per page was recorded with considerable variations among different users. Fortunately, more experienced users can speed up the process considerably, resulting in about 36 seconds per page on average again with considerable variations depending on the layout complexity of the book. Regarding the time expenditure required for correcting ATR results for GT production, the vast majority of users invest less than ten seconds per line on average. Both user groups achieved almost identical CERs (0.47% and 0.49%) by utilizing a very similar amount of GT (988 and 927 lines). These results enable us to compare the time expenditure of the users on a more general level by taking the achieved ATR quality out of the equation. Calculating the time required to process a book, both segmenting it and creating enough GT to obtain an average CER of below 0.5% resulted in just 0.7 minutes per page for the experienced users. Compared to the 2.3 minutes achieved by the unexperienced users this represents a speedup of more than factor 3.

6.2.2.2 Interpretations

The times accounted for segmentation clearly show that performing a precise and fine-grained semantic segmentation of early printed books, even when using a comfortable and versatile tool like LAREX, can still amount to several hours of work for a single book. On top comes the time to generate GT (either from scratch or by correcting an

6 Evaluations

Table 6.1: Results of the precise segmentation and trained ATR of early printed books. The *Books* are grouped by the experience of the processing *Users*, first the unexperienced (1), then the experienced ones (2). Regarding the *Segmentation* we provide the number of pages ($\#P$) and semantic region types ($\#R$) that had to be distinguished as well as the time required for the entire book ($t_{Seg.}$) and per page ($t_{Seg./P}$) on average. For the ATR we indicate the maximum number of GT lines $\#L$ which was used to train the final ATR model along with the achieved CER. Furthermore, the time required to correct all lines required to train the final model and one line on average is shown ($t_{Corr.}$). Finally, some *Key Figures* are derived to ensure comparability of the works. The overall manual time expenditure is calculated for the entire book (t_{All}) by adding up the overall time required for segmentation and ATR and for an average single page (t_{All}/P) by dividing by the number of pages. For both user groups averaged values of all books (*Mean*) and the corresponding standard deviation (*StdDev.*) are provided if sensible.

Book Short	User Exp.	Segmentation				ATR				Key Figures	
		#P	#R	$t_{Seg.}$ [min]	$t_{Seg./P}$ [min]	#L	CER [%]	$t_{Corr.}$ [min]	$t_{Corr./L}$ [s]	t_{All} [min]	t_{All}/P [min]
C1532a	1A	55	7	90	1.6	829	0.47	280	20	370	6.7
C1532b	1A	130	7	110	0.8	611	0.73	146	14	256	2.0
C1533	1A	57	5	82	1.4	806	0.20	129	10	211	3.7
C1535	1A	96	7	104	1.1	723	0.39	176	15	280	2.9
C1552	1A	180	6	110	0.6	384	0.20	44	7	154	0.9
C1554	1B	81	6	66	0.8	487	0.36	76	9	142	1.8
C1557	1B	168	5	194	1.2	1,342	0.34	187	8	381	2.3
C1558	1A	94	8	139	1.5	751	0.25	183	15	322	3.4
C1561	1B	344	5	275	0.8	395	0.40	48	7	323	0.9
C1563	1C	158	5	140	0.9	1,175	0.60	95	5	235	1.5
C1566	1D	471	7	370	0.8	596	0.61	48	5	418	0.9
C1568	1B	342	5	223	0.7	406	0.24	36	5	259	0.8
N1494	1E	156	7	210	1.3	2,302	0.69	315	8	525	3.4
N1494	1F	157	7	360	2.3	969	0.82	97	6	457	2.9
N1549	1G	328	7	210	0.6	2,824	0.45	155	3	365	1.1
P1474	1H	198	4	29	0.1	700	0.90	230	20	259	1.3
P1509	1I	218	5	390	1.8	1,501	0.42	310	12	700	3.2
Mean		190	6.1		1.1	988	0.47		10		2.3
StdDev.					0.5		0.22		5.2		1.5
C1541	2B	439	8	345	0.8	847	0.92	82	6	447	1.0
C1566	2A	240	7	80	0.3	599	0.57	45	4	118	0.5
C1583	2A	606	7	200	0.3	1,647	1.00	123	5	323	0.5
C1594	2A	420	8	200	0.5	352	0.50	26	4	226	0.5
C1598	2B	344	8	245	0.7	256	0.45	28	7	273	0.8
N1498	2A	161	6	130	0.8	622	0.30	22	2	152	0.9
N1499	2A	166	7	105	0.6	632	0.12	110	10	215	1.3
N1506	2A	215	8	180	0.8	3,161	0.20	-	-	-	-
P1484	2B	372	3	65	0.2	226	0.34	22	6	80	0.2
Mean		329	6.9		0.6	927	0.49		5.5		0.7
StdDev.					0.2		0.30		2.4		0.4

6.2 Precise Segmentation and Trained ATR of Early Printed Books

ATR result) and to train an ATR model. Our experiments show that the total processing time from beginning (image processing, page segmentation) to end (ATR result with less than 0.5% CER on average) is less than a day for books containing a few hundred pages.

While this is still a far cry from an expectation of “press a button, wait a few seconds, receive the results”, a meaningful comparison would be to look at the current practice of manual transcription as a baseline. This is usually done either by a person sitting in front of two displays, one of which shows the page image of a book and the other a word processor. The scholar alternately looks at the image, tries to decipher its text and enters it into the word processor on the other display. The process is cumbersome, error prone (hence the practice of double keyboarding with two teams entering the same texts which will later be compared to spot transcription errors), and very time consuming. The transcription of a whole book of several hundred pages can easily consume a few weeks. We did not thoroughly evaluate the manual transcription from scratch but to get a rough impression, the users 2A and 2B transcribed a small number of pages from their respective books (C1541, P1484, and 1499). Extrapolating the effort for the entire book led to an overall time expenditure of 44 hours for C1541, 47 hours for P1484, and 130 hours for N1499. Our method therefore reduces the working time from a few weeks to a day, plus the additional effort to weed out the remaining ATR errors.

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

Extremely complex layouts like the ones of N1498, N1499, and especially N1506 can be very challenging and not trivial to process even for very experienced users. Having said that, in our experience these three examples are about as complex as it gets for early printed books, especially combined with our very strict and detailed segmentation guidelines. Almost on the other end of the spectrum are books like P1484 where most pages are almost trivial to segment and therefore only take a minimal amount of time (around 10 seconds when processed by an experienced user).

Regarding the correction of the ATR result it is noteworthy, that four out of six books which required more than ten seconds per line, were processed by a single user, the same that also achieved most of the slow segmentation results (1A). Since there are no obvious reasons for this effect regarding the material we assume that some users simply require more time during the correction process maybe because they are too frightened to miss something. This is also reflected by the general correction strategy of the two groups: While the experienced users tend to simply scan the results by hopping between the line image and the ATR result on a word to word basis, the unexperienced users often first read the entire text in the line image and the ATR result separately, before performing a third check where smaller junks of the line are compared. It is worth mentioning that cross checks of the produced GT showed no noteworthy effects regarding the quality of the transcriptions among different users.

Not only because of the fact that it was the most experienced user (2A) who achieved the worst ATR results of all books, we have no reason to believe that the user has a noteworthy influence on the ATR accuracy. Most importantly, the reachable CER depends on the book and the contained typography as well as the amount of GT used for training. The obtained results underline this assumption almost perfectly.

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

6.2.3 Evaluating the Iterative Training Approach

The manual correction effort not only scales with the number of lines that have to be corrected but also with their recognition quality. To be able to thoroughly evaluate the effects and benefits of the iterative training, many different values and results were recorded. Since their evaluation and interpretation is a quite complex task we first introduce and describe them in detail in Figure 6.2 before we list the results of selected works in Table 6.2.

6.2.3.1 Results

There are several interesting things to be taken away from the results summarized in Table 6.2. First of all, it is shown that the iterative training approach yields a significant speedup regarding the correction time. On average the manual effort is almost cut in half (average speedup factor 1.9, last column) with the experienced users benefitting considerably more compared to the unexperienced ones (factor 2.3 and 1.3).

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

6.2.3.2 Interpretations

Admittedly, the projection of the speedup achieved by the iterative training approach is quite rough since the factor depends a lot on the pages the mixed model was applied to, which is also shown by the high fluctuations among the speedup factors. Moreover, in a real-world application scenario there has to be some kind of training and testing during the correction phase in order to know when to stop as the results from Table 6.1 have shown that the number of lines needed to reach a certain CER varies considerably.

6.2 Precise Segmentation and Trained ATR of Early Printed Books

Table 6.2: Evaluation of the iterative training approach. For each *Book* processed by a *User* we provide all values and results necessary to reconstruct and evaluate the progress of the GT production and training. In the *New Data* column the number of the newly added pages ($\#P$) and the corresponding number of lines ($\#L$) is listed, as well as the time required to produce the transcription. Furthermore, the CER is given which is calculated from the ATR result achieved by the model from the previous iteration and the newly created GT. For comparison, the CER achieved on the separate and constant evaluation set (*Eval*) is recorded. *All Data* shows the number of available GT pages and lines at this point which then serve as training data for the new model which is used for the next iteration. In the *Correction* columns we compare the actual required correction time when applying the iterative training approach (*ITA*) with the projected time when only using the output of the mixed model (*MM*) to get to the point where the final (and in case of *MM* the first) model is trained. The speedup factor (*SU*) is calculated for each book for the two user groups separately and for both groups combined.

Book	User	It.	New Data					Eval	All Data		Correction		
			#P	#L	$t_{\text{Corr.}}$ [min]	$t_{\text{Corr.}}$ [s/L]	CER [%]	CER [%]	#P	#L	ITA [min]	MM [min]	SU
C1541	2B	1	3	88	18	12	4.80	5.51	3	88			
		2	5	146	23	9.5	2.52	3.09	8	234			
		3	20	613	41	4.0	0.90	1.29	28	847			
		4	-	-	-	-	-	0.92	-	-	82	169	2.1
P1484	2B	1	5	110	14	7.6	3.53	3.95	5	110			
		2	6	116	8	4.1	0.89	1.48	11	226			
		3	-	-	-	-	-	0.34	-	-	22	29	1.3
N1499	2A	1	2	105	65	37	25.22	23.59	2	105			
		2	3	138	20	8.7	0.54	2.23	5	243			
		3	5	389	25	3.9	1.24	1.63	10	632			
		4	-	-	-	-	-	0.20	-	-	110	474	3.6
										Mean(2): 2.3			
C1557	2B	1	4	104	16	9.2	2.00	10.00	4	104			
		2	11	307	70	14	6.06	8.64	15	411			
		3	15	407	56	8.3	1.60	1.17	30	818			
		4	20	524	45	5.2	0.26	0.65	50	1,342			
		5	-	-	-	-	-	0.34	-	-	187	206	1.1
C1558	1A	1	4	125	38	18	15.31	16.86	4	125			
		2	8	251	60	14	1.28	0.65	12	376			
		3	12	375	85	14	0.58	0.34	24	751			
		4	-	-	-	-	-	0.25	-	-	183	225	1.2
C1566	1D	1	5	122	15	7.4	3.85	4.27	5	122			
		2	6	126	18	8.6	3.15	1.45	11	248			
		3	12	348	15	2.6	0.22	0.99	23	596			
		4	-	-	-	-	-	0.61	-	-	48	74	1.5
										Mean(1): 1.3			
										Mean: 1.9			

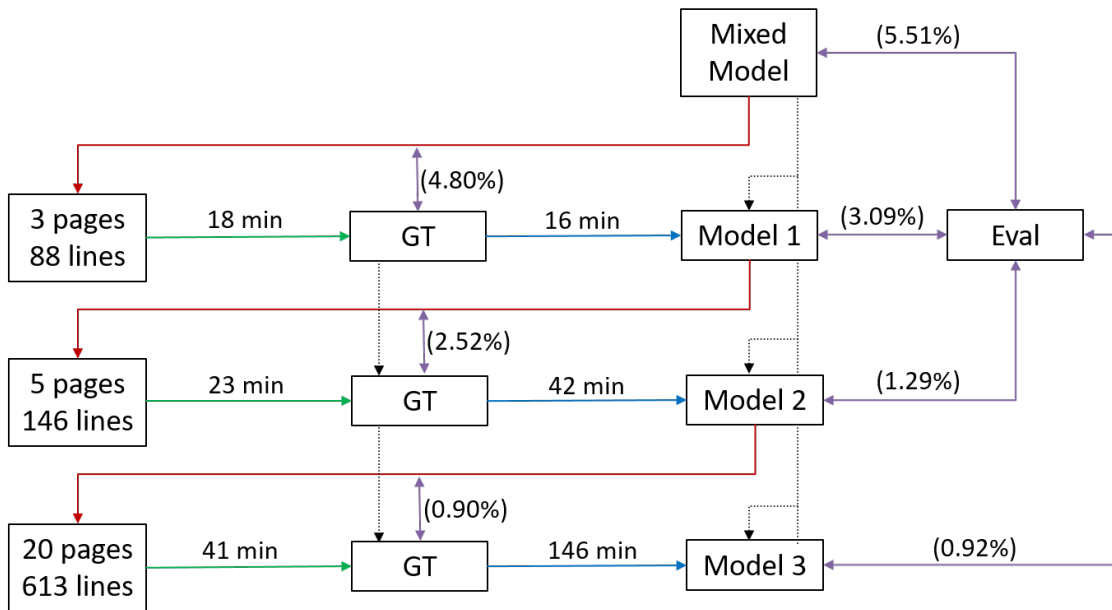


Figure 6.2: Schematic representation of the iterative training approach and its evaluation. As an example we used book C1541 processed by an experienced user. For comparison we refer to the first row of Table 6.2. To begin with, the user selects a few pages (here 3 pages comprising 88 lines) and applies a suitable *Mixed Model* to it. After investing 18 minutes to correct the results a first evaluation shows that the *Mixed Model* achieved a CER of 4.80% on the first batch of lines. Next, the produced GT can be used to train a first book-specific model (*Model 1*) which required 16 minutes, using the initial *Mixed Model* as a starting point. *Model 1* is then applied to the next batch (5 pages / 146 lines). After correcting the erroneous results (23 minutes, 2.52% CER) a second book-specific model is trained (*Model 2*, 42 minutes) using all available GT (8 pages / 234 lines) and again building from the initial *Mixed Model*. This process is repeated until a satisfactory CER is reached or the entire book is transcribed. For evaluation purposes a separate *Eval* dataset can be utilized which was not part of any training set. By applying the *Mixed Model* and the models produced during each iteration to this dataset and evaluating the results we can compare the models objectively. Adopted from [229].

Figure 6.3 depicts this problem and graphically explains the gain obtained by the iterative training approach.

Determining the ideal training route is no trivial task and depends on several factors. To begin with, the user has to estimate how many lines are necessary to reach the desired CER. Due to the variety of the material this is very challenging, even for experienced users, resulting in over and under estimations of the required amount. The smaller the chosen steps are the more accurate the convergence to the optimal value P (Figure 6.3) becomes. One (theoretical) approach is training a model each time a new line of GT is added (red curve, left), however this is not sensible. The other end of the spectrum is

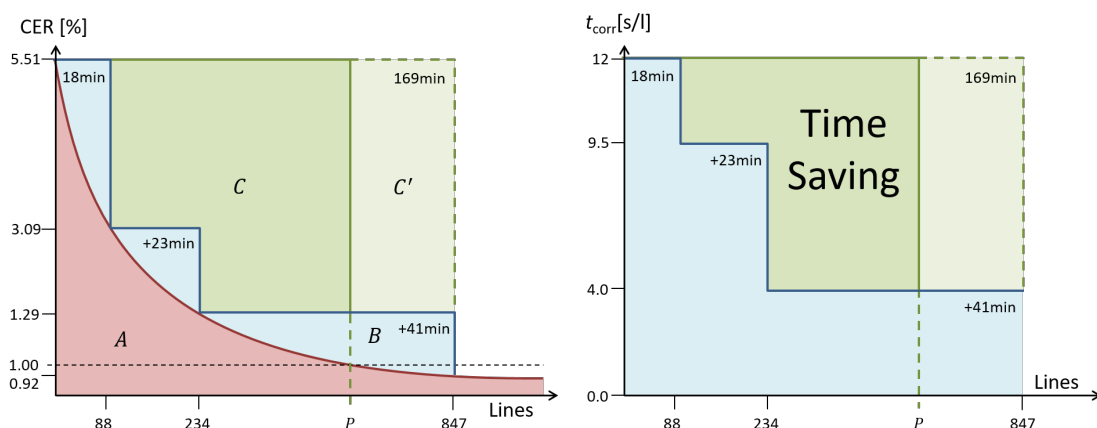


Figure 6.3: Left: Analysis of the iterative training approach using C1541. The goal is to reach point P which represents the (unknown) number of lines necessary to achieve a CER of below 1%. In theory, the red curve describes the unknown relation between GT lines used for training and the achieved CER and therefore represents the theoretical ideal iterative training approach that produces and applies a new ATR model after the transcription of a single new line of GT. The real route chosen by user 2B is shown by the blue stair case function. Green depicts a single training approach using the same final number of line of GT as user 2B ($C + C'$) and the perfect but unknown number of lines (C).

Right: This time the t_{corr} -coordinate shows the manual effort necessary to correct a single (which highly depends on the CER). The time saved by the iterative training approach (in case of user 2B and theoretically ideal) is represented by the green area minus the blue area. In this case this represents a reduction of the manual effort by ca. 52% which equates to an approximate speedup of 2.1. Adopted from [229].

represented by correcting the output of the mixed model until the presumably required number of GT lines is reached (green), which discards the gain of correcting lines with an improving CER (area ratio on the right). Consequently, the optimal or rather a sufficient real world solution has to lie somewhere in between these two extremes. Naturally, the available hardware plays an important role as it directly influences the training duration. For example, most training processes can be completed within a couple of minutes when using several GPUs, allowing the user to continue the transcription almost instantly. When no GPU support is available a training can take several hours, requiring the user to perform different tasks.

Despite the complexity of optimizing the iterative training approach, its general benefits are clear and the results confirm the expectations. This speedup is due to the fact that the average correction time per line clearly correlates with the quality (CER) of the underlying ATR result. The only exception can be seen in the iterations 1 and 2 of book C1566 where it took the user even a little longer to correct a line, despite starting from a somewhat better recognition result. Naturally, comparisons like this are only viable for a

6 Evaluations

single user and within the same book. Since a visual inspection of the concerned pages did not lead to new insights it stands to reason that human factors like tiredness, form on the day, etc. play a non-negligible role. Furthermore, it is worth mentioning that the correction time of course will not decrease linearly since the user will always require a certain amount of time for grasping the line image and reading the ATR result even with a theoretical CER of 0.0%.

In case of C1557 the reason for the striking difference between the CER on the new and the eval data is that most of the Camerarius books incorporate a frequent change of two completely different Antiqua fonts: upright and italics. Of course, other books make use of different fonts as well but mostly in less frequent layout parts like headings while in case of Camerarius they are used for the main text and therefore often fill entire pages. So when the ratio of pages/lines printed in upright and in italics varies considerably between the new and the eval data, this of course also effects the obtainable CER. With respect to C1557 the new data in the first iteration did not contain any italics lines so the mixed model, which was mainly trained on Antiqua upright performed quite well. On the contrary, the eval data had a significant portion of italics lines the default model could not handle. After creating some GT of italics lines during the first iteration, which is indicated by the relatively high CER (6%), the newly trained model can deal with both fonts, resulting in a considerably better CER on the eval set. To counter this problem we trained a new mixed model after several Camerarius books had been processed, which was not only able to deal considerably better with the upright/italics problem but also with the Greek characters.

N1499 is another interesting case. First, despite being printed in a rather normal looking Bastarda font and being among the best books in the corpus in terms of image quality, the ATR result obtained by the mixed model is by far the worst that occurred during our experiments, yielding a CER of around 25%. Naturally, this also explains the unusually high time expenditure necessary for correction we mentioned during the discussion of Table 6.1, since the correction of an ATR result flawed to that degree is a very cumbersome task and barely faster than transcribing the line from scratch, if at all. Second, while the recognition quality on the new data in the second iteration is great (0.54%) the resulting model performs significantly worse on the new data of the next iteration (1.24%). While the processing user (2A, the most experienced user participating in our experiments) could neither explain this by the scan quality nor the use of different fonts, a look at the confusion tables quickly identified the problem: In iteration 2, the error distribution looked like the ones produced by a well converged model, mainly consisting of the misrecognition of rare capital letters or typical ATR errors like the confusion of *c* and *e*. However, in iteration 3 the most frequent errors were dominated by previously negligible errors which all related to the characters *x*, *v*, and *j*. It turned out that the printer decided to use printing types with considerably different looking glyphs for different kinds of marginalia.

While some marginalia serve as reminders for the reader or simply repeat some keywords from the main text, others are referencing related books using Roman numerals which

are numbered and therefore rely on the affected characters a lot. The effect was further fortified as during the third iteration the pages added by the user contained plenty of marginalia containing references. This phenomenon did not only affect the recognition quality but also the correction time. Despite the considerably higher CER in iteration 3 the user handled these lines more than twice as fast on average as the lines from iteration 2. While the effect can partially be explained by the accumulation of very short lines (mainly marginalia), another reason, according to the processing user, clearly was the shift in the error distribution. As explained before, the main cause of error are the highly flawed reference numberings while the remainder of the text was recognized with very high accuracy. Understandably, this effects the amount of effort required for correction heavily since the errors are often clumped and can usually be corrected very efficiently, for example by deleting an entire number and simply typing in *xxviii* without even having to use the virtual keyboard once.

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

6.2.4 Segmentation Without Semantic Classification

As our first experiment has shown, the segmentation step can be considerably more time consuming than the ATR, even when aiming for very low CERs. Of course this is especially true for voluminous books since the effort necessary to obtain a certain recognition accuracy does not scale with the size of the book, but the segmentation effort does. However, the required manual work to segment a book can be severely cut down when the aspirations regarding semantic classification are less strict. Therefore, we conducted another experiment where the single goal of the segmentation is to provide the subsequent ATR with the means sufficient to produce the required output. Apart from a clean text/non-text separation this also includes ensuring the correct reading order. Figure 6.4 shows the desired results for example pages of the three books we used for this experiment: P1484, C1541, and N1506.

We selected these books due to their widely differing layout properties: While P1484's very simple layout apart from images and running text only uses a single additional semantic element (chapter headings), C1541 and N1506 both make use of a wide variety. Still, also these two books differ considerably since C1541's layout elements for the most part are simply arranged from top to bottom in a one column layout, with the comparatively rarely used marginalia being the only exception. N1506 however not only

6 Evaluations



Figure 6.4: Representative example image of three books showing the difference between the basic (top row) and exact (bottom row) segmentation approach. From left to right: P1484, C1541, and N1506 (three images). Adopted from [229].

has plenty of marginalia but also incorporates a complex two column sub layout on every other page.

For our experiment the users 1C and 2B both segmented 20 representative pages of each the books twice, first using the basic approach (text/non-text and reading order only) and then the complex approach from the first experiments including a fine-grained semantic classification. For all sub task we recorded the required times which are presented in Table 6.3.

6.2.4.1 Results

As expected the basic segmentation approach requires considerably less time than the exact one, leading to an average savings of 38% for the experienced and 45% for the unexperienced user. Regarding the comparison of two users with a different degree of experience the results in general show the expected tendency, namely a much faster processing by the more experienced user. On average it takes the novice user 2.65 times longer to perform the basic segmentation on the three books. This factor even rises slightly to 2.95 when an exact segmentation is required.

Table 6.3: Comparison of two users (1C and 2B) and two different segmentation approaches regarding the segmentation with LAREX. Apart from the processed *Book* and the experience of the *User* we differentiate between the two segmentation *Modes* (see Figure 6.4) basic (*B*) and exact (*E*). We give the *Times* required for the segmentation of 20 representative pages and for a single page on average. Finally, we calculate the ratio of the basic approach to the exact one (B / E) and between the less experienced user and the more experienced one ($1C / 2B$).

Book	User	Mode	Time		B / E	1C / 2B
			[s]	[s / P]		
P1484	2B	B	120	6	0.75	3.04
	2B	E	160	8		
	1C	B	365	18	0.59	
	1C	E	615	31		
C1541	2B	B	255	13	0.46	3.43
	2B	E	560	28		
	1C	B	875	44	0.46	
	1C	E	1,920	96		
N1506	2B	B	805	40	0.66	1.49
	2B	E	1,220	61		
	1C	B	1,200	60	0.61	
	1C	E	1,965	98		

6.2.4.2 Interpretations

When taking a closer look at the individual books it is eye-catching that the most time can be saved using the basic segmentation approach with C1541. This makes sense since dropping the requirement of semantic classification reduces the complexity of C1541 and thereby the typical Camerarius books significantly, especially because the reading order in the single column layout is always automatically correct, as soon as the marginalia, if present, have been cut off. In comparison P1484 does not allow to save as much time since the required manual effort remains the same for many pages as they simply contain running text only or running text with images which both are either segmented correctly automatically or have to be manually corrected, regardless of the segmentation mode. Consequently, there is a difference in the results from the (semi-)manual segmentation of the headings. As for N1506, bigger savings were not reached because the complex layout requires most of the manual interventions in order to ensure the reading order and not because of a correct semantic segmentation. For example, when the marginalia are correctly separated from the main text LAREX automatically assigns the correct type. In the two column sections the user can save some time by not separating the headings but the main matter of expense, that is the column separation in itself, remains.

It is worth mentioning the discrepancy between the two users, while being very similar for P1484 and C1541, is comparatively low for N1506, which at first seems odd, since N1506 is the book with the most challenging layout not only in this experiment but in the entire corpus. This effect can be explained when looking at the advantages an experienced has over an unexperienced one. The three main aspects are:

- Mental understanding of the layout.
- Optimized application of advanced segmentation techniques.
- Raw technical ability.

We think that experienced users can understand a layout considerably faster and then act accordingly right away. Furthermore, they can use LAREX much more efficiently by putting its automatic features to work in an optimized way. However, when dealing with extremely complex layouts that require lots of manual interventions, the overall degree of advantage is dominated by the effect of raw technical ability, which we consider to be significant but not as high as the one of the other advantages.

6.3 Fully Automated Processing

Next, we reduce the manual effort to a minimum by choosing a fully automated approach. Since the Dummy Segmentation (cf. Section 5.5.1.2) of OCR4all relies on the layout analysis functionality comprised in the OCRopus 1 line segmentation script (cf. Section 5.5.2), more complex layouts cannot be expected to be solved without human intervention. Consequently, we first focus our in-detail evaluation on the 19th century Fraktur models whose layout is usually rather trivial, before we turn to more challenging material.

6.3.1 19th Century Fraktur Novels

This experiment was performed on ten German Fraktur novels from the corpus described in Section 3.1.2 using the Calamari Fraktur 19th century ensemble which was trained on a wide variety of data also derived from 19th century Fraktur novels (cf. the case study described in Section 7.1 for details).

We randomly selected ten pages from each novel and processed them fully automatically with OCR4all as well as with ABBYY Finereader Engine CLI for Linux¹ version 11 together with ABBYY's historical Fraktur (*Gothic*) module and *Old German* language settings. The results were compared by calculating the CER on a page to page basis. To ensure a fair comparison several regularizations, for example the normalization of the long and short version of the *s*, were performed beforehand. Table 6.4 sums up the results.

6.3.1.1 Results

The values show that OCR4all considerably outperforms ABBYY Finereader on every single book resulting in an average improvement of over 84% and an improvement factor of almost 8 with respect to the error rate. On eight of the ten books CERs of below 1% are achieved while six books even yielded error rates below 0.5%. Wild fluctuations in CER can be observed for ABBYY Finereader (best: 0.48%, worst: 27%) but also for OCR4all (best: 0.06%, worst: 4.89%) caused by the highly variant quality of the scans as shown in Figure 3.4.

6.3.1.2 Interpretations

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

First of all it is apparent that the error distribution of the results produced by OCR4all is more top heavy, with the top ten making up for almost 30% of the total errors, compared to the one of ABBYY (less than 20%). However, a closer look shows that the distributions are actually quite similar to each other, apart from the top error of OCR4all, namely the deletion of whitespaces, which is responsible for almost 12% of the errors alone. Interestingly, while insertions and deletions of whitespaces represent the top two errors for ABBYY and OCR4all also fails to predict them on a regular basis, insertions of whitespaces do not occur in the top ten of OCR4all at all. The remainder of the most frequent OCR4all errors looks as expected, containing well-known errors

¹<https://www.ocr4linux.com/en>

Table 6.4: CERs achieved by *ABBYY* Finereader and *OCR4all* when being applied fully automatically to different *Books*. The final two columns indicate the percentual error reduction *ErrRed.* and the improvement factor *Impr.* yielded by *OCR4all* over *ABBYY*. Furthermore, we provide the average (*Avg.*) over all books for each value.

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

Table 6.5: The ten most common confusions over all ten books for *ABBYY* Finereader and *OCR4all*, consisting of the *GT*, the prediction (*ATR*), the counted number of occurrences (*CNT*) and the percent contribution (%) of a given confusion to the overall number of errors. Whitespaces are shown as \square and empty cells denote no prediction.

ABBYY				OCR4all			
GT	ATR	CNT	%	GT	ATR	CNT	%
\square		64	2.6	\square		63	11.9
	\square	57	2.3	n	u	14	2.7
s	S	57	2.3	f	s	12	2.3
	,	50	2.0	i	l	12	2.3
e	c	40	1.6	r	t	12	2.3
e	"	40	1.6	"	,	9	1.7
s	r	40	1.6	,		9	1.7
-	"	39	1.6	i		8	1.5
	.	36	1.5	c	e	8	1.5
x	"	35	1.4		,	6	1.1
Remaining			81.6	Remaining			71.1

like the confusion of similar looking (at least in 19th Fraktur script) characters like *n* and *u*, *f* and *s* (originally predicted as the long *s* and then regularized), or *c* and *e* as well as the insertions and deletions of tiny elements like commata, sometimes also as

part of quotation marks. ABBYY seems to struggle with quotations marks as well but mostly by confusing seemingly random glyphs like *e*, *-*, and *x* with them. Even when considering that the original recognition mostly showed french quotation marks (»«) which might explain some of the confusions, the notable accumulation of these errors remains inexplicable. Furthermore, some of the aforementioned typical ATR errors like the confusions of *e* and *c* and *s* and *S* still are surprising since one would expect the powerful dictionary and language modelling capabilities of ABBYY to deal with these errors quite comfortable. A possible explanation is that these postcorrection operations do not change characters that have been recognized with a certain degree of confidence to prevent the introduction of errors when “improving” out of dictionary words like unusual proper names. It is noticeable that 27 of the 57 *s/S* errors appear in a single book (F1851) but closer inspection did not lead to any new insights as the used glyphs differ considerably, are often recognized correctly, and no pattern regarding the misrecognitions could be observed.

Again, it has to be emphasized that these very low CERs can only be achieved when a highly performant mixed model is available. In this case we were able to rely on a strong voting ensemble perfectly matching the evaluation material. Unfortunately, comparable ensembles are not available for other scripts and languages, yet, maybe with the exception of AM for modern English (cf. Section 3.3.2).

6.3.2 Early Printed Books

As expected, the fully automatic processing of early printed books is a tricky task and its applicability highly depends on the layout and typography of the book at hand. The results presented above as well as some additional experiments led to the following, mostly qualitative observations:

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

6 Evaluations

Section 5.5.2). This happens because the line segmentation estimates the most likely height of a line on page level and then tries to find fitting lines. A preceding region segmentation prevents this problem from occurring.

- The available mixed models work reasonably well on the majority of books achieving an average CER of 7.7% on the corpus we used for our evaluations (see Table 3.2). However, since this is probably not good enough for most use cases, book-specific training is necessary. Additionally, the CERs vary considerably, ranging from below 2% to over 25% on the new GT of the first iteration of each book.

7 Case Studies

This chapter builds from the methodical contributions introduced in Chapter 4 and applies them in two real-world case studies: First, in Section 7.1 the proposed techniques from book-specific training are transferred to the task of training mixed models, using 19th Fraktur scripts as a test case. Second, in Section 7.2 we tackle the challenge of font identification in a case study with the goal to fully index a historical lexicon by combining ATR and typography recognition.

7.1 Application of Pretraining and Voting to Mixed Models: Case Study on 19th Century Fraktur Scripts¹

In this subchapter we evaluate OCR on 19th century Fraktur scripts without book-specific training using mixed models, i.e. models trained to recognize a variety of fonts and typesets from previously unseen sources. As we have shown during the core evaluations in Chapter 6, early prints suffer from a high variability in terms of printing types and therefore usually require book-specific training in order to reach desirable error rates below 1%. On the contrary, modern typography is much more regular and mixed models comfortably achieve error rates well below 1%, without book-specific training. Printings from the 19th century represent a middle ground between the two classes introduced above, considering both the variability of typesets and the state of preservation of the scans. Mixed models have achieved encouraging results without the need for book-specific training but the expectable recognition accuracy still is considerably lower than for prints from the 21st century [46]. Just as for modern prints, there is a great need for highly performant mixed models for 19th Fraktur scripts since there are masses of scanned data available online, consisting of a variety of materials including novels, newspapers, journals, and even dictionaries. To satisfy this need, we used two open-source engines, OCRopus 1 and Calamari, to train these models. In the progress we also investigate the transferability of the pretraining and voting techniques introduced in Chapter 4 from the application in book-specific training, usually dealing with a very manageable number of tediously created GT lines, to the production of mixed models utilizing several hundred thousand of lines.

¹This section is based on a previously published article [235]: C. Reul, U. Springmann, C. Wick, and F. Puppe, “State of the Art Optical Character Recognition of 19th Century Fraktur Scripts using Open Source Engines,” *DHd 2019 Digital Humanities: multimedial & multimodal*, 2019. [Online]. Available: <https://doi.org/10.5281/zenodo.2596095>

The rest of this subchapter is structured as follows: After discussing related work in Section 7.1.1, we briefly introduce the comprehensive data sets available to use for training and evaluation in Section 7.1.2. The training procedure is explained in Section 7.1.3. Sections 7.1.4 and 7.1.5 evaluate the trained models and discuss the results, before Section 7.1.6 concludes the subchapter.

7.1.1 Related Work

Only few evaluation results are available on 19th century Fraktur ATR data. A rare exception is the evaluation of the Fraktur model of OCRopus 1 trained on around 20,000 mostly synthetically generated text lines [46]. Evaluation on two books of different scan qualities yielded impressive CERs of 0.15% and 1.37%, respectively. An evaluation of ATR data on a wider range of Fraktur texts of different quality is missing. For an overview over the results achieved by mixed models on various materials we refer to Section 2.4.3

7.1.2 Data

In this section we first briefly introduce the data we used for training our models and the evaluation data which is entirely distinct from the training data on line but also on book level. For further information we refer to the corresponding Sections 3.2.1 and 3.2.2 in the Data Chapter. In addition, a brief overview over our transcription guidelines and the resulting codec is given.

7.1.2.1 Training Data

By far the largest part of our training corpus is the DTA19 Corpus (cf. Section 3.2.1.5) provides a very high number of lines (250,000) but “only” covers 39 books. To increase the variety of books and consequently fonts we incorporated the Archiscribe Corpus (cf. Section 3.2.2.1) containing 103 books and comprising close to 3,500 lines and the JZE Corpus (eight books comprising around 1,600 lines; cf. Section 3.2.2.2).

Moreover, we utilized data from foreign domains, i.e. not Fraktur data from the 19th century, in order to perform a pretraining to increase robustness against noise and train abstract features. This data consists of the historical corpora EML, Kallimachos, RIDGES, and ENHG (all introduced in Section 3.2.1) as well as the UW3 data set (cf. Section 3.2.3) which comprises modern Antiqua fonts.

7.1.2.2 Evaluation Data

To enable the proper utilization of language models, if available within the respective engine, we used entire pages as recognition input instead of randomly mixed lines. Additionally, our aim was to evaluate the engines and models on divergent data regarding age,

7.1 Application of Pretraining and Voting to Mixed Models

typeset, quality of the original printing, state of preservation, and source of digitization. The subsets we used for evaluation are:

- “Novels” (*N*): 19th century novels (cf. Section 3.1.2).
- “OCR-Testset” (*O*): The novel “Die Wahlverwandtschaften” and the journal “Die Grenzboten” from the CIS OCR Testset (cf. Section 3.2.2.3)
- “Daheim” (*D*): German journal published between 1864 and 1943 (cf. Section 3.2.2.4).
- “Sanders” (*S*): Daniel Sanders’ *Wörterbuch der Deutschen Sprache* (cf. Section 3.1.3).

Figure 7.1 shows a representative selection of example lines and Table 7.1 provides some additional information about the evaluation data.

For *N* and *O* we randomly selected ten pages for each work. Due to the vastly higher number of lines per page and the occasionally considerably longer lines we settled for one page per volume for *D*. As for *S* we decided to keep all available evaluation data, i.e. six columns transcribed during another project, resulting in a significantly higher amount of GT compared to the other subsets, which will be further addressed during the experiments section.

7.1.2.3 Transcription Guidelines and Resulting Codec

Before starting the training, we had to make several decisions regarding the codec, i.e. the set of characters known to the final model. During a first check of the present GT data we noticed several very rare characters, e.g. Greek letters, which only occurred in an almost negligible number of lines and seemed to be pretty uncommon for 19th century Fraktur data. In order to keep the codec rather small and therefore minimize the possibility of confusion, we decided to remove the lines containing these unwanted characters. Some French characters like é and á were kept since they appeared comparatively frequently.

Furthermore, we determined the following set of rules regarding regularizations:

- The *f* was kept as it represents the embodiment of 19th century Fraktur.
- All ligatures were resolved with the exception of *sz* since it can be transcribed as a single standard unicode character.
- Different depictions of Umlauts are regularized to the more modern spelling *ä*, *ö*, and *ü*. Since both variants are semantically equivalent they can easily be regularized after the recognition if desired.
- For the same reasons we refrained from the depiction of different forms of quotation marks and opted to use the simple double quotes for opening as well as closing.
- The *r rotunda* was regularized to *r* since it does not carry semantic information.

bat ihn wegen seines eigenen Glütes, sich nicht in
~~alle Zweige der gemischten Mathematik, wie~~
wins ragt Sigurds Schreckhorn hell hinaus
Borsaal, und ihre schönen Töne zogen alle
nur eine Kluft, die uns von dem gewünschten
So träumte er; ein Schwarm aufflatternder Eulen
in unsern Tagen von hundert Personen neun und
und entdeckte zuletzt einen schmalen hölzernen Steg;
lassen, daß das schwächliche Kind des reichen Hart-
er dadurch verrathen würde, daß er wirklich eine Neigung
Geschäftig eilten die Diener herbei. Der Herzog und
mich in Harnisch. Es handelt sich also nicht um einen
find der Erste, dem ich von meinem Leben erzähle, damit

Man hat einen vortrefflichen Anblick: unten
äußerung, wie sie die auf ihre Freiheit stolzen, eifersüchtigen

Zwar hat schon manche exilirte Löwenfamilie auf Europa's un-
nicht verkennen. Aber sind wir Menschen nicht gar zu
Es war Hildegard, als ob er seufze, aber das konnte ja
Ausleerungen fand sich auch der richtige Cholerabacillus.

zu den Sternen führende Bahn: Die steile St. Sch. 11b.

Figure 7.1: Example line images of the 20 evaluation works in the order given in Table 7.1. The horizontal lines mark the subsets introduced above (from top to bottom: N, O, D, and S). For practical reasons, all lines have been vertically normalized and some line have been shortened. Adopted from [235].

- We mapped the capital letters *I* and *J* to *J* since the vast majority of 19th century Fraktur typesets use the same glyph to depict both characters.
- Different length hyphens can carry semantic information but we still decided to regularize all of them to -. The reason for this was their variant transcriptions in the GT data combined with their susceptibility for confusion.

Applying these rules resulted in a codec consisting of 93 characters.

Table 7.1: Details on the evaluation data including the *ID* of the single works, consisting of the abbreviation of the subset and the printing year, the *Title*, and the number of transcribed lines (*# Lines*).

ID	(Short) Title	# Lines
N-1781	Eleonore	305
N-1803	Liebe-Hütten	184
N-1810	Der Held des Nordens	264
N-1818	Reinhold	253
N-1826	Frauenwürde	268
N-1836	Die Ruinen im Schwarzwalde	318
N-1848	Levin	269
N-1851	Georg Volker	264
N-1859	Der beseelte Schatten	260
N-1865	Gefahrvolle Wege	333
N-1869	Der Arzt der Seele	250
N-1870	Die Bank des Verderbens	273
N-1873	Natürliche Magie	242
O-1809	Wahlverwandtschaften	223
O-1841	Grenzboten	242
D-1865	Daheim volume 1865	134
D-1875	Daheim volume 1875	144
D-1882	Daheim volume 1882	142
D-1892	Daheim volume 1892	163
S-1865	Sanders Dictionary	630

7.1.3 Training Procedure

Next, we describe the training procedure in detail. After giving a brief overview over the general workflow and our goals we explain the preprocessing of the available GT lines and the different stages of the model training.

7.1.3.1 Overview and Goals

The training procedure consists of three basic steps: pretraining on various, non 19th century data (enhancing robustness and improving voting capabilities), training with synthetic data (targeted improvement of weak spots), and finally, the training on real 19th century data to finish off the model(s).

If possible and sensible we perform several trainings per step in order to achieve the following goals. We aimed to train

- models for **OCRopus 1** because it is widely used as well as for **Calamari** since it has shown excellent recognition capabilities and natively supports cross fold training, voting and pretraining.
- models which are optimized to recognize **binary** line images but also models that can make use of the additional information provided by **grayscale** line images.
- very capable **single models** as well as an **ensemble of models** whose outputs can be combined via voting to considerably increase recognition accuracy even further. Note that the ensemble approach is not applied to OCRopus 1 since it does not support voting in its default configuration.

7.1.3.2 Preprocessing

Before starting with the training, we have to perform some preprocessing in order to convert the lines into a format compatible with the OCRopus 1 and Calamari networks as well as to ensure data consistency. The line images comprised by the DTA19 and Archiscribe dataset have been obtained by cutting out along the line bounding boxes detected by ABBYY Finereader (cf. Section 2.6.1). These boxes are usually tightly fitted around the letters of the line. As opposed to this, OCRopus 1 by default adds a 3 pixel padding, i.e. three rows of white pixels, to the top and bottom of each detected line during the line segmentation step. Considering the methodology of processing vertical slices applied by OCRopus 1 and Calamari during the training and recognition steps, it is apparent that the presence or absence of padding can have a big impact on the outcome. Consequently, we also added a 3 pixel padding to the lines which were obtained from ABBYY Finereader segmentation results. Finally, we convert the line images to binary and normalized grayscale images by applying the OCRopus 1 binarization algorithm to all lines.

7.1.3.3 Pretraining

As shown in Section 4.2, building from one or several existing models is almost always preferable to starting from scratch even when the models do not really fit the training data. Hence, we tried to utilize as much available GT as possible to let our models gain robustness against noise and learn meaningful high level features such as edges, corners, and curves. Different starting points also showed to be very effective in terms of increasing the diversity between voters, improving their voting performance in the process (cf. Section 4.3). Consequently, we laid the groundwork for the voting ensemble by training five models, one on each of the corpora described above: EML, Kallimachos, RIDGES, UW3, and ENHG. Apart from the voting ensemble we also trained a model on all five corpora combined to obtain a starting point for the single model approach.

7.1.3.4 Adding Synthetic Data

Training on synthetic data can be very effective when dealing with 19th century Fraktur. The standard OCRopus 1 Fraktur model was trained on 20,000 mostly synthetic lines and showed to be very capable when applied to real world data [46]. Furthermore, training with artificially generated data allowed us to deal with known weaknesses like numbers or rare capital letters in a purposeful way.

We used the OCRopus 1 *linegen* script to generate the data. Apart from the the desired GT text suitable true type fonts are required as input. Therefore, we collected 66 freely available fonts and used close to 244,000 text lines already available to us in the form of line-based GT. However, we did not include text from the evaluation data.

Since the codecs of the fonts differ some of them are not able to render all required characters. For example, almost one third of the fonts cannot display the *f*. Of course, we could have regularized the available GT to the common denominator of all fonts to circumvent rendering problems. However, our goal was to cover as many characters of our desired codec as possible during the training with synthetic data in order to maximize robustness and performance. Therefore, we adapted the GT on a font to font basis by checking the codec of the font and regularizing accordingly by enforcing some simple rules. For example, if a font could not render the *f* we replaced all occurrences by *s* in the font specific GT. Similar rules applied for hyphens of different length, umlauts, or vowels with accents. During a last step, we focused on known weak points of existing polyfont models which usually can be attributed to a lack of training data: numbers and rare capital letters. We tackled the first problem by adding lines either containing just a single number between 0 and 999 or lines with a number and frequently occurring markups which are typical for page numbers, e.g. “- 123 -”. Additionally, we randomly added numbers to the existing GT lines to prevent the network from jumping to unwanted conclusions regarding the position and surroundings of numbers. In order to cover rarely occurring capital letters (e.g. “ÄÖÜQY”) we used a freely available list of suitable words² and inserted them into the existing GT.

After the described preparations we finally generated around 1,500 lines for each font resulting in a total of close to 100,000 lines of synthetic GT. In addition to the aforementioned inclusion of numbers and words with rare characters, random distortions were added by generating 80% of the lines using the low degradation model provided within the *linegen* script, 15% with the medium and 5% with the high one. Figure 7.2 shows some examples of generated lines.

Finally, we trained on the synthetic data, starting from the pretrained models. In case of the ensemble approach we applied a standard five fold cross fold training approach. The single model resulted from using 80% of the generated lines for training and the remaining 20% for selecting the best model. Since the *linegen* script only allows for the creation of binary images we did not differentiate between binary and grayscale images until the upcoming training steps.

²https://github.com/jze/ocropus-model_fraktur/blob/master/words.txt

voriger Stunde als diejenigen Kräfte gedacht,
näher kam, ohne an den Schwierigkeiten zu scheitern,
Gerhardus und Gudula, Sand in Sand,
160 **171** ~~255~~ ~~496~~
Augen 539 auf die Knie, die Äpfel Hände fallend zum

Figure 7.2: Examples of synthetically generated lines: three standard lines generated with different fonts and varying degrees of degradation (lines 1-3), page number lines (line 4), and a line with a number (539) and a word containing a rare character (*Äpfel*) embedded in real text (line 5). Adopted from [235].

7.1.3.5 Adding Real Fraktur Data

While pretraining on diverse data and synthetic data can offer a lot of robustness, nothing compares to training on real data. Therefore, the last training steps are exclusively performed on real lines containing 19th century Fraktur fonts.

The number of available lines varies considerably from book to book. While some books of the DTA19 dataset contain thousands of lines, most of the works transcribed by using Archiscribe only comprise 50 lines or even less. Naturally, we want our models to be trained on as much appropriate real data as possible but we also want to prevent the models from overfitting towards the typesets of the books with the most available lines. Therefore, we divided the training with real data in two steps: After running the first training on all available lines (ca. 285,000) we chose a maximum of 50 lines of every available book. During the final training we use this selected data (5,634 lines from 148 books) to refine and consequently balance the mixed models and force them into performing well on a wide variety of fonts. The training procedures were performed analogously to the approach described during the previous section.

7.1.4 Experiments

In this section we first sum up the engines, models, and evaluation data before we thoroughly evaluate the trained models. For practical reason, we first identify the best performing trained model for the three general approaches – *OCROPUS 1*, *Calamari single*, and *Calamari voted* – and then perform the rest of the evaluations with this identified subset of models. Before finally comparing the performance of our models to existing approaches, we investigate the influence of the single training steps.

7.1.4.1 Engines, Models, and Data

During the upcoming sections we evaluate the following open-source and commercial ATR engines and models:

7.1 Application of Pretraining and Voting to Mixed Models

- Calamari: We trained a voting ensemble as well as a single model on both binary and grayscale line images, respectively.
- OCRopus 1: Apart from training a single binary and grayscale model we also incorporate the standard OCRopus 1 Fraktur model (FRK; cf. Section 3.3.1) into our evaluations.
- ABBYY: For comparison we use the commercial state-of-the-art system ABBYY Recognition Server 4 using the *Gothic* recognition mode and setting the dictionaries to *German* and *Old German*.
- Tesseract 4: As another out of the box open-source alternative we also used Tesseract’s *Fraktur* script recognition model. There are other suitable mixed models available, for example *deu-frk* but we chose the one which performed best during preliminary tests.

For evaluation we utilize the data introduced above. Apart from the predefined single subsets of *N*, *O*, *D*, and *S* we also always accumulate the results for all subsets (*All*). Additionally, we give the overall results for all subsets except “Sanders” – *NOD* – as the material has to be considered very special and does not necessarily represent the usual field of application for generic 19th century Fraktur ATR models.

7.1.4.2 Evaluating the Trained Models

For reasons of clarity, we first want to identify the best performing models and use only them for further evaluations. Furthermore, for the moment we refrain from listing the results for each individual book and instead only show the pooled results for the four evaluation sets. Thereto, we recognized the evaluation data with the final single models and voting ensembles and measured the CER. Table 7.2 shows the results.

There are several things to be taken away from these results: First of all, the superior recognition capabilities of the deep network structure of Calamari become apparent as even the single models considerably outperform their OCRopus 1 counterparts reducing the CER by close to 60%. As expected, the application of voting ensembles pushes down the CER even further, leading to additional improvements between 30 and 40% for *NOD*, *All*, and the four subsets. Contrary to our initial expectations, the binary models outperformed the grayscale ones on every single subset, with the exception of *D* where both OCRopus 1 models perform equally well. Most likely, this behaviour can be attributed to the line segmentation methodology. When OCRopus 1 splits a page or segment into lines it identifies the characters of a line, extracts the related CCs, and finally copies them into a clean line image. However, for our experiments, we decided to cut out the line rectangles detected by ABBYY instead, to ensure that we provide equal starting points and focus entirely on the recognition performance of the various engines. Admittedly, the extraction of the line rectangles has some drawbacks as it may include parts of characters from adjacent lines or, in the case of grayscale images, background noise. During previous experiments, unwanted inclusions have proven to be negligible in the

Table 7.2: ATR results achieved by the final OCRopus 1 and Calamari models. *bin/gray* indicates whether the models have been trained and evaluated on binary or grayscale line images. *single* denotes results achieved by a single ATR model, whereas the *voted* results were provided by an ensemble of five models/voters whose outputs were combined by confidence voting. Since the *OCR-TS* only offers binary images, we put the results achieved by the grayscale models into parenthesis.

Eval Sets	OCRopus 1		Calamari			
	bin single	gray single	bin single	voted	gray single	voted
<i>N</i>	1.69	2.28	0.75	0.47	0.98	0.61
<i>O</i>	1.01	(2.07)	0.19	0.11	(0.50)	(0.24)
<i>D</i>	0.47	0.47	0.16	0.09	0.24	0.13
<i>S</i>	5.55	6.12	3.00	2.14	4.20	2.76
<i>NOD</i>	1.39	(1.96)	0.59	0.37	(0.80)	(0.49)
<i>All</i>	1.95	(2.52)	0.91	0.61	(1.25)	(0.79)

vast majority of cases, especially when the models used for recognition have been properly trained on similar data, like it is the case here. Nevertheless, the background noise in grayscale images seems to have a noteworthy impact on the recognition accuracy even when the utilized models have been thoroughly trained. Furthermore, the synthetically created data exclusively consisted of binary images which might well contribute to the superiority of the binary models. Based on the obtained results, we decide to use the binary models for the remaining experiments of this case study.

7.1.4.3 Influence of the Training Steps

Next, we examined the influence of the different training steps on the performance of the best models. For practical reason we refrain from performing an exhaustive evaluation of all possible combinations of training steps. Instead, we focus on the contribution of the early training steps to the overall performance as well as the relevance of the last training step – the refinement with a maximum number of 50 lines per book – which is of particular interest to us. Table 7.3 summarizes the results.

Interestingly, the behaviour shown by the three evaluated approaches varies considerably. The OCRopus 1 model achieves the best results when the first two training steps are skipped and only real fraktur data is used. As for the single Calamari model, the same model and the model trained over all four training stadiums show almost no difference in their performance. However, the fully trained voting ensemble considerably outperforms the models which skipped one or several training steps. In contrast to our previous findings, where we performed book-specific training using pretrained mixed models and only a few lines of GT, the extensive pretraining on only partly fitting or synthetic data

7.1 Application of Pretraining and Voting to Mixed Models

Table 7.3: Results for the models emerging from different *Training Steps* for the single subsets (*N*, *O*, *D*, and *S*), *NOD*, and *All*. Each step that was performed during the training pipeline is marked by an “x”. For example, the first line for each model indicates that the pretraining (*pt*), training on synthetic data (*synth*), and training on all available real Fraktur data (*real*) steps were performed but the refinement step (*refine*) was skipped. The optimum value for each data set is marked bold. The *Err. Red.* column shows the proportion of errors the fully trained models (second row for each approach) get rid of in comparison to the reduced models of that row. For example, the fully trained OCRopus 1 model commits 19% less errors on *NOD* and 21% on *All* compared to the OCRopus 1 model which skipped the refinement step. A negative value therefore indicates that the respective reduced model performed better than the corresponding fully trained one.

OCRopus 1 binary single											
pt	Training Steps			CER						Err. Red.	
	synth	real	refine	N	O	D	S	NOD	All	NOD	All
x	x	x		2.16	0.91	0.38	7.43	1.72	2.48	19%	21%
x	x	x	x	1.69	1.01	0.38	5.55	1.39	1.95	-	-
		x	x	1.61	0.82	0.33	6.22	1.31	1.96	-6%	1%
			x	1.58	0.77	0.38	5.91	1.29	1.90	-8%	-3%
			x	1.72	1.21	0.58	5.95	1.47	2.07	5%	6%

Calamari binary single											
pt	Training Steps			CER						Err. Red.	
	synth	real	refine	N	O	D	S	NOD	All	NOD	All
x	x	x		0.92	0.24	0.11	2.72	0.70	0.94	21%	11%
x	x	x	x	0.71	0.17	0.14	2.74	0.55	0.84	-	-
		x	x	0.79	0.22	0.12	3.18	0.62	0.96	11%	13%
			x	0.75	0.23	0.17	3.37	0.59	0.96	7%	13%
			x	0.89	0.36	0.24	4.08	0.72	1.17	24%	28%

Calamari binary voted											
pt	Training Steps			CER						Err. Red.	
	synth	real	refine	N	O	D	S	NOD	All	NOD	All
x	x	x		0.67	0.14	0.08	2.14	0.51	0.73	27%	16%
x	x	x	x	0.47	0.11	0.09	2.14	0.37	0.61	-	-
		x	x	0.55	0.13	0.10	2.49	0.43	0.70	14%	13%
			x	0.55	0.14	0.14	2.82	0.43	0.75	14%	19%
			x	0.63	0.23	0.18	3.74	0.51	0.94	27%	35%

slightly hurts the performance of our single models. It seems natural pretraining loses its effect as, in this application, we have a huge number of real GT lines available to us. Nonetheless, it seems odd that the resulting models even perform slightly worse than the ones trained from scratch. Fortunately, the voting ensemble benefits significantly from the earlier training steps which can be attributed to the fact, that the extensive pretraining and especially the first step, led to to a considerable increase of diversity between the

voters. Finally, all three approaches gain considerably from the final refinement step since the models are forced to generalize better onto a large variety of fonts.

7.1.4.4 Comparison with other Engines and Models

Finally, the identified best self-trained models are compared against the engines and polyfont models introduced above. As for OCRopus 1 and the single Calamari model we use the models that skipped the first two training steps while the best Calamari voting ensemble passed through all four training steps. For a more in-detail analysis we measured the CER for every single work of the dataset. The results are shown in Table 7.4.

The error rates for the individual books confirm the results gathered from evaluating the different sets. Despite achieving impressive recognition results, below 1% CER for about two thirds of the single works, the OCRopus 1 model gets considerably outperformed by the single Calamari model on every single occasion. The same applies to the Calamari ensemble compared to the single model, with exception of D-1875, where the voting approach leads to no improvement. Compared to FRK, our OCRopus 1 model reduces the CER by more than 50% for NOD, the single Calamari model by close to 80%, and the voted approach by over 85%. The discrepancy rises with increasing printing quality of the subsets from *N* over *O* to *D*. As for ABBYY, our models lead to improvement rates for *NOD* of 46% (OCRopus 1), 77% (Calamari single), and 84% (voted). While the models significantly outperform ABBYY on *N* and *O*, they cannot match the almost perfect results on *D*. Another eye-catching result is the great variation among the CERs, e.g. by a factor of more than 2,500 from 26.54% to 0.01% for ABBYY and more than 400 from 4.75% to 0.01% for Calamari voted, which apparently depends on the quality of the scans as well as the similarity of each font to the training data.

To get a better impression of the results and probably also the strengths and weaknesses of the evaluated approaches, we summed up the confusion statistics for *NOD* in Table 7.5.

One thing that immediately stands out is that all four approaches share a common most frequent error: the deletion of whitespaces. This represents a common problem with historical prints, as the inter word distances vary heavily. Another eye-catching property is the overall distribution of errors. While ABBYY's and OCRopus 1's distribution is relatively flat, the two Calamari approaches show a considerably more top heavy accumulation of errors which is also indicated by the amount of remaining errors outside of the top ten. Interestingly, the Calamari models seem to struggle with the recognition of quotation marks and confuse them with single primes. It is worth mentioning, that OCRopus 1 struggles with umlauts and frequently recognizes *äöü* with their vocal counterparts *aou*, whereas the other approaches comfortably deal with these cases. Naturally, the confusion of *f* (small *F*) and *s* stem from the miss-recognition of a *f* (long *s*) which for evaluation purposes got then normalized to *s*.

7.1 Application of Pretraining and Voting to Mixed Models

Table 7.4: In detail comparison of our trained models using *OCRopus 1* and *Calamari* with the standard Fraktur models of Tesseract (*Tess*), OCRopus 1 (*FRK*) and *ABBY* Finereader. The CERs for all single works are given as well as the accumulation for the four individual subsets, *NOD*, and *All*. The three rightmost columns indicate the percentual improvement of our models compared to *ABBY*.

Data	Tess	FRK	OCRopus 1	ABBY	Calamari		Impr. over ABBY		
	bin single	bin single	bin single	original default	bin single	voted	OCRopus 1 single	Calamari single	voted
N-1781	6.61	4.08	2.48	2.79	0.81	0.56	11	71	80
N-1803	17.17	18.21	11.30	26.54	6.38	4.75	57	76	82
N-1810	5.26	5.30	1.92	3.22	0.45	0.21	40	86	93
N-1818	7.90	7.73	3.85	9.30	1.85	0.96	59	80	90
N-1826	2.77	1.00	0.40	1.04	0.08	0.01	62	92	99
N-1836	6.88	4.68	2.01	2.70	0.70	0.56	26	74	79
N-1848	1.58	1.17	0.33	0.57	0.08	0.02	42	86	96
N-1851	1.93	0.63	0.24	0.70	0.09	0.04	66	87	94
N-1855	4.58	4.42	1.38	3.83	0.80	0.58	64	79	85
N-1859	2.19	1.42	0.31	0.38	0.17	0.08	18	55	79
N-1865	2.44	1.31	0.62	1.23	0.19	0.13	50	85	89
N-1870	2.09	1.97	0.43	0.47	0.26	0.10	9	45	79
N-1873	2.53	1.14	0.32	0.34	0.22	0.14	6	35	59
N-all	4.39	3.42	1.58	3.13	0.71	0.47	50	77	85
O-1809	3.04	2.22	1.13	1.62	0.26	0.20	30	84	88
O-1841	2.09	1.06	0.60	0.79	0.13	0.07	24	84	91
O-all	2.40	1.44	0.77	1.06	0.17	0.11	27	84	90
D-1865	2.10	1.85	0.71	0.16	0.26	0.17	-344	-63	-6
D-1875	1.50	0.85	0.17	0.04	0.09	0.09	-325	-125	-125
D-1882	1.53	1.17	0.43	0.09	0.20	0.12	-378	-122	-33
D-1892	0.90	0.45	0.23	0.01	0.02	0.01	-2,200	-100	0
D-all	1.48	1.05	0.38	0.07	0.17	0.09	-443	-100	-29
S-1865	5.12	10.02	5.91	5.47	2.74	2.14	-8	50	61
NOD	3.68	2.80	1.29	2.38	0.55	0.37	46	77	84
All	3.87	3.76	1.90	2.80	0.84	0.61	32	70	78

Table 7.5: The ten most common confusions on *NOD* for *ABBY*, *Tesseract*, and our self trained models (*OCROPUS 1* and *Calamari voted*), consisting of the *GT*, the prediction *ATR*, and the percentual contribution (*PERC*) of a given confusion to the overall number of errors. Whitespaces are shown as \square and empty cells denote blanks, i.e. no prediction.

ABBY			OCROPUS 1			Tesseract			Calamari voted		
GT	ATR	PERC	GT	ATR	PERC	GT	ATR	PERC	GT	ATR	PERC
\square		1.63	\square		3.97		\square	7.06	\square		8.99
	\square	1.53	u	n	2.47	ch	<	6.58	"	'	5.82
e	c	1.42	i	t	2.17	.	,	2.42		.	1.39
s	S	1.38	ü	u	1.94		-	2.37		,	1.39
	,	1.25	ö	o	1.90		.	2.20		\square	1.27
	.	1.12		\square	1.87	c		1.95	f	s	1.14
e	"	0.88		n	1.54	h	<	1.48	n	u	1.14
-	"	0.86	i	l	1.34		:	1.41	,		1.14
s	r	0.77	ä	a	1.27	d	v	1.15	u	n	0.89
	-	0.69	t	e	1.27	s	f	1.14	i	t	0.89
Remaining		88.47	Remaining		80.27	Remaining		72.24	Remaining		71.94

7.1.5 Discussion

First of all, we showed that our trained 19th century Fraktur mixed models can significantly outperform the commercial state-of-the-art system ABBYY on most evaluated materials. The specific training on a very extensive amount of GT combined with the impressive recognition capabilities of OCROPUS 1 and especially Calamari led to more precise and particularly more flexible and robust models than the more generic ABBYY approach. However, ABBYY performed almost flawlessly on the Daheim subset which apparently has not only been printed in a font very similar to one ABBYY was trained on but also provided by far the best state of preservation and scan quality compared to the other subsets. Of course, the other engines also profit from this, which can be deduced from the Daheim subset having by far the lowest average CER of all subsets, but even the Calamari voting ensemble still performs considerably worse than ABBYY. It is standing to reason that the availability of a seemingly very performant language modeling and dictionary functionality gives ABBYY the edge. Of course, these postprocessing methods also come in handy when the raw recognition accuracy is (considerably) lower but it makes sense that they perform best when already provided with an ATR output which comprises at most 1-2 errors per sentence.

As expected, the deep network structure of Calamari is undeniably better suited to the given task than the shallow OCROPUS 1 LSTM. The considerably higher number of trainable parameters in the CNN-LSTM combination of Calamari allows the network to almost perfectly learn the different typesets from hundred of thousands of lines.

7.1 Application of Pretraining and Voting to Mixed Models

Moreover, Calamari profits massively from the ability of recognizing text with several models and combine their outputs in an ensemble approach utilizing confidence voting. This underlines the efficiency of the cross fold training and voting approach and proves that it is not only applicable to book-specific training with a small amount of GT but also to the training of mixed models with an immense number of lines available.

The same can be said about the pretraining approach which benefitted the different trainings in three ways: First, the training on only remotely related data, i.e. text lines printed in Latin script but otherwise very different in terms of typography or age, is thought to lead to a higher robustness against noise and supports the learning of abstract features. The OCRopus 1 model was not able to improve with this additional training step as its shallow network cannot be expected to offer enough learning capacity in order to profit from this kind of extra data. This holds especially true, when there are close to 250,000 lines of fitting GT available for the model training. Second, the effect of pretraining on different data in the very first training step still seems to linger until the end and leads to more diverse voters in the ensemble approach, resulting in a better voting performance. Third, the final refinement step proved to be very effective for all three approaches. Overall, the method to train the models step by step and even reuse the already trained data in an adjusted way appeared to be the way to go when training very comprehensive mixed models.

Unfortunately, the inclusion of synthetic training material did not yield the desired results as its addition barely improved or even slightly worsened the results compared to only performing the final to training steps on real Fraktur data. Since the FRK was trained almost exclusively on synthetic data there had been high hopes for this approach. However, as mentioned above, nothing compares to real data. When reconsidering, our training situation is way different to the one on the FRK model as we have plenty of real data available to us which lessens the effect of incorporating synthetic data up to a point where it becomes irrelevant. Additionally, we have to take the overall recognition quality into consideration. Synthetic data did a good job to get the CER down to around 3% but our approaches reduced it by a further factor of about two to six.

Finally, it has to be said that our experiments were performed on well segmented line images. The segmentation process is a very challenging task which has not been solved in a generic and satisfactory way, yet. It is standing to reason to utilize the very robust ABBYY segmentation and combine it with the (for the most part) superior recognition Calamari capabilities. Consequently, this *CalamABBY* functionality was directly integrated into the Calamari prediction script and allows to provide page images and corresponding ABBYY XML files as an input. The script then extracts the lines according to the coordinates stored in the XML output, applies the necessary preprocessing steps, recognizes the lines using standard Calamari models, e.g. the strong polyfont models trained during this case study, and finally outputs the results in another ABBYY XML file.

7.1.6 Conclusion and Future Work

Our evaluations have shown that, depending on the material at hand, open-source engines and mixed models are capable of matching or even considerably outperforming the commercial state-of-the-art system ABBYY. The resulting models have been made publicly available³ as has the data required to adjust the model's codec, if desired.

There are some viable ideas to improve the recognition accuracy even further. First of all, the incorporation of even deeper networks comes to mind as we are dealing with tons of training data. First experiments on the UW3 data set, comprising almost 100,000 GT lines, showed positive effects when utilizing deeper network structures. Another idea is the usage of data augmentation. On a first look this does not seem like a particularly promising approach, as it is standing to reason that augmented data comes with similar problems as synthetic data, when a lot of real GT is available and already leads to very performant models. Because of that, augmenting the pretraining data, or even all of the available real Fraktur data, cannot necessarily be expected to yield significantly better results. However, the refinement steps might offer an interesting opportunity to incorporate data augmentation by splitting the training in two stages⁴: First, the refinement data, i.e. at most 50 lines per source, is augmented and used to train a single model or a voting ensemble. Then, further refinement can be achieved by only training the real selected lines, just as we did before, but starting from the model(s) obtained during the first stage. Maybe, this way we would be able to reinforce the effect of the refinement step while keeping the robustness obtained during the earlier training steps.

Naturally, adding further training data always represents a valid approach in order to improve the recognition accuracy. While any amount of additional valid real data is always welcomed, the obtained results indicate that, at this stage, the trained models would benefit more from a wide variety of works, and consequently a multiplicity of fonts and typefaces, even if only a few lines are available, than from some selected complete works providing thousands or even tens of thousands of lines.

Furthermore, while ABBYY already has strong postprocessing techniques available, this represents an opportunity to improve the results achieved by Calamari and OCRopus 1 even further, in particular the inclusion of dictionaries and language models.

³<https://github.com/chreul/19th-century-fraktur-OCR>

⁴By now this proposed two-staged procedure has been incorporated into Calamari as the default approach when using data augmentation.

7.2 Automatic Semantic Text Tagging on Historical Lexica by Combining ATR and Typography Classification⁵

Despite the recent progress in the area of historical OCR the thorough indexing of a lexicon remains a challenging task. To fully exploit the content, it does not suffice to just gather the text by performing ATR or by transcribing it, since most lexica contain a complex structure within the text consisting not only of the lemmata themselves, but also definitions, grammatical information, or possible word formations. Usually, to encode this information, printed lexica rely on typographical variations of the text (compare Figure 7.3). Therefore, if the typography within a dictionary article changes, e.g. from Fraktur to Antiqua, it carries semantic meaning.

Gathering this information is not a trivial task, since manual tagging is cumbersome and machine learning approaches like *GROBID dictionaries* [115], while being promising, are usually not flexible enough to smoothly adapt to new material or user requirements.

The approach proposed in this subchapter treats the gathering of the information stored in historical lexica as two separate sequence classification tasks: ATR and typography recognition. To ensure flexibility and adaptability, we rely on the application of an open-source ATR engine which can easily be trained and geared towards recognizing different text and typography classes. Analogously to the ATR, we also produce line-based typography GT by assigning a distinct label to each of the typography classes. After training two separate models (one for text and one for typography) using the respective GT, each model recognizes the line images and the outputs are aligned on word level. Our case study on Daniel Sanders' *Wörterbuch der Deutschen Sprache* showed very promising results leading to low CERs, both for ATR and typography recognition.

The remainder of the subchapter is structured as follows: after discussing related work in Section 7.2.1, we introduce the used material in Section 7.2.2 and describe our approach in Section 7.2.3. The outcome of the experiments performed in Section 7.2.4 are discussed in Section 7.2.5 before Section 7.2.6 concludes the subchapter.

7.2.1 Related Work

In this section we first give an overview over the related work regarding font identification and then briefly discuss the adjacent topic of script identification. Finally, we discuss the differences between our method and previous approaches.

⁵This section is based on a previously published article [231]: C. Reul, S. Göttel, U. Springmann, C. Wick, K.-M. Würzner, and F. Puppe, "Automatic Semantic Text Tagging on Historical Lexica by Combining OCR and Typography Classification: A Case Study on Daniel Sander's *Wörterbuch der Deutschen Sprache*," in *Proceedings of the 3rd International Conference on Digital Access to Textual Cultural Heritage*. ACM, 2019, pp. 33–38. [Online]. Available: <https://doi.org/10.1145/3322905.3322910>

7.2.1.1 Font Identification

Zramdini and Ingold [331] extracted features from the horizontal and vertical projection profiles of text lines and applied a Bayesian classifier to discriminate the font weight, slope, and size. Experiments on 100 French text lines each printed in 48 fonts (three standard font families with four sizes, two slopes, and two weights) showed encouraging results.

In [332] the same authors gathered typographical features while searching for global aspects of the text which allow a human reader to distinguish visually between font weights, slopes, sizes, and typefaces. First, in order to delimit the feature extraction area, CCs are classified into six typographical and six morphological classes using the text lines vertical projection profile. Then, descriptive features like black pixel density, x-height, and the presence of serifs were extracted from the CCs and their bounding rectangles. 280 font models representing ten typefaces, seven sizes, and four styles were trained on about 100 English text lines and then evaluated on a test set consisting of at least 100 French text lines for each font. During the evaluation almost 97% of the fonts were correctly classified as a whole. This number even rose to over 99.9% when only considering the more practical problem of identifying the style of a given typeface.

Other approaches consider font identification as a global texture analysis problem. e.g. Zhu et al. [330] first normalize an input document in order to obtain a uniform block of text. Afterwards, they extract font features by applying multichannel Gabor filters [102] to the text blocks and utilize a simple weighted Euclidian distance classifier for font recognition. Experiments on computer-generated images using 56 (14 typefaces, 4 styles) Chinese and English fonts resulted in an average recognition accuracy of 99.1%. Furthermore, the method was shown to be very robust as the recognition dropped only slightly even when heavily contaminating the text block images with salt and pepper noise.

Nicoulaou et al. [194] also considered font recognition as a texture classification problem. They used Local Binary Patterns [210] extracted from text blocks as features and a simple Nearest Neighbour classifier to distinguish 100 fonts (10 font families and 10 distinct font sizes). For training and evaluation the wide-spread APTI database⁶ consisting of over 100,000 arabic words was used. During evaluation, the Local Binary Pattern approach achieved comparable results to the Gaussian Mixture Models approach [271] which is considered the state-of-the-art in Arabic optical font recognition.

Of course, the topic of font identification has also been addressed by many (deep) neural network based techniques. Tao et al. [291] interpreted the task of classifying the font of single Chinese characters as a sequence classification problem. Therefore, they rearranged the input ordering of the points on a stroke and applied a 2-D LSTM network to it. Additionally, they added a principal component layer for preprocessing purposes. For evaluation they created a synthetic dataset combining seven typefaces with four font styles

⁶<https://diuf.unifr.ch/diva/APTI>

and added different levels of noise. Their system outperformed five other approaches and showed a high robustness against noise due to the added principal component layer.

In [294] Tensmeyer et al. compared the performance of two CNN architectures (AlexNet [149] and ResNet-50 [125]) on an Arabic font dataset containing 40 typefaces in four styles and ten different sizes. While the training of the CNNs is performed on single patches of script the classification works by predicting several patches individually and then averaging the class probability distribution. The ResNet-50 outperformed AlexNet in all evaluation scenarios with the highest accuracy of 99.2% being achieved by training and testing on segmented line images.

The winners of the most recent third edition of the ICDAR competition on multi-font and multi-size digitally represented Arabic text [270] also applied a CNN-based approach. They used ResNet-18 as their base model and only adjusted it by replacing the final layer with fully connected layers which fitted the individual sub tasks of font recognition, font-size recognition, and joint font and font-size recognition.

7.2.1.2 Script Identification

A very comprehensive and up to date overview over the various methods of script identification is given in [303]. First, the authors discuss script detection methods categorized in different domains (e.g. printed and handwritten documents). Then, a vast amount of local and global features is described and the performance of the corresponding systems is evaluated and discussed.

An especially noteworthy approach was recently proposed by Fujii et al. [101] who performed sequence-to-label script identification for multilingual ATR on 232 languages and 20 scripts. Their system consisted of two components which are trained end-to-end: The encoder converts a line image into a feature sequence and the summarizer then aggregates the obtained sequence in order to classify the line. After training their system on over 1.1 million lines evaluation on over 180,000 lines showed a script identification error rate of 3.1%.

Despite our focus on font identification the sequence learning method for script identification proposed by Ul-Hasan et al. [304] represents an important piece of preparatory work for our purposes. The authors used OCRopus 1 and its underlying LSTM implementation [46] to separate English and Greek script. First, they generated close to 100,000 synthetic text lines in a variety of fonts using the OCRopus 1 *linegen* script and freely available English-Greek text. In the process, character-based GT was produced by putting the label 0 for English characters and label 1 for Greek ones. After training an OCRopus 1 model on the GT they were able to not only differentiate English from Greek script but also to identify the corresponding patches in the images by making use of the llocs output by OCRopus 1. The proposed method achieved a CER of 1.81% on an evaluation set consisting of 9,500 synthetically generated text lines.

7.2.1.3 Our Method

The approach proposed in this section differs from all methods described above in one or several of the following aspects:

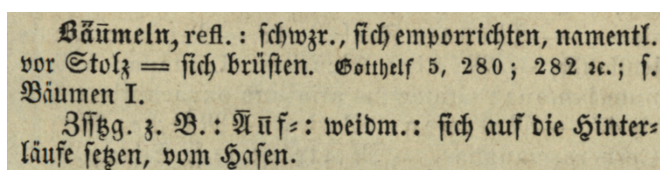
- While many approaches deal with synthetically generated images, we are working on **real data**. We therefore do not only present a theoretical proof-of-concept but give a fully realistic example of our method.
- Due to the age of the material, we are not dealing with modern standard fonts but with **historical non-uniform fonts**.
- The **frequent change of typography even within lines** presents an additional challenge since each word has to be classified by only using a small amount of information.
- In consequence of the usage of an open-source ATR engine, our approach is **easily adaptable** to other works.

7.2.2 Material: Sanders' Dictionary

In this case study we worked with Daniel Sanders' "Wörterbuch der Deutschen Sprache". While we refer to Section 3.1.3 for a general overview over the dictionary, we now focus on the semantic function of typography it comprises (see Figure 7.3) being explained in the following (see also Figure 7.4, 7.5 for the typographical class labels).

As mentioned above, if the typography within a dictionary article changes this represents semantic meaning. Such a syntax within a dictionary article or an entire dictionary is nothing unusual, but it is particularly complex in Sanders' dictionary. The lemmatization is usually based on the elementary words of the German vocabulary. The word family oriented arrangement is a lexicographical challenge, but makes it possible to systematically develop and find the countless derivatives and compounds. The macrostructure of the dictionary is highly dense and therefore complex. The semantic function of typography in Sanders' dictionary is explained in the following using a few examples. Due to the complexity only the most important use-cases can be shown here.

The main lemmata at the beginning of each dictionary-article are always in bold and indented. The corresponding GT representation we are aiming to reproduce with our method is shown below with lemmata color-coded in red (assigned label of typographical class *l*). Their typographic appearance is unique. In this case, as with most dictionary articles, the lemma is just followed by grammatical properties, regarding this example for verbs, such as "tr." (transitive), printed in Antiqua (green, *a*). The semantic paraphrases are annotated as a simple Fraktur type (black, *f*). The typeface of the quotations is divided into two types consisting of the author's name (yellow, *n*) and the page number (again Antiqua). At the end of a reading possible word formations may be listed. The typeface again uses Fraktur but in letter-spacing (blue, *F*) followed by a meaning.



Bäumeln, refl.: schwzr., sich emporrichten, namentl.
vor Stolz = sich brüsten. Gotthelf 5, 280; 282 zc.; f.
Bäumen I.

Zfßtg. z. B.: Äuf-: weidm.: sich auf die Hinter-
läufe setzen, vom Hafen.

```

<entry>
  <orth>Bäumeln</orth>
  <gram>refl.</gram>
  <sense>
    <usg type="geo">schwzr.</usg>
    <def>sich emporrichten, namentlich
      vor Stolz = sich brüsten</def>
  </sense>
  <bibl>
    <author>Gotthelf</author>
    <biblScope>5, 280; 282 etc.</biblScope>
  </bibl>
  <re>
    <orth>Äuf-</orth>
    <usg type="dom">weidm.</usg>
    <def>sich auf die Hinterläufe setzen,
      vom Hafen</def>
  </re>
</entry>

```

Figure 7.3: Input (top) and color-coded output (middle) of our method for an example article depicting the five typography labels. The raw ATR output together with its aligned typography label determines the color code as described in Figure 7.5. This output can then be further transformed into a (reduced) TEI representation (bottom), explicitly encoding not only the lemma (*orth*) but also all additional information about grammatical properties (*gram*), sources (*bibl*), and compositions (*re*). Adopted from [231].

7.2.3 Methods

In this section, we describe our approach in detail. After a brief look at the required preprocessing and segmentation steps, we first discuss the ATR and the typography recognition as separate subtasks and afterwards combine them in order to obtain the final result.

7.2.3.1 Preprocessing and Segmentation

To be able to work with Calamari, some segmentation steps had to be performed. First, the scans are preprocessed by applying the *ocropus-nlbin* script (cf. Section 5.4.2) resulting in a deskewed binary image for each page. Next, the columns have to be detected and segmented into lines. For the first step, we used a rule-based implementation⁷ based on the detection of whitespace separators. Before the line segmentation is performed by applying *ocropous-gpageseg* (cf. Section 5.5.2), the columns are once again deskewed separately. For every extracted line, the coordinates of its enclosing rectangle are stored for later use.

7.2.3.2 Automatic Text Recognition

To maximize the ATR accuracy while keeping the computational effort reasonable, we trained an ensemble of five book-specific models using the voting ensemble for the recognition of 19th century Fraktur trained during the case study described in Section 7.1 as a starting point. The required GT was produced by transcribing a selection of lines which gave a good representation of the occurring characters and typography classes within the book.

During the recognition step, the start and end positions of the recognized characters and the voted probabilities both for the most likely character as well as for the top alternatives are stored.

Due to the complexity of the material at hand, we had to handle a significantly bigger alphabet compared to average 19th century Fraktur printing. 150 distinct characters occurred in the training and evaluation set. For comparison, the mixed models we used as a starting point were trained on GT comprising only 93 characters (cf. Section 7.1.2.3).

7.2.3.3 Typography Recognition

In general, the typography recognition sub task is very similar to the ATR one. The main difference lies in the GT production, as manually assigning abstract typography labels to each character is even more cumbersome and error prone than producing textual GT. Therefore, we first introduce a method that considerably simplifies this task. Additionally, we describe the training and recognition process including data augmentation.

Ground Truth Production Regarding the typography GT, we followed the approach proposed by Ul-Hasan et al. ([304], cf. Section 7.2.1.2) and assigned a distinct label to each of the five classes (*f*, *F*, *n*, *l*, *a*). Since the typography never changes during the course of a word, we developed a GUI (see Figure 7.4) which uses the ATR GT as input and consequently allows for an efficient typography GT production on word level. After

⁷<https://github.com/wrznr/column-detect>

the transcription of the ATR GT, it is first split into words according to the occurring whitespaces. Then, the words are processed from left to right. The currently active word is highlighted within an additional label. Next, the user can assign one of the predefined classes to each of the characters of the active word by performing a single mouse click. After the insertion of a whitespace, the next word of the line becomes active and is processed in the same way. Moreover, it is possible to assign a specific label to the remainder of the line at once. Figure 7.4 demonstrates this process.

To ensure native compatibility to OCRopus 1 and Calamari our labeling procedure works on raw line images and produces textual GT files. Naturally, the input and output could also easily be imported from or exported to other (e.g. XML-based) representations.

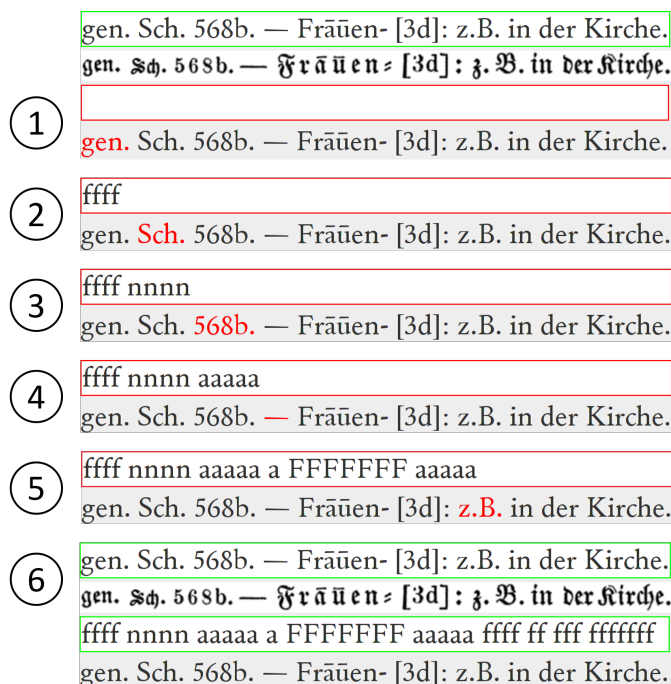


Figure 7.4: Typography GT production process of a single line. The ATR GT (green border) and the line image are shown at the very top. The remaining steps demonstrate the word-wise typography labelling: 1) The first word is highlighted as active and a single mouse click assigns the selected class label (here: *f*) to each letter. 2) The step is repeated for the following words with the to-be-labeled word always being highlighted in red. 3) Final ATR and typography GT result after assigning the *f* label to the last four words at once. Adopted from [231].

Training and Recognition With the obtained typography GT, we performed another training run. In order to add robustness to the models, some data augmentation techniques from the OCRopus 3 *ocrodeg* module [203] were utilized. For every line in the training data set, we created augmented lines by artificially degrading the original image using

random distortion and blurring operations. With the obtained data, we performed a pretraining and refined the resulting models by training on real lines exclusively.

7.2.3.4 Combining the Outputs

Whitespace Pruning To combine the output of the two recognition steps for ATR and typography, we first prune the results by removing leading and trailing whitespaces from the lines. Furthermore, chains of whitespaces are combined into a single one by updating the start and end positions accordingly and setting the confidence to 1.

Alignment Next, we perform an alignment on word level by iterating over the whitespaces recognized by the ATR. For each word in the ATR result, we compute the corresponding word in the typography output by checking the typography labels which got recognized since the last whitespace in the ATR. If the bounds of a label end before the next whitespace it is added to the current word. Otherwise, the label is dropped, the current word is concluded, and the next one is started. Additional whitespaces in the typography are also dropped. Figure 7.5 explains the alignment step and the upcoming typography voting.

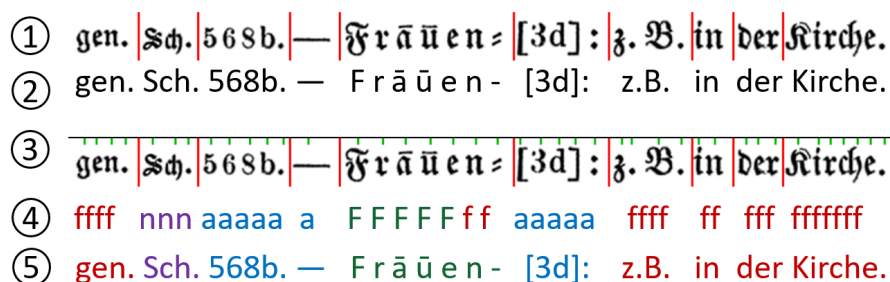


Figure 7.5: Typographic alignment for an example line: 1) Line image with whitespace positions (red). 2) ATR output. 3) Typography output with whitespace and character positions (green). 4) Raw (and slightly flawed) textual typography output. 5) Final combined output with voted typography classes assigned on word level. Adopted from [231].

Typography Voting Since the typography does not change during a word, each word is assigned a unified label by selecting the most likely one from the labels matched to the characters of the word. Therefore, we iterate over all recognized characters and sum up the confidence values for all classes. Finally, the label with the highest confidence sum is assigned to the word as a whole. With respect to our example this approach not only covers up the misclassification of the last two characters in the fifth word “Frauen” but also shows its robustness against the insertion or deletion of characters: Despite the typography model only recognizing three characters of type n in the four letter word “Sch.”, the method provides the correct end result of class n .

Final Output The final output of the recognition process is a JSON file which stores a list of words for each line. Each word is represented by the ATR result and the assigned typography label. Additionally, the minimal character confidence which occurred during the ATR of the word is stored. If needed, this allows for an efficient correction of suspicious semantic key words, e.g. lemmata. Finally, the bounding box of each word is determined from the y -coordinates of its containing line and the x -coordinates of its preceding and subsequent whitespaces. Reversing the cutting and rotational operations during the column segmentation and deskewing steps allows to map the word bounding boxes to the original scan image, e.g. for presentation purposes.

7.2.4 Experiments

The proposed approach is thoroughly evaluated in this section. After a brief explanation of the used data, we first take a look at the ATR performance before evaluating the typography recognition, both with and without the inclusion of data augmentation.

7.2.4.1 Data

To examine the influence of the number of GT lines on the ATR and typography recognition accuracy, we performed the upcoming experiments not only with all available GT (765 lines) but also with subsets of 400, 200, 100, and 50 lines respectively. The lines of each subset were randomly picked from the next bigger set of lines. For evaluation purposes, we selected six columns consisting of 630 lines and created the corresponding GT. Table 7.6 provides additional information about the evaluation data.

Table 7.6: Statistics of the evaluation data including the distribution of the number of words (W) and characters (C) as well as the average word length (L) for the five typography classes.

	a	f	F	n	l	All
W	1,007 17.2%	4,378 74.7%	105 1.8%	314 5.4%	53 0.9%	5,857 100%
C	2,580 9.4%	21,936 80.2%	747 2.7%	1,768 6.5%	333 1.2%	27,364 100%
L	2.56	5.01	7.11	5.63	6.28	4.67

7.2.4.2 Automatic Text Recognition

In this section, we briefly evaluate the ATR performance. Apart from varying the number of training lines, the results are compared to the ones achieved by ABBYY FineReader

7 Case Studies

(cf. Section 2.6.1) and the freely available raw Fraktur 19th century polyfont model we used for pretraining. For evaluation, we apply the built-in Calamari evaluation script which first computes the Levenshtein distance [164] between the ATR result and the GT and then divides it by the number of GT characters in order to obtain the CER. Table 7.7 sums up the results.

Table 7.7: CER of the line-based ATR dependent on the number of lines used for training (*# Lines*). The first row shows the results achieved without book-specific training.

<i># Lines</i>	Calamari	ABBY
-	3.69	10.28
50	1.83	-
100	1.05	-
200	0.67	-
400	0.43	-
765	0.35	-

Unsurprisingly, the CER decreases considerably with a growing number of training lines. It is worth mentioning that, despite the very challenging material at hand, a quite manageable number of 100 lines of GT is already enough to get close to a very low CER of 1%. Furthermore, the familiar saturation effect becomes visible as almost doubling the number of training lines from 400 to 765 only results in a smaller relative reduction in CER compared to previous steps. The mixed model performs reasonably well, especially when considering that over 20% of the errors occur due to different representations of hyphens and can mostly be attributed to diverging transcription guidelines. Yet, these results demonstrate that a book-specific training is reasonable. This is underlined even further by the mediocre recognition accuracy achieved by ABBY, which struggled with the diversity of the alphabet as well as with the peculiarities of the material, differing considerably from other historical sources such as novels, where ABBY’s strong dictionary and language modelling features are much more useful.

7.2.4.3 Typography Recognition

After showing the ATR capabilities of Calamari and the pretrained Fraktur models, the next step is to evaluate the performance of the typography recognition. Therefore, we first train exclusively on real data and investigate the influence of a postprocessing step, i.e. the confidence voting in order to unify the labels within a word. Then, we perform data augmentation to synthetically enrich the training corpus. Analogously to the ATR evaluation, we examine the influence of the number of training lines for every step. We use the Levenshtein distance [164] as a performance measure for the typography classification task. Since the typography does not change within a word it makes sense to utilize the WER: After determining the most likely label by confidence voting we collapse each word to a single character and remove all whitespaces, e.g. a voted output

of “lllllll aaaa ffffffff fff fff” becomes “lafff”. Subsequently, we simply calculate the CER by applying Calamari’s eval script using an analogously preprocessed GT.

Training Exclusively on Real Data For the first typography experiment, we used the same set of lines as for the ATR training. After the recognition we first measured the raw CER and then performed the postprocessing described above to obtain the voted CER and the WER. The results are shown in Table 7.8. There are many interesting things to take away from these results: First, as expected and analogous to the ATR results, a strong negative correlation between the number of lines and the CER can be observed.

Table 7.8: Performance of the line-based typography recognition dependent on the number of GT lines (*# Lines*).

# Lines	Raw	Voted	
	CER	CER	WER
50	8.48	8.53	9.82
100	3.16	3.12	4.08
200	2.46	2.40	2.66
400	1.39	1.38	1.72
765	1.17	1.18	1.47

Interestingly, the postprocessing does not lead to noteworthy improvements which is at first surprising since intuitively one expects a regular occurrence of single wrong characters within an otherwise correct word, which should easily be fixed by the postprocessing, e.g. “fffnff” → “ffffff”. However, in reality the dominating errors are insertions and deletion of single characters which, naturally, cannot be eradicated by our postprocessing.

To come along on top, the postprocessing can even worsen the results because of deleted whitespaces between two words with different labels. For example, two words “aaaa fff” apart from a single error, i.e. the missed whitespace, got correctly recognized as “aaaaffff”, but the confidence voting turned the recognized string into “aaaaaaa” introducing four additional errors.

Concerning the WER the same tendencies can be observed as for the CER. Incorporating all available training lines results in a model which is capable of assigning the correct typography label to over 98.5% of the words.

Incorporating Data Augmentation In this last experiment, we investigate the influence of adding synthetically altered data to our training set. Therefore, we augmented each line 5, 20, and 100 times for a small (50) and medium (200) set of real lines, as well as for the complete set. Table 7.9 sums up the results. Unsurprisingly, the effectiveness of data augmentation highly depends on the number of available real lines. While for 50 and 200 lines significant improvements in recognition accuracy can be observed, only a

7 Case Studies

moderate reduction in WER of 6% can be achieved when using all available real training lines. Apart from the apparent diminishing return another drawback of excessive data augmentation is the runtime of the training. While in our case the most comprehensive training, that is using 765 real lines and augmenting them 100 times, could be completed within a few hours due to the parallel utilization of five suitable GPUs, it would take at least several days to complete this task without GPU support even on a modern PC.

Table 7.9: Effect of data augmentation on the line-based typography recognition. Each cell shows the resulting WER as well as the improvement in percent compared to training exclusively on real lines. For example, augmenting 200 lines 5 times leads to a WER of 1.89 which represents an improvement of 29% over the real-line-only WER (2.66).

# Real Lines	# Augmentations			
	-	5	20	100
50	9.82	6.37	5.34	4.90
	-	35%	46%	50%
200	2.66	1.89	1.67	1.66
	-	29%	37%	38%
765	1.47	1.45	1.43	1.38
	-	1%	3%	6%

7.2.5 Discussion

During our experiments, we showed that typography recognition using the open-source ATR engine Calamari is not only possible but very precise. Naturally, a high quality recognition of several very similar typography classes, just like the ones used in Sanders’ dictionary, can only be achieved by performing a thorough book-specific training. However, the results showed that even a manageable number of GT lines (200) results in an excellent WER of 1.7% when utilizing data augmentation. Due to the proposed approach to semi-automatically produce the typography GT by building from the available ATR GT, the required manual effort compared to a fully manual transcription is considerably reduced as is the susceptibility to errors.

While the ATR was not the main focus of this case study, the achieved recognition results are still worth mentioning, getting very close to 1% CER with a mere 100 training lines and on a particularly demanding material. This also demonstrates the impressive recognition capabilities of Calamari and underlines the usefulness of the established approach of combining pretraining and confidence voting with a combined CNN/LSTM network. The unequal distribution of typography types in the GT, which is typical for historical lexica, caused a problem, as the ATR model overfits the very frequent classes and underfits the rare ones. This problem can be addressed during the GT selection but

only to a certain degree, since lines which contain a less frequent type are still pretty much always dominated by the main types *a* and *f*.

The high quality ATR result combined with the reliable typography based tagging lays the foundation for methods to fully automatically transform comprehensive (in Sanders' case ca. 800,000 text lines) scanned lexica with reasonable effort into a fine-structured output, e.g. TEI (see bottom of Figure 7.3 for an example). Appropriate rule-based algorithms have been developed at the Berlin-Brandenburg Academy of Sciences and Humanities⁸, laying the groundwork not only for a fully searchable and semantically marked up online version of the dictionary⁹ but also for future complex search queries like “find all lemmata which include Goethe as a source”.

7.2.6 Conclusion and Future Work

This subchapter proposed an approach to apply typography recognition to a real world dictionary comprising a particularly complex semantic function of typography. While we only performed a case study on a single lexicon, both the high quality results for ATR and typography tagging as well as the heavily assisted method for producing typography GT imply that our method can serve as a generic workflow together with a specific typographic model geared towards a complete electronic representation of historical lexica.

There are several promising ideas to further improve the proposed method: For example, the errors caused by misrecognized whitespaces could be reduced by introducing a meta learner which exclusively deals with accepting or rejecting whitespaces proposed by the textual and typographical outputs. Moreover, training type-specific ATR models and applying them on word instead of line level would allow the models to specialize on recognizing a single type, and therefore improving the accuracy even further. Naturally, this approach is most effective when being applied to font or even script types which differ considerably. Despite most of the five typography classes occurring in Sanders' dictionary being very similar to each other, first experiments yielded very encouraging results.

It is worth mentioning, that the proposed methodology is not yet part of OCR4all, yet but an integration is definitively planned for the future. While the typography-driven indexing of historical lexica cannot be considered the main area of application of OCR4all by any means, a font or script detection approach could also provide great benefit to the “standard” ATR of historical printings. An obvious example are the Camerarius works (cf. Section 3.1.1.2), not only because of the Greek embeddings but also the frequent use of two quite different standard fonts: Antiqua upright and italics. This topic will be addressed in greater detail during the Conclusion chapter (cf. Chapter 8)

⁸<http://www.bbaw.de/en>

⁹The final dictionary will soon (early 2020) be available at <http://sanders.bbaw.de>.

8 Conclusion

This final chapter concludes the thesis by first summing up the results and their implications before pointing out our goals for the future of OCR4all and our work in the area of (historical) OCR in general.

8.1 Summary

This section sums up the results and their meanings, mainly focusing on the OCR workflow implemented in OCR4all (cf. Chapter 5) as a whole and the respective evaluations (cf. Chapter 6), while also incorporating the results regarding the ATR training and recognition steps obtained during the *Methods* and *Case Studies* chapters (cf. Chapter 4 and Chapter 7, respectively). To begin with, we list the main findings of our experiments (for a more detailed description of the techniques, evaluations, and findings we refer to the respective sections; a concise overview of the results related to OCR4all is given in Table 8.1):

- The experiments with 19th century Fraktur novels (cf. Section 6.3) showed that a fully automated application of OCR4all is not only possible but can be highly precise on material with moderate layouts and if a suitable ATR model is available. OCR4all achieved an average CER of 0.85% compared to ABBYY's 5.3%. It is worth mentioning, that the printing quality and the consequential ATR quality varies considerably, indicated by the high standard deviation of the CER as shown in Table 8.1 and also the fact, that when taking only the best 50% of the books into consideration, the average CER drops to a very low 0.15%.
- When using OCR4all and dealing with challenging early printed books, inexperienced users, on average, had to invest 2.3 minutes per page to perform a precise segmentation and to reach a CER of below 0.5% which highlights the effectiveness of the proposed approach (cf. Section 6.2.2). Experienced users can perform much more efficiently, with 0.7 minutes per page on average, reaching a speedup factor of more than 3. From the obtained results a rule of thumb for the total manual effort necessary to segment a book using a fine-grained semantic distinction and to produce the necessary GT to reach a CER of below 0.5% can be approximated: The estimated effort for an unexperienced user is roughly 150 minutes for the GT production plus 1.1 minutes per page for the segmentation. For experienced users these numbers drop to 57 minutes plus 0.6 minutes per page, respectively.

8 Conclusion

- To reach a CER below 0.5% the iterative training approach yielded significant speedups (factor 1.9) compared to the naive correction of the output of the mixed model (cf. Section 6.2.3).
- On early printed books with partial complicated layout, a basic segmentation approach, which only ensures a sufficient text/non-text separation and a correct reading order, reduced the manual effort required for segmentation considerably by a factor of more than 2.5 (cf. Section 6.2.4).
- Combining the cross fold training and confidence voting procedures discussed in Section 4.1 with the pretraining method introduced in Section 4.2 and applying them to the book-specific training of early printed books, reduced the CER by 53% compared to the baseline, which is training a single model from scratch. Incorporating Active Learning improved the results by another 16% on average (cf. Section 4.3).
- The first case study on 19th century Fraktur script (cf. Section 7.1) showed the transferability of the proposed voting and pretraining techniques from book-specific models to the task of training mixed models. The resulting models significantly outperform existing open-source and commercial engines and models.
- The second case study dealt with the fine-grained semantic text tagging of historical lexica (cf. Section 7.2) and successfully combined ATR and typography recognition.

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

Regarding the two main steps, segmentation and ATR, which require manual intervention, the first one in general seems to show more room for improvement since the ATR of historical printings made great progress over the last few years which could also be observed during our experiments. Having said that, this of course depends heavily on the material at hand and the aspirations of the user. For example, if the goal is to fully automatically process books with trivial layout but written in Cyrillic script, the ATR becomes a problem as no suitable Calamari model is available, yet. Still, Calamari's training and recognition capabilities combined with the easy to use iterative training approach provided by OCR4all allow the users to utilize state-of-the-art deep learning software and accuracy improving techniques like pretraining, voting, and data

Table 8.1: Results (achievable CER and thereto necessary manual effort) obtained when using OCR4all categorized by the type of the *Material*, the experience of the processing *User*, and whether the processing is done fully automatically or in an interactive way. In addition to the mean values the standard deviation is given whenever sensible. The roughly estimated *Total Manual Effort* is composed of the average correction effort necessary to produce enough GT needed to obtain the given CER and the segmentation effort which scales with the number of pages within a book.

Material	Books Printed Before 1600		19 th Century Fraktur Novels
General Processing Approach	interactive		fully automatic
Segmentation	fine-grained semantic distinction		no semantic markup
ATR	book-specific training		application of existing mixed models
Users	unexperienced	experienced	any
Achieved CER (OCR4all)	0.47% ± 0.22%	0.49% ± 0.30%	0.85% ± 1.47%
Achieved CER (ABBYY)	not applicable		5.28% ± 8.13%
Segmentation Effort per Page	1.1 min ± 0.5 min	0.6 min ± 0.2 min	-
Correction Effort per Line	10 s ± 5.2 s	5.5 s ± 2.4 s	-
Total Correction Effort	150 min ± 92 min	57 min ± 42 min	-
Total Manual Effort	150 min+	57 min+	-
	1.1 min · N_{Pages}	0.6 min · N_{Pages}	

augmentation without ever being forced to acquire a deeper understanding of the technical concepts behind them. As shown by the evaluations, CERs below 1% or even 0.5% should almost be considered the norm after a thorough book-specific training was performed. The segmentation using LAREX proved to be intuitive and highly accurate.

While the usage of OCR4all reduces the manual effort necessary to transcribe early printed books tremendously, especially compared to the fully manual approach which often requires several weeks of full-time transcribing to process a single book, we still think that there is room for improvement as detailed in the remainder of this chapter. Regarding the accuracy-improving and time-saving techniques incorporated into the OCR4all workflow, the following guidelines can be derived from our evaluations and experiences:

- Cross fold training and confidence voting should be used whenever feasible from a hardware and time management point of view as it boosts the recognition accuracy considerably regardless of whether dealing with book-specific training or the application of mixed models.
- Relying on existing models and using them for pretraining should be considered almost mandatory, especially if somewhat appropriate (in terms of the codec)

models are available, since it speeds up the training process and results in more robust models.

- Data augmentation is most effective when only a limited amount of GT is available for book-specific training and is therefore highly recommended during the first few iterations of the iterative training approach. Although, since it can slow down the training considerably, an eye has to be kept on time management and hardware aspects.

8.2 Outlook

In this section we first discuss possible features and extensions we would like to incorporate into OCR4all or its submodules before concluding the thesis by reflecting on the general future of OCR4all. To begin with, Table 8.2 provides an overview over planned features and upcoming steps, which we discuss in-detail in the following.

8.2.1 Generic Interfaces for and Extensions of Core Components within the Workflow

Wrapper for the Integration of New Tools The main motto and goal for future developments of OCR4all is to maximize the benefit from sophisticated external open-source OCR solutions provided by the community. Hence, we aim to make the integration of further external tools as easy as possible by providing clean interfaces and various helper routines, for example to automatically generate the settings GUI (cf. Section 5.8.3.2), including the differentiation between general and advanced settings, from a list of given parameters.

Of course, an integration of the developments achieved during the OCR-D project (cf. Section 2.6.7) is an especially promising use-case. Most importantly, the data format has to be adjusted, since OCR-D relies on a METS container for each book which then refers to the individual PageXML files. Naturally, existing databases could be converted to the new format almost trivially, for example by automatically generating a Dummy METS file. However, this would require some changes to the OCR4all core software and LAREX since they currently operate on pure PageXML files. In addition, smaller adaptations have to be performed, for example concerning the passing of parameters.

Further Output and Input Formats In order to allow non-technical users to keep all semantic annotations without having to deal with the peculiarities of the PageXML format, additional alternatives to the two existing output formats are imperative. Since TEI is considered the go-to format for textual markup applications, a comfortable and attractive solution would be to allow the user to simply specify at least some basic mappings between PageXML types and TEI tags, for example defining that the text between two “heading” elements forms a “chapter”, and then let OCR4all export the

Table 8.2: Planned features for OCR4all divided into five groups.

Functionality	Options for Implementation
<i>Generic Interfaces for and Extensions of Core Components within the Workflow</i>	
Wrapper for the Integration of New Tools Extension of the Workflow	adopting OCR-D interfaces full integration of scan preparation, full integration of postcorrection, feedback loop between segmentation and ATR
Automatic Quality Control Further Output and input formats	ATR confidence values TEI, ALTO, PDF, ABBYY XML
<i>Extending the OCR Functionality</i>	
Preprocessing	
<ul style="list-style-type: none"> incorporating external tools improving existing tools new development 	OCR-D tools - -
Segmentation	
<ul style="list-style-type: none"> incorporating external tools improving existing tools new development 	dhSegment, ARU-Net use line segmentation results for region segmentation trainable pixel classifier, rule-based detection of the reading order
ATR	
<ul style="list-style-type: none"> incorporating external tools improving existing tools new development 	Tesseract 4, Kraken font/script-specific ATR, voting blanks (Calamari) -
Postcorrection	
<ul style="list-style-type: none"> incorporating external tools improving existing tools new development 	PoCoTo, OCR-D tools - ATR confidences, book-specific dictionaries
<i>Intelligent Interactive Tools</i>	
Extending the LAREX Functionality	using confidences for interactive postcorrection, AL
<i>Usability</i>	ongoing user-driven refinements, ongoing optimization of guides for user and developers, public repository for GT and models
<i>Miscellaneous</i>	extension towards HTR tasks, upgrade to a full-featured server application including resource management and user administration

result as a valid TEI file. Furthermore, conversions to ALTO or maybe PDF should be considered. There are suitable tools available for each task (cf. Section 2.7.4) which have to be evaluated and at least offer some of the required functionality and could be used as a starting point.

It is worth mentioning that, analogously to the output formats, OCR4all could also profit considerably from the option to build from existing results provided from external software. For example, if an ABBYY output is available it might be sensible to build from the segmentation result and just redo the ATR step (cf. the *CalamABBYY* functionality described in Section 7.1.5). For the main part, this would simply require suitable converters and some adaptations of the core software.

The adoption of the OCR-D interfaces described above would allow the integration of various helpful tools and methods for most steps of the workflow. Most notably this includes solutions for page frame detection, binarization, line segmentation, wrappers for all noteworthy open-source ATR engines, and fully automatic postcorrection techniques.

Extension of Workflow: Full Integration of Scan Preparation and Postcorrection

The scan preparation, including steps like splitting double pages as well as page frame detection (cf. Section 2.2.1.3) and orientation correction (cf. Section 2.2.1.2), which as of now is either skipped or performed externally by using ScanTailor (cf. Section 2.2.2.1), should be integrated directly into the web GUI.

A most desirable issue is the incorporation of a fully automatic postcorrection step, for example using (book-specific) dictionaries or language modelling. Especially the latter could well be implemented directly within the ATR engines. Calamari has taken first steps in that direction. However, especially for (very) early printed books this is not a trivial task due to the lack of consistent spelling rules and the frequent use of abbreviations.

Automatic Quality Estimation An admittedly quite visionary extension would be an automatic quality estimation module that checks the results obtained by a fully automatic workflow and provides hints about possible errors. On the one hand, this would enable the user to identify and purposefully target the main source of error. On the other hand, the system might be able to improve itself by, for example, adjusting certain parameters and performing the respective step again.

Feedback Loop between Segmentation and ATR The automatic quality estimation could well be combined with an equally visionary functionality, namely the integration of a feedback loop between different workflow steps and their respective modules. This could enable the system to use information obtained from later steps in order to optimize preceding steps. The most obvious example would be a feedback loop between the segmentation and the ATR which, among others things, would allow to detect missegmentations on region level, e.g. non-text regions classified as text regions, or line level, e.g. double lines or lines vertically cut in half (cf. Section 5.5.2).

8.2.2 Extending the OCR Functionality

In the following we present possible solutions for the four main steps of the OCR workflow.

8.2.2.1 Preprocessing

Regarding the preprocessing step we plan to fully rely on already existing external solutions and also to use them in their original state. Apart from the already integrated binarization and deskewing functionality (cf. Sections 5.4.2 and 5.4.3), we will focus on OCR-D solutions. Most importantly this includes the OCR-D-CIS package [200], which offers a variety of helpful preprocessing tools applicable not only on page but also on region and line level, but also the OCR-D compliant bundle [201] of the *Olena* platform [86], among others providing additional means for binarization

8.2.2.2 Segmentation

One of the main goals to address is the obvious lack of a more potent segmentation method that can deal with (more) complex layouts in a highly automatized way. While approaches based on deep learning, that mostly aim to assign a layout class label to each individual pixel, carry some promise, they are usually geared towards the training on and application to a single book (cf. Section 2.3.1.3). However, when a basic segmentation (text/non-text separation and maybe a correct reading order) is considered sufficient, a mixed model approach for segmentation similar to the ATR one represents a very promising idea. Unfortunately, as of now we are missing the required GT (page images and pixel labeling) to train these models. Admittedly, there is some useful data more or less openly available, but usually it shows considerably shortcomings in terms of quality and accuracy, or only offers a manageable amount of data from a small number of books and/or a very specific time period. A comprehensive and versatile data set of high quality segmentation GT is missing, especially for early printed books.

The current semi-automatic and precise segmentation approach of OCR4all allows to produce a large amount of GT with manageable effort. During the evaluations for the OCR4all workflow alone, more than 6,000 pages of GT have been created. Naturally, the manual part of the segmentation represents a non-negligible extra effort and expense, yet it also offers a double benefit as it not only leads to higher ATR results for the (humanist) users but it also allows the developers to utilize the new GT to train strong mixed models for the segmentation task. Of course, the next step would be to introduce an iterative training approach for the segmentation task to OCR4all, allowing the users to build from a reasonable result produced by a mixed model and then gear it towards the book at hand. This could be achieved by correcting a few pages (hopefully) with minimal effort and then train a new model which might then even be able to provide a more sophisticated semantic distinction of layout elements if desired by the user and if properly trained.

Of course, depending on the material and the use case, a fully automated approach, that does not require the user to look at every single page, seems to be currently out of reach. That is not only due to the complexity of the layouts but also due to the very high demands of the users regarding the quality of the segmentation and degree of semantic distinction. However, we think that the efficiency of this part of the workflow can be further increased.

Incorporating External Tools Regarding the region segmentation we monitor generic solutions like `dhSegment` (cf. Section 2.2.1.3) but do not expect an existing solution to fit our specific demands in the near future.

The line segmentation, despite mostly performing admirably, also definitely offers a lot of room for improvement, especially because of its overreliance on the scale value derived from the occurring CCs as discussed in Section 5.5.2. Our preferred course of action will most likely be to switch towards a baseline-based methodology, allowing us, for example, to rely on `dhSegment` or solutions developed by the HTR community like `ARU-Net` (cf. Section 2.3.2.3). On the downside, this would require an additional processing step when passing the lines into most ATR engines, which is not a trivial task and might lead to accuracy drops (cf. Section 2.3.2.1).

Improving Existing Tools Especially the segmentation of the *Camerarius* books appeared to be a quite frustrating task since the one-column layout in itself is rather trivial but the repeated manual classification of semantic sub regions can be exhausting. Pushing the labeling part to a later stage in the workflow, and for example perform it after the line segmentation or even after the ATR, seems to be a viable solution, but the realization is not trivial. While it is possible to first OCR the entire text and then add semantic labels later, for example by encoding them using TEI, the fine-grained positional information with regards to the scan would get lost, considerably limiting the options regarding the presentation of the result. Having said that, the region coordinates do not necessarily have to be recorded during the region segmentation step. An alternative could be to first perform a less time consuming text/image separation similar to the basic segmentation approach we evaluated earlier (cf. Section 6.2.4). Next, the user could perform the line segmentation and apply the iterative training approach. Finally, the semantic classification of layout elements would take place at the very end of the workflow by making use of the line coordinates. For example, when dealing with a sub heading with adjacent running text above and below, the user could simply select the line and apply the new type, resulting in three new regions. This could easily be integrated into the manual postcorrection step since, at least in the *Camerarius* use case, the user has to look at all pages and lines anyway. Especially when processing works with similar layout properties like the *Camerarius* books from our experiments, this approach would significantly speed up the segmentation and labeling process without noteworthy affecting the rest of the workflow. In addition, this would allow to incorporate textual information into a (semi-)automatic classification approach. Our evaluations showed that a basic

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

New Developments We aim to include a trainable pixel classifier in order to either provide a valid starting point for other segmentation approaches by classifying pixels and consequently CCs as text, image, and noise or even perform a fine-grained semantic markup [317]. Of course, a more powerful segmentation approach must also comprise a more sophisticated method for the determination of the reading order which also has to be integrated into LAREX. To generate the reading order, the idea is to allow the user to comfortably specify rules based on the detected region types as well as their absolute and relative position.

8.2.2.3 Automatic Text Recognition

Incorporating External Tools Regarding additional ATR engines, the obvious candidates to integrate are Kraken and especially Tesseract 4 since they offer a wide variety of pretrained mixed models, something that Calamari is still lacking. As mentioned above, the integration of the respective OCR-D wrappers would be rather straight forward after adopting the related interfaces.

Improving Existing Tools Regarding training and recognition we want to provide the user with the option to comfortably train several type/font/script-specific models for a single work. This can be very helpful when the book comprises a few, possibly highly different fonts or even scripts, e.g. Latin and Greek. However, the suitable approach and the associated effort highly depends on the specific task at hand: For example, when a suitable semantic distinction has been performed during the region segmentation step, training a book-specific model for running text and one specialized on paratext is almost a trivial task, since training and recognition could always operate on entire lines. However, if the font (cf. the case study dealing with the Sanders' lexicon described in Section 7.2) or script (cf. the Camerarius use case Section 3.1.1.2) changes within a line, the task becomes significantly harder since a two-step approach has to be applied. Still, a relatively straight forward implementation using just a single ATR engine is possible as successfully demonstrated by Kraken: After applying a model trained to recognize various scripts and storing the coordinates as well as the assigned script label of the line sections, a previously assigned ATR model is applied to the respective sections before combining the outputs back to line level. This approach could be transferred to Calamari, and most likely also to other ATR engines, with manageable effort. Naturally, the described solution is perfectly capable and ideally suited to deal with the two related

8 Conclusion

task treated in this thesis: typography recognition in dictionaries and script detection in Camerarius' works. However, when aiming for a fully generic solution, that allows a free combination of engines and models for both steps, the complexity of the task of course rises considerably.

Regarding the confidence-based voting functionality introduced in Section 4.1, the incorporation of blank labels and their respective confidence values represent an apparent improvement. As of now, blanks are ignored during the alignment process, leading to forced simplifications during the alignment process of different ATR results as described in Section 4.1.2.6. Consequently, the voting algorithm misses valuable information. For example, the confidence of an alternative, that was recognized at a position, where a blank represents the top choice of the recognizer, naturally is an important indicator for a possible deletion of a character. Analogously, a high blank probability at a position where the top choice is a non-blank character, could indicate a possible insertion. Apparently, this information could not only be used to improve the effectiveness of the voting procedure but would also be very helpful for the confidence-based interactive postcorrection functionality discussed above. Unfortunately, incorporating this feature is far from trivial, due to the particularities of blanks within the CTC decoding compared to "normal" characters.

New Developments Due to the vast progress regarding ATR engines over the last couple of years, new developments in this area are currently not a priority.

8.2.2.4 Postcorrection

Incorporating External Tools As described in Section 2.5.2 the interactive postcorrection tool PoCoTo represents an interesting option for the integration into OCR4all. Its drawback is that it requires word coordinates for its visualizations. Consequently, this output would have to be enabled in the respective ATR engines, by interpreting the LSTM coordinates of the whitepaces, which should for the most part be a straight forward task.

In addition, we are closely monitoring the progress made in OCR-D related tools like *KerasLM* [140] (character-based language modelling using the *Viterbi* algorithm [97]) and *Cor-ASV-ANN* [76] (character-based sequence-to-sequence-LSTM relying on an encoder-attention-decoder schema and an A^* beam search [123, 171]).

New Developments Apart from the interactive postcorrection we also plan to work on a fully automatic solution which, among others, relies on the intrinsic confidences provided by ATR engines and on dictionaries. Due to the highly variant historical spelling these dictionaries might have to be generated, extended, and altered based on the current book at hand, which would probably require manual intervention by the user.

8.2.3 Intelligent Interactive Tools

Other than the core software, LAREX currently contains all of the interactive functionality. Consequently, we will focus on extending the LAREX functionality to improve in this aspect.

Apart from the voting, there are several other use cases in which we want to profit from the character confidences provided by Calamari and possibly other ATR engines: First, the correction process can be supported by highlighting suspicious characters. Second, by averaging the confidence values over several lines it is possible to identify segments or pages which contain a worse recognition result compared to the rest of the book. This could help to identify text parts that suffer from an increased amount of degradation, contain segmentation errors, use a different type of font, etc. Third, the average confidence calculated over a representative number of recognized lines can serve as a form of quality estimation. We know that the confidence values correlate with the recognition rate and that neural networks tend to overestimate their performance [111, 120]. Therefore, we hope that it is possible to use a lot of existing measurements to derive a model which is able to estimate the true recognition accuracy based on the average confidence [279]. In addition to the automatic selection of the best fitting model for given data, this would be particularly helpful when the goal of a ATR process is to reach a certain recognition quality (for example 2% CER are considered to be sufficient for most NLP tasks) and it is unclear whether the output of a mixed model suffices or if book-specific training is required.

Furthermore, in order to train more robust models, a more flexible selection of lines for training, recognition, and correction is desirable as this allows to train models using GT that is widely spread over the course of the entire book. This would help to further optimize the iterative training approach by integrating AL, i.e. adding lines to the existing GT pool that the current models had problems recognizing, instead of random ones. In Section 4.3.3.2 we showed the effectiveness of this approach by purposefully adding lines to the training set, that showed the largest disagreement between the separate outputs of the voters. It stands to reason that the confidence values returned by the ATR engine would also be a valuable indicator for suitable training lines. Since the aforementioned confidence-based interactive postcorrecting functionality could easily allow to sort lines according to their average recognition confidence, the integration of an AL component would be straight forward.

8.2.4 Usability

Of course, a probably never-ending task will be the constant refinement of OCR4all and its related components in an user-driven fashion, i.e. dealing with bug reports and feature requests. Furthermore, a key aspect remains the optimization of the teaching material associated with the tool. In the future we want to build from the already existing written guides not only by adding screencasts or even tutorial videos but by setting up

8 Conclusion

a knowledge base, most likely in the form of a Semantic MediaWiki (SMW) [151], that not only contains the guides mentioned above in a more modular form but also extends them with and crosslinks them to the theoretical concepts behind each individual step of the OCR4all workflow. Combined with a public repository for GT and models, best practices as well as an assembly of frequently occurring difficulties and proven ways to successfully deal with them, the described SMW would provide the community with a place to share material, knowledge, problems, and solutions.

8.2.5 Miscellaneous

A particularly interesting and challenging goal is to overcome the additional difficulties of HTR. Despite the general workflow being very similar, there are several steps that might require adaptations. For example, handwritten text often does not consist of single characters, which the current line segmentation approach heavily relies on, but features cursive handwriting. Yet, there are already a few pertinent open-source algorithms available (see for example the ARU-Net approach proposed by [118] and described in Section 2.3.2.3), which can at least serve as a valid starting point. Actually, OCR4all has already been successfully applied to a Greek manuscript (Aëtius Amidenus - Libri medicinales, 16th century), achieving character recognition rates in the mid nineties when using only a few hundred lines of GT. However, to examine the capabilities and shortcomings of the current workflow and its components in greater detail, thorough evaluations using comprehensive corpora are required.

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

8.3 Concluding Remarks

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

Despite these comprehensive plans for the future, we already reached our main goal of creating a tool which provides non-technical users with access to a powerful and easy to

8.3 Concluding Remarks

use OCR workflow. This is not only shown by the evaluations but also by the successful application in numerous real-world projects where OCR4all leads to significant speedups of the OCR of our precious cultural heritage.

Bibliography

- [1] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard *et al.*, “Tensorflow: A system for large-scale machine learning,” in *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, 2016, pp. 265–283.
- [2] “ABBYY Technology Portal - Dictionaries and OCR,” https://abbyy.technology/en/features:ocr:dictionary_support, accessed: 2019-12-03.
- [3] “ABBYY Finereader for Gothic/Fraktur recognition,” https://abbyy.technology/en/features:ocr:old_font_recognition, accessed: 2019-11-13.
- [4] “ABBYY Technology Portal - Supported OCR Languages,” <https://abbyy.technology/en/products:fre:win:v12:languages>, accessed: 2019-12-03.
- [5] “ABBYY Technology Portal - XML Export,” <https://abbyy.technology/en/features:ocr:xml>, accessed: 2019-12-07.
- [6] M. Z. Afzal, M. Krämer, S. S. Bukhari, M. R. Yousefi, F. Shafait, and T. M. Breuel, “Robust binarization of stereo and monocular document images using percentile filter,” in *International Workshop on Camera-Based Document Analysis and Recognition*. Springer, 2013, pp. 139–149.
- [7] M. Al Azawi, M. Liwicki, and T. M. Breuel, “Combination of multiple aligned recognition outputs using WFST and LSTM,” in *Document Analysis and Recognition (ICDAR), 2015 13th Int. Conf. on*. IEEE, 2015, pp. 31–35.
- [8] M. Alberti, V. Pondenkandath, M. Würsch, R. Ingold, and M. Liwicki, “DeepDIVA: a highly-functional python framework for reproducible experiments,” in *2018 16th International Conference on Frontiers in Handwriting Recognition (ICFHR)*. IEEE, 2018, pp. 423–428.
- [9] M. Alberti, M. Seuret, V. Pondenkandath, R. Ingold, and M. Liwicki, “Historical document image segmentation with lda-initialized deep neural networks,” in *Proceedings of the 4th International Workshop on Historical Document Imaging and Processing*. ACM, 2017, pp. 95–100.
- [10] M. Alberti, L. Vögtlin, V. Pondenkandath, M. Seuret, R. Ingold, and M. Liwicki, “Labeling, Cutting, Grouping: an Efficient Text Line Segmentation Method for Medieval Manuscripts,” *arXiv preprint arXiv:1906.11894*, 2019.

Bibliography

- [11] C. Allauzen, M. Riley, J. Schalkwyk, W. Skut, and M. Mohri, “OpenFst: A general and efficient weighted finite-state transducer library,” in *International Conference on Implementation and Application of Automata*. Springer, 2007, pp. 11–23.
- [12] “Analyzed Layout and Text Object (ALTO),” <https://www.loc.gov/standards/alto>, accessed: 2019-10-06.
- [13] “Analyzed Layout and Text Object (ALTO) Documentation,” <https://github.com/altoxml/documentation>, accessed: 2019-12-07.
- [14] “ALTO XML GitHub repository,” <https://github.com/altoxml>, accessed: 2019-10-12.
- [15] A. Antonacopoulos, D. Bridson, C. Papadopoulos, and S. Pletschacher, “A realistic dataset for performance evaluation of document layout analysis,” in *2009 10th International Conference on Document Analysis and Recognition*. IEEE, 2009, pp. 296–300.
- [16] A. Antonacopoulos, C. Clausner, C. Papadopoulos, and S. Pletschacher, “Historical document layout analysis competition,” in *2011 International Conference on Document Analysis and Recognition*. IEEE, 2011, pp. 1516–1520.
- [17] —, “ICDAR 2013 competition on historical book recognition (HBR 2013),” in *2013 12th International Conference on Document Analysis and Recognition*. IEEE, 2013, pp. 1459–1463.
- [18] —, “Icdar 2013 competition on historical newspaper layout analysis (hnla 2013),” in *2013 12th International Conference on Document Analysis and Recognition*. IEEE, 2013, pp. 1454–1458.
- [19] A. Antonacopoulos, S. Pletschacher, D. Bridson, and C. Papadopoulos, “ICDAR 2009 page segmentation competition,” in *2009 10th International Conference on Document Analysis and Recognition*. IEEE, 2009, pp. 1370–1374.
- [20] J. Artsimovich, “ScanTailor GitHub repository,” <https://github.com/scantailor/scantailor>, accessed: 2019-11-16.
- [21] —, “ScanTailor Homepage,” <https://scantailor.org>, accessed: 2019-12-12.
- [22] “ARU-Net GitHub repository,” <https://github.com/TobiasGruening/ARU-Net>, accessed: 2019-10-27.
- [23] N. Arvanitopoulos and S. Süssstrunk, “Seam carving for text line extraction on color and grayscale historical manuscripts,” in *2014 14th International Conference on Frontiers in Handwriting Recognition*. IEEE, 2014, pp. 726–731.
- [24] A. Asi, R. Saabni, and J. El-Sana, “Text line segmentation for gray scale historical document images,” in *Proceedings of the 2011 workshop on historical document imaging and processing*. ACM, 2011, pp. 120–126.

- [25] F. Aurenhammer, “Voronoi diagrams—a survey of a fundamental geometric data structure,” *ACM Computing Surveys (CSUR)*, vol. 23, no. 3, pp. 345–405, 1991.
- [26] S. Avidan and A. Shamir, “Seam carving for content-aware image resizing,” in *ACM Transactions on graphics (TOG)*, vol. 26, no. 3. ACM, 2007, p. 10.
- [27] T. Baier, J. Hamm, and U. Schlegelmilch, “Opera Camerarii project at the University of Würzburg,” <http://www.camerarius.de>, accessed: 2019-12-12.
- [28] K. Baierer, R. Dong, and C. Neudecker, “okralact-a multi-engine Open Source OCR training system,” in *Proceedings of the 5th International Workshop on Historical Document Imaging and Processing*. ACM, 2019, pp. 25–30.
- [29] H. S. Baird, “Background structure in document images,” *International Journal of Pattern Recognition and Artificial Intelligence*, vol. 8, no. 05, pp. 1013–1030, 1994.
- [30] J. Baiter, “Archiscribe Corpus GitHub repository,” <https://github.com/jbaiter/archiscribe-corpus>, accessed: 2019-11-03.
- [31] —, “Archiscribe GitHub repository,” <https://github.com/jbaiter/archiscribe>, accessed: 2019-11-03.
- [32] H. Balk and A. Conteh, “IMPACT: centre of competence in text digitisation,” in *Proceedings of the 2011 Workshop on Historical Document Imaging and Processing*. ACM, 2011, pp. 155–160.
- [33] A. Belaid, Y. Rangoni, and I. Falk, “XML data representation in document image analysis,” in *Ninth International Conference on Document Analysis and Recognition (ICDAR 2007)*, vol. 1. IEEE, 2007, pp. 78–82.
- [34] A. L. Berger, V. J. D. Pietra, and S. A. D. Pietra, “A maximum entropy approach to natural language processing,” *Computational linguistics*, vol. 22, no. 1, pp. 39–71, 1996.
- [35] D. M. Blei, A. Y. Ng, and M. I. Jordan, “Latent dirichlet allocation,” *Journal of machine Learning research*, vol. 3, no. Jan, pp. 993–1022, 2003.
- [36] D. S. Bloomberg, G. E. Kopec, and L. Dasari, “Measuring document image skew and orientation,” in *Document Recognition II*, vol. 2422. International Society for Optics and Photonics, 1995, pp. 302–316.
- [37] T. Bluche, B. Moysset, and C. Kermorvant, “Automatic line segmentation and ground-truth alignment of handwritten documents,” in *2014 14th International Conference on Frontiers in Handwriting Recognition*. IEEE, 2014, pp. 667–672.
- [38] M. Boenig, K. Baierer, V. Hartmann, M. Federbusch, and C. Neudecker, “Labelling OCR Ground Truth for Usage in Repositories,” *Proceedings of the 3rd International Conference on Digital Access to Textual Cultural Heritage*, 2019.

Bibliography

- [39] G. Borgefors, “Distance transformations in digital images,” *Computer vision, graphics, and image processing*, vol. 34, no. 3, pp. 344–371, 1986.
- [40] V. Bosch, A. H. Toselli, and E. Vidal, “Statistical text line analysis in handwritten documents,” in *2012 International Conference on Frontiers in Handwriting Recognition*. IEEE, 2012, pp. 201–206.
- [41] —, “Semiautomatic text baseline detection in large historical handwritten documents,” in *2014 14th International Conference on Frontiers in Handwriting Recognition*. IEEE, 2014, pp. 690–695.
- [42] F. Boschetti, M. Romanello, A. Babeu, D. Bamman, and G. Crane, “Improving OCR accuracy for classical critical editions,” *Research and Advanced Technology for Digital Libraries*, pp. 156–167, 2009.
- [43] B. E. Boser, I. M. Guyon, and V. N. Vapnik, “A training algorithm for optimal margin classifiers,” in *Proceedings of the fifth annual workshop on Computational learning theory*. ACM, 1992, pp. 144–152.
- [44] G. Bradski and A. Kaehler, *Learning OpenCV: Computer vision with the OpenCV library*. O’Reilly Media, Inc., 2008.
- [45] T. Bray, “The javascript object notation (json) data interchange format,” 2014.
- [46] T. M. Breuel, A. Ul-Hasan, M. A. Al-Azawi, and F. Shafait, “High-Performance OCR for Printed English and Fraktur Using LSTM Networks,” *12th International Conference on Document Analysis and Recognition*, pp. 683–687, 2013.
- [47] T. Breuel, “Recent progress on the OCRopus OCR system,” in *Proceedings of the International Workshop on Multilingual OCR*. ACM, 2009, p. 2.
- [48] T. M. Breuel, “OCRopus English Default Model,” <http://www.tmbdev.net/en-default.pyrnn.gz>, accessed: 2019-10-15.
- [49] —, “OCRopus Fraktur Model,” <http://tmbdev.net/ocropy/fraktur.pyrnn.gz>, accessed: 2019-10-15.
- [50] —, “A practical, globally optimal algorithm for geometric matching under uncertainty,” *Electronic Notes in Theoretical Computer Science*, vol. 46, pp. 188–202, 2001.
- [51] —, “Robust least-square-baseline finding using a branch and bound algorithm,” in *Document Recognition and Retrieval IX*, vol. 4670. International Society for Optics and Photonics, 2001, pp. 20–28.
- [52] —, “Representations and metrics for off-line handwriting segmentation,” in *Proceedings Eighth International Workshop on Frontiers in Handwriting Recognition*. IEEE, 2002, pp. 428–433.

- [53] —, “Two geometric algorithms for layout analysis,” in *International workshop on document analysis systems*. Springer, 2002, pp. 188–199.
- [54] —, “Implementation techniques for geometric branch-and-bound matching methods,” *Computer Vision and Image Understanding*, vol. 90, no. 3, pp. 258–294, 2003.
- [55] —, “The hOCR microformat for OCR workflow and results,” in *Ninth International Conference on Document Analysis and Recognition (ICDAR 2007)*, vol. 2. IEEE, 2007, pp. 1063–1067.
- [56] —, “The OCRopus open source OCR system,” in *Document Recognition and Retrieval XV*, vol. 6815. International Society for Optics and Photonics, 2008, p. 68150F.
- [57] —, “Benchmarking of LSTM networks,” *arXiv preprint arXiv:1508.02774*, 2015.
- [58] —, “High Performance Text Recognition Using a Hybrid Convolutional-LSTM Implementation,” in *2017 14th IAPR International Conference on Document Analysis and Recognition (ICDAR)*. IEEE, 2017, pp. 11–16.
- [59] J. S. Bridle, “Probabilistic interpretation of feedforward classification network outputs, with relationships to statistical pattern recognition,” in *Neurocomputing*. Springer, 1990, pp. 227–236.
- [60] B. Burrichter and J. Hamm, “Narragonien digital project at the University of Würzburg,” <http://kallimachos.de/kallimachos/index.php/Narragonien>, accessed: 2019-11-07.
- [61] J. Burrows, “‘Delta’: a measure of stylistic difference and a guide to likely authorship,” *Literary and linguistic computing*, vol. 17, no. 3, pp. 267–287, 2002.
- [62] A. Büttner, “Nashi – an Efficient Tool for the OCR-Aided Transcription of Printed Texts,” *Preprints preprints:2019090062*, 2019. [Online]. Available: <https://www.preprints.org/manuscript/201909.0062/v1>
- [63] “Calamari GitHub repository,” <https://github.com/Calamari-OCR/calamari>, accessed: 2019-11-02.
- [64] “Calamari models GitHub repository,” https://github.com/Calamari-OCR/calamari_models, accessed: 2019-11-02.
- [65] R. G. Casey and E. Lecolinet, “A survey of methods and strategies in character segmentation,” *IEEE transactions on pattern analysis and machine intelligence*, vol. 18, no. 7, pp. 690–706, 1996.
- [66] R. Cattoni, T. Coianiz, S. Messelodi, and C. M. Modena, “Geometric layout analysis techniques for document image understanding: a review,” *ITC-irst Technical Report*, vol. 9703, no. 09, 1998.

Bibliography

- [67] “ICDAR 2017 Competition on Baseline Detection in Archival Documents (cBAD) Dataset,” <https://zenodo.org/record/835441#.XdEtGFdKhPY>, accessed: 2019-11-17.
- [68] K. Chen, M. Seuret, J. Hennebert, and R. Ingold, “Convolutional neural networks for page segmentation of historical document images,” in *2017 14th IAPR International Conference on Document Analysis and Recognition (ICDAR)*, vol. 1. IEEE, 2017, pp. 965–970.
- [69] K. Chen, M. Seuret, M. Liwicki, J. Hennebert, and R. Ingold, “Page segmentation of historical document images with convolutional autoencoders,” in *2015 13th International Conference on Document Analysis and Recognition (ICDAR)*. IEEE, 2015, pp. 1011–1015.
- [70] K. Chen, M. Seuret, H. Wei, M. Liwicki, J. Hennebert, and R. Ingold, “Ground truth model, tool, and dataset for layout analysis of historical documents,” in *Document Recognition and Retrieval XXII*, vol. 9402. International Society for Optics and Photonics, 2015, p. 940204.
- [71] M. Christy, A. Gupta, E. Grumbach, L. Mandell, R. Furuta, and R. Gutierrez-Osuna, “Mass digitization of early modern texts with optical character recognition,” *Journal on Computing and Cultural Heritage (JOCCH)*, vol. 11, no. 1, p. 6, 2018.
- [72] L. Cinque, S. Levisardi, L. Lombardi, and S. Tanimoto, “Segmentation of page images having artifacts of photocopying and scanning,” *Pattern Recognition*, vol. 35, no. 5, pp. 1167–1177, 2002.
- [73] “CIS OCR Testset,” <https://github.com/cisocrgroup/Resources/tree/master/ocrtestset>, accessed: 2019-11-16.
- [74] C. Clausner, S. Pletschacher, and A. Antonacopoulos, “Aletheia—an advanced document layout and text ground-truthing system for production environments,” in *2011 International Conference on Document Analysis and Recognition*. IEEE, 2011, pp. 48–52.
- [75] —, “Scenario driven in-depth performance evaluation of document layout analysis methods,” in *2011 International Conference on Document Analysis and Recognition*. IEEE, 2011, pp. 1404–1408.
- [76] “Cor-ASV-ANN GitHub repository,” <https://github.com/ASVLeipzig/cor-asv-ann>, accessed: 2019-12-05.
- [77] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, “Imagenet: A large-scale hierarchical image database,” in *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 2009, pp. 248–255.

- [78] A. Dengel, R. Hoch, F. Hönes, T. Jäger, M. Malburg, and A. Weigel, “Techniques for improving OCR results,” in *Handbook of Character Recognition and Document Image Analysis*. World Scientific, 1997, pp. 227–258.
- [79] B.-B. A. der Wissenschaften, “Deutsches Textarchiv,” *Referenzkorpus der Neuhochdeutschen Sprache*, 2007.
- [80] S. Dey, B. Mitra, J. Mukhopadhyay, and S. Sural, “A Comparative Study of Margin Noise Removal Algorithms on MarNR: A Margin Noise Dataset of Document Images,” in *2017 14th IAPR International Conference on Document Analysis and Recognition (ICDAR)*, vol. 4. IEEE, 2017, pp. 35–39.
- [81] “German Research Foundation (Deutsche Forschungsgemeinschaft, DFG),” <https://www.dfg.de>, accessed: 2019-10-11.
- [82] “DFG-Praxisregeln „Digitalisierung “(2016),” https://www.dfg.de/formulare/12_151/12_151_de.pdf, accessed: 2019-10-11.
- [83] “dhSegment GitHub repository,” <https://github.com/dhlab-epfl/dhSegment>, accessed: 2019-10-27.
- [84] M. Diem, F. Kleber, S. Fiel, T. Grüning, and B. Gatos, “cbad: Icdar2017 competition on baseline detection,” in *2017 14th IAPR International Conference on Document Analysis and Recognition (ICDAR)*, vol. 1. IEEE, 2017, pp. 1355–1360.
- [85] D. Doermann and K. Tombre, Eds., *Handbook of Document Image Processing and Recognition*, ser. Springer Reference. London: Springer, 2014.
- [86] A. Duret-Lutz, “Olena: a component-based platform for image processing, mixing generic, generative and OO programming,” in *symposium on Generative and Component-Based Software Engineering, Young Researchers Workshop*, vol. 10. Citeseer, 2000.
- [87] “eMOP - early Modern OCR Project,” <http://emop.tamu.edu>, accessed: 2019-10-05.
- [88] T. Englmeier, F. Fink, and K. U. Schulz, “A-I-PoCoTo - Combining Automated and Interactive OCR Postcorrection,” *Proceedings of the 3rd International Conference on Digital Access to Textual Cultural Heritage*, 2019.
- [89] “Reference corpus Early New High German,” <https://www.linguistics.ruhr-uni-bochum.de/ref>, accessed: 2019-11-07.
- [90] S. Eskenazi, P. Gomez-Krämer, and J.-M. Ogier, “A comprehensive survey of mostly textual document segmentation algorithms since 2008,” *Pattern Recognition*, vol. 64, pp. 1–14, 2017.
- [91] K.-C. Fan, Y.-K. Wang, and T.-R. Lay, “Marginal noise removal of document images,” *Pattern Recognition*, vol. 35, no. 11, pp. 2593–2611, 2002.

Bibliography

- [92] M. Federbusch and C. Polzin, “Volltext via OCR- Möglichkeiten und Grenzen,” *Testszzenarien zu den Funeralschriften der Staatsbibliothek zu Berlin-Preußischer Kulturbesitz. Berlin: Staatsbibliothek zu Berlin*, 2013.
- [93] M. Feldbach and K. D. Tonnie, “Line detection and segmentation in historical church registers,” in *Proceedings of Sixth International Conference on Document Analysis and Recognition*. IEEE, 2001, pp. 743–747.
- [94] F. Fink, K. U. Schulz, and U. Springmann, “Profiling of OCR’ed Historical Texts Revisited,” in *Proceedings of the 2nd International Conference on Digital Access to Textual Cultural Heritage*. ACM, 2017, pp. 61–66.
- [95] A. Fischer, M. Wüthrich, M. Liwicki, V. Frinken, H. Bunke, G. Viehhauser, and M. Stolz, “Automatic transcription of handwritten medieval documents,” in *15th International Conference on Virtual Systems and Multimedia, 2009 (VSMM’09)*. IEEE, 2009, pp. 137–142. [Online]. Available: <http://dx.doi.org/10.1109/VSMM.2009.26>
- [96] L. A. Fletcher and R. Kasturi, “A robust algorithm for text string separation from mixed text/graphics images,” *IEEE transactions on pattern analysis and machine intelligence*, vol. 10, no. 6, pp. 910–918, 1988.
- [97] G. D. Forney, “The viterbi algorithm,” *Proceedings of the IEEE*, vol. 61, no. 3, pp. 268–278, 1973.
- [98] W. N. Francis and H. Kučera, *Manual of information to accompany a standard corpus of present-day edited American English, for use with digital computers*. Brown University, Department of Linguistics, 1989.
- [99] “Franken+ Homepage,” <https://emop.tamu.edu/outcomes/Franken-Plus>, accessed: 2019-12-12.
- [100] “Fred’s ImageMagick Scripts,” <http://www.fmwconcepts.com/imagemagick/index.php>, accessed: 2019-11-16.
- [101] Y. Fujii, K. Driesen, J. Baccash, A. Hurst, and A. C. Popat, “Sequence-to-Label Script Identification for Multilingual OCR,” in *Document Analysis and Recognition (ICDAR), 2017 14th IAPR International Conference on*, vol. 1. IEEE, 2017, pp. 161–168.
- [102] D. Gabor, “Theory of communication. part 1: The analysis of information,” *Journal of the Institution of Electrical Engineers-Part III: Radio and Communication Engineering*, vol. 93, no. 26, pp. 429–441, 1946.
- [103] A. Garz, A. Fischer, R. Sablatnig, and H. Bunke, “Binarization-free text line segmentation for historical documents based on interest point clustering,” in *2012 10th IAPR International Workshop on Document Analysis Systems*. IEEE, 2012, pp. 95–99.

- [104] A. Garz, M. Seuret, A. Fischer, and R. Ingold, “A User-Centered Segmentation Method for Complex Historical Manuscripts Based on Document Graphs,” *IEEE Transactions on Human-Machine Systems*, vol. 47, no. 2, pp. 181–193, 2016.
- [105] A. Garz, M. Seuret, F. Simistira, A. Fischer, and R. Ingold, “Creating ground truth for historical manuscripts with document graphs and scribbling interaction,” in *2016 12th IAPR Workshop on Document Analysis Systems (DAS)*. IEEE, 2016, pp. 126–131.
- [106] “Gesamtkatalog der Wiegendrucke,” <http://www.gesamtkatalogderwiegendrucke.de/GWEN.xhtml>, accessed: 2019-11-16.
- [107] A. Geyken, S. Haaf, B. Jurish, M. Schulz, J. Steinmann, C. Thomas, and F. Wiegand, “Das deutsche textarchiv: Vom historischen korpus zum aktiven archiv,” *Digitale Wissenschaft*, p. 157, 2011.
- [108] X. Glorot and Y. Bengio, “Understanding the difficulty of training deep feedforward neural networks,” in *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, 2010, pp. 249–256.
- [109] “Google Books Digitization Project,” <https://books.google.com>, accessed: 2019-10-05.
- [110] J. Gosling, B. Joy, G. Steele, and G. Bracha, *The Java language specification*. Addison-Wesley Professional, 2000.
- [111] A. Graves, “Practical variational inference for neural networks,” in *Advances in neural information processing systems*, 2011, pp. 2348–2356.
- [112] A. Graves, S. Fernández, F. Gomez, and J. Schmidhuber, “Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks,” in *Proceedings of the 23rd international conference on Machine learning*. ACM, 2006, pp. 369–376.
- [113] “Die Grenzboten: Eine digitalisierte Zeitschrift,” <http://brema.suub.uni-bremen.de/grenzboten>, accessed: 2019-11-07.
- [114] G. Griffin, A. Holub, and P. Perona, “Caltech-256 object category dataset,” 2007.
- [115] “GROBID Dictionaries GitHub repository,” <https://github.com/MedKhem/grobid-dictionaries>, accessed: 2019-12-12.
- [116] T. Grüning, R. Labahn, M. Diem, F. Kleber, and S. Fiel, “Read-bad: A new dataset and evaluation scheme for baseline detection in archival documents,” in *2018 13th IAPR International Workshop on Document Analysis Systems (DAS)*. IEEE, 2018, pp. 351–356.

Bibliography

- [117] T. Grüning, G. Leifert, T. Strauss, and R. Labahn, “A robust and binarization-free approach for text line detection in historical documents,” in *2017 14th IAPR International Conference on Document Analysis and Recognition (ICDAR)*, vol. 1. IEEE, 2017, pp. 236–241.
- [118] T. Grüning, G. Leifert, T. Strauß, J. Michael, and R. Labahn, “A two-stage method for text line detection in historical documents,” *International Journal on Document Analysis and Recognition (IJ DAR)*, vol. 22, no. 3, pp. 285–302, 2019.
- [119] “GT4HistOCR Corpus at Zenodo,” <https://zenodo.org/record/1344131>, accessed: 2019-10-14.
- [120] C. Guo, G. Pleiss, Y. Sun, and K. Q. Weinberger, “On calibration of modern neural networks,” in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, 2017, pp. 1321–1330.
- [121] “Project Gutenberg,” <http://www.gutenberg.org>, accessed: 2019-10-05.
- [122] J. C. Handley, “Improving OCR accuracy through combination: A survey,” in *Systems, Man, and Cybernetics, 1998. 1998 IEEE Int. Conf. on*, vol. 5. IEEE, 1998, pp. 4330–4333.
- [123] P. E. Hart, N. J. Nilsson, and B. Raphael, “A formal basis for the heuristic determination of minimum cost paths,” *IEEE transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.
- [124] “Hathi Trust Digitization Project,” <https://www.hathitrust.org>, accessed: 2019-10-05.
- [125] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.
- [126] G. Heyer, U. Quasthoff, and T. Wittig, “Text mining: Wissensrohstoff text,” *W3l, Herdecke*, vol. 18, 2006.
- [127] “DIVA-HisDB Dataset,” <https://diuf.unifr.ch/main/hisdoc/diva-hisdb>, accessed: 2019-11-17.
- [128] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.
- [129] P. V. Hough, “Method and means for recognizing complex patterns,” Dec. 18 1962, uS Patent 3,069,654.
- [130] L. ImageMagick Studio, “ImageMagick,” 2008.
- [131] “IMPACT - IMProving ACcess to Text Project,” <http://www.impact-project.eu>, accessed: 2019-10-05.

- [132] “Internet Archive,” <https://archive.org>, accessed: 2019-11-08.
- [133] “Internet Archive - Presentation,” <http://wiredspace.wits.ac.za/bitstream/handle/10539/11513/Kahle%20-%20The%20Internet%20Archive%20PPT.pdf>, accessed: 2019-11-08.
- [134] S. Ioffe, “Batch renormalization: Towards reducing minibatch dependence in batch-normalized models,” in *Advances in neural information processing systems*, 2017, pp. 1945–1953.
- [135] P. Jaccard, “Lois de distribution florale dans la zone alpine,” *Bull Soc Vaudoise Sci Nat*, vol. 38, pp. 69–130, 1902.
- [136] “Java Enterprise Edition,” <https://javaee.github.io>, accessed: 2019-11-23.
- [137] R. Johnson, J. Hoeller, K. Donald, C. Sampaleanu, R. Harrop, T. Risberg, A. Arendsen, D. Davison, D. Kopylenko, M. Pollack *et al.*, “The spring framework–reference documentation,” *interface*, vol. 21, p. 27, 2004.
- [138] P. Kahle, S. Colutto, G. Hackl, and G. Mühlberger, “Transkribus - a service platform for transcription, recognition and retrieval of historical documents,” in *2017 14th IAPR International Conference on Document Analysis and Recognition (ICDAR)*, vol. 04, Nov 2017, pp. 19–24.
- [139] “Kallimachos project at the University of Würzburg,” <http://kallimachos.de>, accessed: 2019-11-07.
- [140] “KerasLM GitHub repository,” https://github.com/OCR-D/ocrd_keraslm, accessed: 2019-12-05.
- [141] D. Keysers, F. Shafait, and T. M. Breuel, “Document image zone classification-a simple high-performance approach,” in *in 2nd Int. Conf. on Computer Vision Theory and Applications*. Citeseer, 2007.
- [142] B. Kiessling, “Kraken - an Universal Text Recognizer for the Humanities,” *DH 2019 Digital Humanities*, 2019.
- [143] B. Kiessling, M. T. Miller, G. Maxim, S. B. Savant *et al.*, “Important New Developments in Arabographic Optical Character Recognition (OCR),” *Al-'USūr al-Wustā*, vol. 25, pp. 1–13, 2017.
- [144] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” *arXiv preprint arXiv:1412.6980*, 2014.
- [145] F. Kirchner, M. Dittrich, P. Beckenbauer, and M. Nöth, “OCR bei Inkunabeln – Offizinspezifischer Ansatz der UB Würzburg,” *ABI Technik*, vol. 36 (3), pp. 178–188, 2016. [Online]. Available: <http://dx.doi.org/10.1515/abitech-2016-0036>

Bibliography

- [146] K. Kise, A. Sato, and M. Iwata, “Segmentation of page images using the area Voronoi diagram,” *Computer Vision and Image Understanding*, vol. 70, no. 3, pp. 370–382, 1998.
- [147] R. Kneser and H. Ney, “Improved backing-off for m-gram language modeling,” in *1995 International Conference on Acoustics, Speech, and Signal Processing*, vol. 1. IEEE, 1995, pp. 181–184.
- [148] “Kraken GitHub repository,” <https://github.com/mittagessen/kraken>, accessed: 2019-12-12.
- [149] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” in *Advances in neural information processing systems*, 2012, pp. 1097–1105.
- [150] A. Krogh and J. Vedelsby, “Neural network ensembles, cross validation, and active learning,” in *Advances in neural information processing systems*, 1995, pp. 231–238.
- [151] M. Krötzsch, D. Vrandečić, and M. Völkel, “Semantic mediawiki,” in *International semantic web conference*. Springer, 2006, pp. 935–942.
- [152] J. B. Kruskal, “On the shortest spanning subtree of a graph and the traveling salesman problem,” *Proceedings of the American Mathematical society*, vol. 7, no. 1, pp. 48–50, 1956.
- [153] K. Kukich, “Techniques for automatically correcting words in text,” *Acm Computing Surveys (CSUR)*, vol. 24, no. 4, pp. 377–439, 1992.
- [154] S. Kullback, *Information theory and statistics*. Courier Corporation, 1997.
- [155] “DIVA-DIA Text Line Segmentation Method for Medieval Manuscripts GitHub repository,” <https://github.com/DIVA-DIA/Text-Line-Segmentation-Method-for-Medieval-Manuscripts>, accessed: 2019-10-27.
- [156] “LAREX GitHub repository,” <https://github.com/OCR4all/LAREX>, accessed: 2019-10-14.
- [157] G. Lazzara, R. Levillain, T. Géraud, Y. Jacquélet, J. Marquegnies, and A. Crépin-Leblond, “The SCRIBO module of the Olena platform: a free software framework for document image analysis,” in *2011 International Conference on Document Analysis and Recognition*. IEEE, 2011, pp. 252–258.
- [158] D. X. Le, G. R. Thoma, and H. Wechsler, “Automated borders detection and adaptive segmentation for binary document images,” in *Proceedings of 13th International Conference on Pattern Recognition*, vol. 3. IEEE, 1996, pp. 737–741.
- [159] F. Le Bourgeois, H. Emptoz, E. Trinh, and J. Duong, “Networking digital document images,” in *Proceedings of Sixth International Conference on Document Analysis and Recognition*. IEEE, 2001, pp. 379–383.

- [160] F. LeBourgeois, “Robust multifont OCR system from gray level images,” in *Proceedings of the Fourth International Conference on Document Analysis and Recognition*, vol. 1. IEEE, 1997, pp. 1–5.
- [161] S.-W. Lee and D.-S. Ryu, “Parameter-free geometric document layout analysis,” *IEEE Transactions on pattern analysis and machine intelligence*, vol. 23, no. 11, pp. 1240–1256, 2001.
- [162] “Leptonica GitHub repository,” <https://github.com/DanBloomberg/leptonica>, accessed: 2019-10-27.
- [163] M. Levandowsky and D. Winter, “Distance between sets,” *Nature*, vol. 234, no. 5323, pp. 34–35, 1971.
- [164] V. I. Levenshtein, “Binary codes capable of correcting deletions, insertions, and reversals,” in *Soviet physics doklady*, vol. 10, no. 8, 1966, pp. 707–710.
- [165] L. Likforman-Sulem and C. Faure, “Extracting text lines in handwritten documents by perceptual grouping,” *Advances in handwriting and drawing: a multidisciplinary approach*, pp. 117–135, 1994.
- [166] L. Likforman-Sulem, A. Hanimyan, and C. Faure, “A Hough based algorithm for extracting text lines in handwritten documents,” in *Proceedings of 3rd International Conference on Document Analysis and Recognition*, vol. 2. IEEE, 1995, pp. 774–777.
- [167] L. Likforman-Sulem, A. Zahour, and B. Taconet, “Text line segmentation of historical documents: a survey,” *International Journal of Document Analysis and Recognition (IJDAR)*, vol. 9, no. 2-4, pp. 123–138, 2007.
- [168] M. Liwicki, H. Bunke, J. A. Pittman, and S. Knerr, “Combining diverse systems for handwritten text line recognition,” *Machine Vision and Applications*, vol. 22, no. 1, pp. 39–51, 2011.
- [169] D. Lopresti and J. Zhou, “Using consensus sequence voting to correct OCR errors,” *Computer Vision and Image Understanding*, vol. 67, no. 1, pp. 39–47, 1997.
- [170] G. Louloudis, B. Gatos, I. Pratikakis, and C. Halatsis, “Text line and word segmentation of handwritten documents,” *Pattern Recognition*, vol. 42, no. 12, pp. 3169–3183, 2009.
- [171] B. T. Lowerre, “The HARP Y speech recognition system,” CARNEGIE-MELLON UNIV PITTSBURGH PA DEPT OF COMPUTER SCIENCE, Tech. Rep., 1976.
- [172] W. B. Lund, D. D. Walker, and E. K. Ringger, “Progressive alignment and discriminative error correction for multiple OCR engines,” in *Document Analysis and Recognition (ICDAR), 2011 Int. Conf. on.* IEEE, 2011, pp. 764–768.

Bibliography

- [173] L. C. Mandell, C. Neudecker, A. Antonacopoulos, E. Grumbach, L. Auvil, M. J. Christy, J. A. Heil, and T. Samuelson, “Navigating the storm: IMPACT, eMOP, and agile steering standards,” *Digital Scholarship in the Humanities*, vol. 32, no. 1, pp. 189–194, 2017.
- [174] D. Mane and U. V. Kulkarni, “A survey on supervised convolutional neural network and its major applications,” *International Journal of Rough Sets and Data Analysis (IJRSDA)*, vol. 4, no. 3, pp. 71–82, 2017.
- [175] R. Manmatha and N. Srimal, “Scale space technique for word segmentation in handwritten documents,” in *International conference on scale-space theories in computer vision*. Springer, 1999, pp. 22–33.
- [176] S. Mao and T. Kanungo, “Empirical performance evaluation methodology and its application to page segmentation algorithms,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 23, no. 3, pp. 242–256, 2001.
- [177] —, “Software architecture of PSET: A page segmentation evaluation toolkit,” *International Journal on Document Analysis and Recognition*, vol. 4, no. 3, pp. 205–217, 2002.
- [178] S. Mao, A. Rosenfeld, and T. Kanungo, “Document structure analysis algorithms: a literature survey,” in *Document Recognition and Retrieval X*, vol. 5010. International Society for Optics and Photonics, 2003, pp. 197–207.
- [179] U.-V. Marti and H. Bunke, “On the influence of vocabulary size and language models in unconstrained handwritten text recognition,” in *Proceedings of Sixth International Conference on Document Analysis and Recognition*. IEEE, 2001, pp. 260–265.
- [180] —, “Using a statistical language model to improve the performance of an HMM-based cursive handwriting recognition system,” in *Hidden Markov models: applications in computer vision*. World Scientific, 2001, pp. 65–90.
- [181] —, “The IAM-database: an English sentence database for offline handwriting recognition,” *International Journal on Document Analysis and Recognition*, vol. 5, no. 1, pp. 39–46, 2002.
- [182] “Materialize - A modern responsive front-end framework based on Material Design,” <https://materializecss.com>, accessed: 2019-11-22.
- [183] D. Merkel, “Docker: lightweight linux containers for consistent development and deployment,” *Linux Journal*, vol. 2014, no. 239, p. 2, 2014.
- [184] J. Michael, R. Labahn, T. Grüning, and J. Zöllner, “Evaluating Sequence-to-Sequence Models for Handwritten Text Recognition,” in *Document Analysis and Recognition (ICDAR), 2019 15th IAPR International Conference on (accepted for)*. IEEE, 2019.

- [185] “Million Book Collection Project,” <http://ulib.isri.cmu.edu>, accessed: 2019-10-05.
- [186] F. Moretti, *Distant reading*. Verso Books, 2013.
- [187] G. Mühlberger, J. Zelger, and D. Sagmeister, “User-driven correction of OCR errors: combining crowdsourcing and information retrieval technology,” in *Proceedings of the First International Conference on Digital Access to Textual Cultural Heritage*. ACM, 2014, pp. 53–56.
- [188] G. Mühlberger, “Transkribus presentation,” <https://www.slideshare.net/ETH-Bibliothek/transkribus-eine-forschungsplattform-fr-die-automatisierte-digitalisierung-erkennung-und-suche-in-historischen-dokumenten>, accessed: 2019-11-10.
- [189] G. Nagy, “Twenty years of document image analysis in PAMI,” *IEEE Transactions on Pattern Analysis & Machine Intelligence*, no. 1, pp. 38–62, 2000.
- [190] G. Nagy, S. Seth, and M. Viswanathan, “A prototype document image analysis system for technical journals,” *Computer*, vol. 25, no. 7, pp. 10–22, 1992.
- [191] “Nashi GitHub repository,” <https://github.com/andbue/nashi>, accessed: 2019-12-12.
- [192] C. Neudecker, K. Baierer, M. Federbusch, K.-M. Würzner, M. Boenig, and V. Hartmann, “OCR-D: An end-to-end open-source OCR framework for historical documents,” *Proceedings of the 3rd International Conference on Digital Access to Textual Cultural Heritage*, 2019.
- [193] W. Niblack, *An introduction to digital image processing*. Strandberg Publishing Company, 1985.
- [194] A. Nicolaou, F. Slimane, V. Maergner, and M. Liwicki, “Local binary patterns for arabic optical font recognition,” in *Document Analysis Systems (DAS), 2014 11th IAPR International Workshop on*. IEEE, 2014, pp. 76–80.
- [195] E. Niggemann, J. De Decker, and M. Lévy, “The new renaissance,” *Report of the ‘comité des sages’. Reflection group on bringing Europe’s cultural heritage online*. Brussels, European Commission, 2011.
- [196] “OCR4all GitHub repository,” <https://github.com/OCR4all>, accessed: 2019-10-06.
- [197] “OCR4all GitHub getting started repository,” https://github.com/OCR4all/getting_started, accessed: 2019-12-12.
- [198] “OCR4all models GitHub repository,” https://github.com/Calamari-OCR/ocr4all_models, accessed: 2019-12-12.
- [199] “OCR Converter GitHub repository,” <https://github.com/cneud/ocr-conversion>, accessed: 2019-12-05.

Bibliography

- [200] “OCR-D CIS GitHub repository,” <https://github.com/wrznr/cis-ocrd-py>, accessed: 2019-12-05.
- [201] “OCR-D Olena GitHub repository,” https://github.com/kba/ocrd_olena, accessed: 2019-12-05.
- [202] “OCR Fileformat GitHub repository,” <https://github.com/UB-Mannheim/ocr-fileformat>, accessed: 2019-12-07.
- [203] “OCROdeg GitHub repository,” <https://github.com/NVlabs/ocrodeg>, accessed: 2019-11-19.
- [204] “OCROpus2 GitHub repository,” <https://github.com/tmbdev/ocropy2>, accessed: 2019-11-15.
- [205] “OCROpus3 GitHub repository,” <https://github.com/NVlabs/ocropus3>, accessed: 2019-11-15.
- [206] “OCROpy GitHub repository,” <https://github.com/tmbdev/ocropy>, accessed: 2019-11-15.
- [207] C. Odebrecht, M. Belz, A. Zeldes, A. Lüdeling, and T. Krause, “RIDGES Herbology: designing a diachronic multi-layer corpus,” *Language Resources and Evaluation*, vol. 51, no. 3, pp. 695–725, 2017.
- [208] C. Odebrecht, M. Belz, A. Zeldes, A. Lüdeling, and T. Krause, “RIDGES Herbology: designing a diachronic multi-layer corpus,” *Language Resources and Evaluation*, vol. 51, no. 3, pp. 695–725, 2017. [Online]. Available: <https://doi.org/10.1007/s10579-016-9374-3>
- [209] L. O’Gorman, “The document spectrum for page layout analysis,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 15, no. 11, pp. 1162–1173, 1993.
- [210] T. Ojala, M. Pietikäinen, and T. Mäenpää, “Multiresolution gray-scale and rotation invariant texture classification with local binary patterns,” *IEEE Transactions on Pattern Analysis & Machine Intelligence*, no. 7, pp. 971–987, 2002.
- [211] S. A. OLIVEIRA and B. SEGUIN, “dhSegment Documentation,” 2019.
- [212] S. A. Oliveira, B. Seguin, and F. Kaplan, “dhSegment: A generic deep-learning approach for document segmentation,” in *2018 16th International Conference on Frontiers in Handwriting Recognition (ICFHR)*. IEEE, 2018, pp. 7–12.
- [213] N. Otsu, “A threshold selection method from gray-level histograms,” *IEEE transactions on systems, man, and cybernetics*, vol. 9, no. 1, pp. 62–66, 1979.
- [214] “PAGE 2 TEI GitHub repository,” <https://github.com/dariok/page2tei>, accessed: 2019-12-07.

- [215] “PRImA PAGE Converter GitHub repository,” <https://github.com/PRImA-Research-Lab/prima-page-converter>, accessed: 2019-12-07.
- [216] “PageXML GitHub repository,” <https://github.com/PRImA-Research-Lab/PAGE-XML>, accessed: 2019-10-11.
- [217] B. Pang, L. Lee *et al.*, “Opinion mining and sentiment analysis,” *Foundations and Trends® in Information Retrieval*, vol. 2, no. 1–2, pp. 1–135, 2008.
- [218] R. Pascanu, T. Mikolov, and Y. Bengio, “On the difficulty of training recurrent neural networks,” in *International conference on machine learning*, 2013, pp. 1310–1318.
- [219] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga *et al.*, “Pytorch: An imperative style, high-performance deep learning library,” in *Advances in Neural Information Processing Systems*, 2019, pp. 8024–8035.
- [220] K. Pearson, “X. on the criterion that a given system of deviations from the probable in the case of a correlated system of variables is such that it can be reasonably supposed to have arisen from random sampling,” *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, vol. 50, no. 302, pp. 157–175, 1900.
- [221] W. Peerawit and A. Kawtrakul, “Marginal noise removal from document images using edge density,” in *Proc. Fourth Information and Computer Eng. Postgraduate Workshop*. Citeseer, 2004.
- [222] I. Phillips, “User’s reference manual for the UW english/technical document image database III,” *UW-III English/Technical Document Image Database Manual*, 1996.
- [223] S. Pletschacher and A. Antonacopoulos, “The PAGE (page analysis and ground-truth elements) format framework,” in *2010 20th International Conference on Pattern Recognition*. IEEE, 2010, pp. 257–260.
- [224] “The PRImA Research Lab of the University of Salford,” <https://www.primaresearch.org>, accessed: 2019-10-11.
- [225] Y. Pu and Z. Shi, “A natural learning algorithm based on Hough transform for text lines extraction in handwritten documents,” in *Advances In Handwriting Recognition*. World Scientific, 1999, pp. 141–150.
- [226] J.-Y. Ramel, S. Busson, and M.-L. Demonet, “AGORA: the interactive document image analysis tool of the BVH project,” in *Second International Conference on Document Image Analysis for Libraries (DIAL’06)*. IEEE, 2006, pp. 11–pp.
- [227] F. Rayar, P. Bourquin, J.-Y. Ramel, R. Jimenes, T. Uetani, S. Breuil, and M.-L. Demonet, “PaRADIIT Project: Main Concepts and Outcomes,” 2014.

Bibliography

- [228] X. Ren and J. Malik, “Learning a classification model for segmentation,” in *null*. IEEE, 2003, p. 10.
- [229] C. Reul, D. Christ, A. Hartelt, N. Balbach, M. Wehner, U. Springmann, C. Wick, C. Grundig, A. Büttner, and F. Puppe, “OCR4all—An Open-Source Tool Providing a (Semi-) Automatic OCR Workflow for Historical Printings,” *Applied Sciences*, vol. 9, no. 22, p. 4853, 2019. [Online]. Available: <https://doi.org/10.3390/app9224853>
- [230] C. Reul, M. Dittrich, and M. Gruner, “Case Study of a Highly Automated Layout Analysis and OCR of an Incunabulum: ‘Der Heiligen Leben’ (1488),” in *Proceedings of the 2Nd Int. Conf. on Digital Access to Textual Cultural Heritage*, ser. DATECH2017. New York, NY, USA: ACM, 2017, pp. 155–160. [Online]. Available: <http://doi.acm.org/10.1145/3078081.3078098>
- [231] C. Reul, S. Göttel, U. Springmann, C. Wick, K.-M. Würzner, and F. Puppe, “Automatic Semantic Text Tagging on Historical Lexica by Combining OCR and Typography Classification: A Case Study on Daniel Sander’s Wörterbuch der Deutschen Sprache,” in *Proceedings of the 3rd International Conference on Digital Access to Textual Cultural Heritage*. ACM, 2019, pp. 33–38. [Online]. Available: <https://doi.org/10.1145/3322905.3322910>
- [232] C. Reul, U. Springmann, and F. Puppe, “LAREX: A semi-automatic open-source tool for layout analysis and region extraction on early printed books,” in *Proceedings of the 2nd International Conference on Digital Access to Textual Cultural Heritage*, ser. DATECH2017. New York, NY, USA: ACM, 2017, pp. 137–142. [Online]. Available: <http://doi.acm.org/10.1145/3078081.3078097>
- [233] C. Reul, U. Springmann, C. Wick, and F. Puppe, “Improving OCR Accuracy on Early Printed Books by combining Pretraining, Voting, and Active Learning,” *JLCL: Special Issue on Automatic Text and Layout Recognition*, vol. 33, no. 1, pp. 3–24, 2018. [Online]. Available: https://jlcl.org/content/2-allissues/1-heft1-2018/jlcl_2018-1_1.pdf
- [234] —, “Improving OCR Accuracy on Early Printed Books by utilizing Cross Fold Training and Voting,” in *2018 13th IAPR International Workshop on Document Analysis Systems (DAS)*. IEEE, 2018, pp. 423–428. [Online]. Available: <https://doi.org/10.1109/DAS.2018.30>
- [235] —, “State of the Art Optical Character Recognition of 19th Century Fraktur Scripts using Open Source Engines,” *DHd 2019 Digital Humanities: multimedial & multimodal*, 2019. [Online]. Available: <https://doi.org/10.5281/zenodo.2596095>
- [236] C. Reul, C. Wick, U. Springmann, and F. Puppe, “Transfer learning for OCRopus model training on early printed books,” *027.7 Journal for Library Culture*, vol. 5, no. 1, pp. 38–51, 2017. [Online]. Available: <http://dx.doi.org/10.12685/027.7-5-1-169>

- [237] S. V. Rice, F. R. Jenkins, and T. A. Nartker, *The fifth annual test of OCR accuracy*. Information Science Research Institute, 1996.
- [238] S. V. Rice, J. Kanai, and T. A. Nartker, *A report on the accuracy of OCR devices*. University of Nevada, Information Science Research Institute, 1992.
- [239] ———, “An algorithm for matching OCR-generated text strings,” *International Journal of Pattern Recognition and Artificial Intelligence*, vol. 8, no. 05, pp. 1259–1268, 1994.
- [240] S. V. Rice and T. A. Nartker, “The ISRI analytic tools for OCR evaluation,” *UNLV/Information Science Research Institute, TR-96-02*, 1996.
- [241] L. Richardson and S. Ruby, *RESTful web services*. "O'Reilly Media, Inc.", 2008.
- [242] “RIDGES corpus of herbals,” <http://korpling.org/ridges>, accessed: 2019-11-07.
- [243] V. Romero, J. A. Sanchez, V. Bosch, K. Depuydt, and J. de Does, “Influence of text line segmentation in handwritten text recognition,” in *2015 13th International Conference on Document Analysis and Recognition (ICDAR)*. IEEE, 2015, pp. 536–540.
- [244] O. Ronneberger, P. Fischer, and T. Brox, “U-net: Convolutional networks for biomedical image segmentation,” in *International Conference on Medical image computing and computer-assisted intervention*. Springer, 2015, pp. 234–241.
- [245] C. Rother, V. Kolmogorov, and A. Blake, “Grabcut: Interactive foreground extraction using iterated graph cuts,” in *ACM transactions on graphics (TOG)*, vol. 23, no. 3. ACM, 2004, pp. 309–314.
- [246] P. J. Rousseeuw and A. M. Leroy, *Robust regression and outlier detection*. John wiley & sons, 2005, vol. 589.
- [247] J. A. Rydberg-Cox, “Digitizing Latin incunabula: Challenges, methods, and possibilities,” *Digital Humanities Quarterly*, vol. 3, no. 1, 2009. [Online]. Available: <http://digitalhumanities.org:8081/dhq/vol/3/1/000027/000027.html>
- [248] R. Saabni and J. El-Sana, “Language-independent text lines extraction using seam carving,” in *2011 International Conference on Document Analysis and Recognition*. IEEE, 2011, pp. 563–568.
- [249] J. A. Sánchez, V. Bosch, V. Romero, K. Depuydt, and J. De Does, “Handwritten text recognition for historical documents in the transcriptorium project,” in *Proceedings of the First International Conference on Digital Access to Textual Cultural Heritage*. ACM, 2014, pp. 111–117.
- [250] J. Sauvola and M. Pietikäinen, “Adaptive document image binarization,” *Pattern recognition*, vol. 33, no. 2, pp. 225–236, 2000.

Bibliography

- [251] H. F. Schantz, *History of OCR, Optical Character Recognition*. Recognition Technologies Users Association, 1982.
- [252] H.-G. Schmidt, “Kallimachos: Digital Humanities als Auftrag der Universitätsbibliothek Würzburg,” *ABI Technik*, vol. 36, no. 3, pp. 170–177, 2016.
- [253] J. Serra, *Image analysis and mathematical morphology*. Academic Press, Inc., 1983.
- [254] B. Settles, “Active learning,” *Synthesis Lectures on Artificial Intelligence and Machine Learning*, vol. 6, no. 1, pp. 1–114, 2012.
- [255] H. S. Seung, M. Opper, and H. Sompolinsky, “Query by committee,” in *Proceedings of the fifth annual workshop on Computational learning theory*. ACM, 1992, pp. 287–294.
- [256] M. Seuret, M. Bouillon, F. Simistira, M. Würsch, M. Liwicki, and R. Ingold, “A Semi-Automatized Modular Annotation Tool for Ancient Manuscript Annotation,” in *2018 13th IAPR International Workshop on Document Analysis Systems (DAS)*. IEEE, 2018, pp. 340–344.
- [257] M. Seuret, R. Ingold, and M. Liwicki, “N-light-n: A highly-adaptable java library for document analysis with convolutional auto-encoders and related architectures,” in *2016 15th International Conference on Frontiers in Handwriting Recognition (ICFHR)*. IEEE, 2016, pp. 459–464.
- [258] F. Shafait and T. M. Breuel, “Document image dewarping contest,” in *2nd Int. Workshop on Camera-Based Document Analysis and Recognition, Curitiba, Brazil, 2007*, pp. 181–188.
- [259] —, “A simple and effective approach for border noise removal from document images,” in *2009 IEEE 13th International Multitopic Conference*. IEEE, 2009, pp. 1–5.
- [260] —, “The effect of border noise on the performance of projection-based page segmentation methods,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 33, no. 4, pp. 846–851, 2010.
- [261] F. Shafait, D. Keysers, and T. Breuel, “Performance evaluation and benchmarking of six-page segmentation algorithms,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 30, no. 6, pp. 941–954, 2008.
- [262] F. Shafait, D. Keysers, and T. M. Breuel, “Performance comparison of six algorithms for page segmentation,” in *International Workshop on Document Analysis Systems*. Springer, 2006, pp. 368–379.
- [263] —, “Pixel-accurate representation and evaluation of page segmentation in document images,” in *18th International Conference on Pattern Recognition (ICPR’06)*, vol. 1. IEEE, 2006, pp. 872–875.

- [264] ———, “Projection methods require black border removal,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 31, no. 4, pp. 763–764, 2008.
- [265] F. Shafait and R. Smith, “Table detection in heterogeneous documents,” in *Proceedings of the 9th IAPR International Workshop on Document Analysis Systems*. ACM, 2010, pp. 65–72.
- [266] F. Shafait, J. Van Beusekom, D. Keysers, and T. M. Breuel, “Document cleanup using page frame detection,” *International Journal of Document Analysis and Recognition (IJ DAR)*, vol. 11, no. 2, pp. 81–96, 2008.
- [267] Z. Shi and V. Govindaraju, “Line separation for complex document images using fuzzy runlength,” in *First International Workshop on Document Image Analysis for Libraries, 2004. Proceedings*. IEEE, 2004, pp. 306–312.
- [268] F. Simistira, M. Seuret, N. Eichenberger, A. Garz, M. Liwicki, and R. Ingold, “Divahisdb: A precisely annotated large dataset of challenging medieval manuscripts,” in *2016 15th International Conference on Frontiers in Handwriting Recognition (ICFHR)*. IEEE, 2016, pp. 471–476.
- [269] F. Simistira, M. Bouillon, M. Seuret, M. Würsch, M. Alberti, R. Ingold, and M. Liwicki, “Icdar2017 competition on layout analysis for challenging medieval manuscripts,” in *2017 14th IAPR International Conference on Document Analysis and Recognition (ICDAR)*, vol. 1. IEEE, 2017, pp. 1361–1370.
- [270] F. Slimane, R. Ingold, and J. Hennebert, “ICDAR2017 Competition on Multi-Font and Multi-Size Digitally Represented Arabic Text,” in *Document Analysis and Recognition (ICDAR), 2017 14th IAPR International Conference on*, vol. 1. IEEE, 2017, pp. 1466–1472.
- [271] F. Slimane, S. Kanoun, J. Hennebert, A. M. Alimi, and R. Ingold, “A study on font-family and font-size recognition applied to arabic word images at ultra-low resolution,” *Pattern Recognition Letters*, vol. 34, no. 2, pp. 209–218, 2013.
- [272] D. Smith and R. Cordell, “A research agenda for historical and multilingual optical character recognition,” 2018.
- [273] R. Smith, “A simple and efficient skew detection algorithm via text row accumulation,” in *Proceedings of 3rd International Conference on Document Analysis and Recognition*, vol. 2. IEEE, 1995, pp. 1145–1148.
- [274] ———, “An overview of the Tesseract OCR engine,” in *Ninth International Conference on Document Analysis and Recognition (ICDAR 2007)*, vol. 2. IEEE, 2007, pp. 629–633.
- [275] R. W. Smith, “Hybrid page layout analysis via tab-stop detection,” in *2009 10th International Conference on Document Analysis and Recognition*. IEEE, 2009, pp. 241–245.

Bibliography

- [276] S. Snyderman, R. Sanderson, and T. Cramer, “The international image interoperability framework (iiif): A community & technology approach for web-based images,” in *Archiving Conference*, vol. 2015, no. 1. Society for Imaging Science and Technology, 2015, pp. 16–21.
- [277] U. Springmann and F. Fink, “CIS OCR Workshop v1. 0: OCR and postcorrection of early printings for digital humanities,” 2016. [Online]. Available: <https://doi.org/10.5281/zenodo.46571>
- [278] U. Springmann, “OCR für alte Drucke.” *Informatik Spektrum*, vol. 39, no. 6, pp. 459–462, 2016. [Online]. Available: <https://doi.org/10.1007/s00287-016-1004-3>
- [279] U. Springmann, F. Fink, and K. U. Schulz, “Automatic quality evaluation and (semi-) automatic improvement of mixed models for OCR on historical documents,” *arXiv preprint arXiv:1606.05157*, 2016. [Online]. Available: <https://arxiv.org/abs/1606.05157>
- [280] U. Springmann and A. Lüdeling, “OCR of historical printings with an application to building diachronic corpora: A case study using the RIDGES herbal corpus,” *Digital Humanities Quarterly*, vol. 11, no. 2, 2017. [Online]. Available: <http://www.digitalhumanities.org/dhq/vol/11/2/000288/000288.html>
- [281] U. Springmann, D. Najock, H. Morgenroth, H. Schmid, A. Gotscharek, and F. Fink, “OCR of historical printings of Latin texts: problems, prospects, progress,” in *Proceedings of the First International Conference on Digital Access to Textual Cultural Heritage*. ACM, 2014, pp. 71–75. [Online]. Available: <https://doi.org/10.1145/2595188.2595205>
- [282] U. Springmann, C. Reul, S. Dipper, and J. Baiter, “Ground Truth for training OCR engines on historical documents in German Fraktur and Early Modern Latin,” *JLCL: Special Issue on Automatic Text and Layout Recognition*, 2019. [Online]. Available: https://jlcl.org/content/2-allissues/1-heft1-2018/jlcl_2018-1_5.pdf
- [283] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, “Dropout: a simple way to prevent neural networks from overfitting,” *The journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014.
- [284] T. Stäcker, “Konversion des kulturellen Erbes für die Forschung: Volltextbeschaffung und -bereitstellung als Aufgabe der Bibliotheken,” *o-bib. Das offene Bibliotheksjournal*, vol. 1, no. 1, pp. 220–237, 2014.
- [285] N. Stamatopoulos, B. Gatos, and A. Kesidis, “Automatic borders detection of camera document images,” in *2nd International Workshop on Camera-Based Document Analysis and Recognition, Curitiba, Brazil*, 2007, pp. 71–78.
- [286] N. Stamatopoulos, B. Gatos, and T. Georgiou, “Page frame detection for double page document images,” in *Proceedings of the 9th IAPR International Workshop on Document Analysis Systems*. ACM, 2010, pp. 401–408.

- [287] C. Stollwerk, “Machbarkeitsstudie zu Einsatzmöglichkeiten von OCR-Software im Bereich “Alter Drucke” zur Vorbereitung einer vollständigen Digitalisierung deutscher Druckerzeugnisse zwischen 1500 und 1930,” *Niedersächsische Staats- und Universitätsbibliothek Göttingen, Göttingen*, 2016.
- [288] A. Sulaiman, K. Omar, and M. F. Nasrudin, “Degraded Historical Document Binarization: A Review on Issues, Challenges, Techniques, and Future Directions,” *Journal of Imaging*, vol. 5, no. 4, p. 48, 2019.
- [289] H.-M. Sun, “Page segmentation for Manhattan and non-Manhattan layout documents via selective CRLA,” in *Eighth International Conference on Document Analysis and Recognition (ICDAR’05)*. IEEE, 2005, pp. 116–120.
- [290] S. Tanner, “Digitization of Printed Material: The Metadata Engine Project (METAe),” *Library Hi Tech News*, vol. 18, no. 4, 2001.
- [291] D. Tao, X. Lin, L. Jin, and X. Li, “Principal component 2-D long short-term memory for font recognition on single Chinese characters,” *IEEE transactions on cybernetics*, vol. 46, no. 3, pp. 756–765, 2016.
- [292] “Text Encoding Initiative (TEI),” <https://tei-c.org>, accessed: 2019-10-15.
- [293] C. Tensmeyer, B. Davis, C. Wigington, I. Lee, and B. Barrett, “Pagenet: Page boundary extraction in historical handwritten documents,” in *Proceedings of the 4th International Workshop on Historical Document Imaging and Processing*. ACM, 2017, pp. 59–64.
- [294] C. Tensmeyer, D. Saunders, and T. Martinez, “Convolutional Neural Networks for Font Classification,” in *Document Analysis and Recognition (ICDAR), 2017 14th IAPR International Conference on*, vol. 1. IEEE, 2017, pp. 985–990.
- [295] “Tesseract GitHub repository,” <https://github.com/tesseract-ocr/tesseract>, accessed: 2019-10-27.
- [296] “Tesseract Wiki: Improving OCR Quality,” <https://github.com/tesseract-ocr/tesseract/wiki/ImproveQuality>, accessed: 2019-10-27.
- [297] “Tesseract Tutorial at DAS 2016,” https://github.com/tesseract-ocr/docs/tree/master/das_tutorial2016, accessed: 2019-11-10.
- [298] “Tesstrain - Training workflow for Tesseract 4 as a Makefile,” <https://github.com/tesseract-ocr/tesstrain>, accessed: 2019-12-12.
- [299] The Unicode Consortium, “The Unicode Standard,” <http://www.unicode.org/versions/latest>, accessed: 2019-12-01.
- [300] “Apache Tomcat 9 Documentation Index,” <http://tomcat.apache.org/tomcat-9.0-doc>, accessed: 2019-11-22.

Bibliography

- [301] M. C. Traub, J. Van Ossenbruggen, and L. Hardman, “Impact analysis of OCR quality on research tasks in digital archives,” in *International Conference on Theory and Practice of Digital Libraries*. Springer, 2015, pp. 252–263.
- [302] “Typenrepertorium der Wiegendrucke,” <https://tw.staatsbibliothek-berlin.de>, accessed: 2019-12-01.
- [303] K. Ubul, G. Tursun, A. Aysa, D. Impedovo, G. Pirlo, and T. Yibulayin, “Script Identification of Multi-Script Documents: A Survey.” *IEEE Access*, vol. 5, pp. 6546–6559, 2017.
- [304] A. Ul-Hasan, M. Z. Afzal, F. Shafait, M. Liwicki, and T. M. Breuel, “A sequence learning approach for multiple script identification,” in *Document Analysis and Recognition (ICDAR), 2015 13th International Conference on*. IEEE, 2015, pp. 1046–1050.
- [305] A. Ul-Hasan and T. M. Breuel, “Can we build language-independent OCR using LSTM networks?” *Proceedings of the 4th International Workshop on Multilingual OCR*, p. 9, 2013.
- [306] “Unpaper Homepage,” <https://www.berlios.de/software/unpaper>, accessed: 2019-10-27.
- [307] “Unpaper GitHub repository,” <https://github.com/Flameeyes/unpaper>, accessed: 2019-10-27.
- [308] “Lines of the UW3 dataset,” <http://www.tmbdev.net/ocrdata-split>, accessed: 2019-10-15.
- [309] J. Van Beusekom, F. Shafait, and T. M. Breuel, “Automated OCR ground truth generation,” in *2008 The Eighth IAPR International Workshop on Document Analysis Systems*. IEEE, 2008, pp. 111–117.
- [310] —, “Combined orientation and skew detection using geometric text-line modeling,” *International Journal on Document Analysis and Recognition (IJDAR)*, vol. 13, no. 2, pp. 79–92, 2010.
- [311] O. V. Virtualbox, “Oracle vm virtualbox,” *Change*, vol. 107, pp. 1–287, 2011.
- [312] T. Vobl, A. Gotscharek, U. Reffle, C. Ringlstetter, and K. U. Schulz, “PoCoTo—an open source system for efficient interactive postcorrection of OCRed historical texts,” in *Proceedings of the First International Conference on Digital Access to Textual Cultural Heritage*. ACM, 2014, pp. 57–61.
- [313] Y. Wang, I. T. Phillips, and R. M. Haralick, “Document zone content classification and its performance evaluation,” *Pattern Recognition*, vol. 39, no. 1, pp. 57–73, 2006.

- [314] H. Wei, K. Chen, R. Ingold, and M. Liwicki, “Hybrid feature selection for historical document layout analysis,” in *2014 14th International Conference on Frontiers in Handwriting Recognition*. IEEE, 2014, pp. 87–92.
- [315] D. Wemhoener, I. Z. Yalniz, and R. Manmatha, “Creating an improved version using noisy OCR from multiple editions,” in *Document Analysis and Recognition (ICDAR), 2013 12th Int. Conf. on*. IEEE, 2013, pp. 160–164.
- [316] C. Wick and F. Puppe, “Leaf Identification Using a Deep Convolutional Neural Network,” *arXiv preprint arXiv:1712.00967*, 2017. [Online]. Available: <https://arxiv.org/abs/1712.00967>
- [317] —, “Fully convolutional neural networks for page segmentation of historical document images,” in *2018 13th IAPR International Workshop on Document Analysis Systems (DAS)*. IEEE, 2018, pp. 287–292. [Online]. Available: <https://doi.org/10.1109/DAS.2018.39>
- [318] C. Wick, C. Reul, and F. Puppe, “Calamari - A High-Performance Tensorflow-based Deep Learning Package for Optical Character Recognition,” *Digital Humanities Quarterly (forthcoming)*, 2018. [Online]. Available: <https://arxiv.org/abs/1807.02004>
- [319] —, “Comparison of OCR Accuracy on Early Printed Books using the Open Source Engines Calamari and OCRopus,” *JLCL: Special Issue on Automatic Text and Layout Recognition*, vol. 33, no. 1, pp. 79–96, 2018. [Online]. Available: https://jlcl.org/content/2-allissues/1-heft1-2018/jlcl_2018-1_4.pdf
- [320] K. Y. Wong, R. G. Casey, and F. M. Wahl, “Document analysis system,” *IBM journal of research and development*, vol. 26, no. 6, pp. 647–656, 1982.
- [321] M. Würsch, R. Ingold, and M. Liwicki, “Sdk reinvented: Document image analysis methods as restful web services,” in *2016 12th IAPR Workshop on Document Analysis Systems (DAS)*. IEEE, 2016, pp. 90–95.
- [322] M. Würsch, M. Liwicki, and R. Ingold, “Web services in document image analysis—recent developments on divaservices and the importance of building an ecosystem,” in *2018 13th IAPR International Workshop on Document Analysis Systems (DAS)*. IEEE, 2018, pp. 334–339.
- [323] K.-M. Würzner, “(Open-Source-) OCR-Workflows,” https://edoc.bbaw.de/files/2786/20170804_dh_kolloquium_Wuerzner.pdf, accessed: 2019-11-15.
- [324] M. Wüthrich, M. Liwicki, A. Fischer, E. Indermühle, H. Bunke, G. Viehhauser, and M. Stolz, “Language model integration for the recognition of handwritten medieval documents,” in *2009 10th International Conference on Document Analysis and Recognition*. IEEE, 2009, pp. 211–215.

Bibliography

- [325] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson, “How transferable are features in deep neural networks?” in *Advances in neural information processing systems*, 2014, pp. 3320–3328.
- [326] M. R. Yousefi, M. R. Soheili, T. M. Breuel, E. Kabir, and D. Stricker, “Binarization-free OCR for historical documents using LSTM networks,” in *2015 13th International Conference on Document Analysis and Recognition (ICDAR)*. IEEE, 2015, pp. 1121–1125.
- [327] A. Zahour, B. Taconet, P. Mercy, and S. Ramdane, “Arabic hand-written text-line extraction,” in *Proceedings of Sixth International Conference on Document Analysis and Recognition*. IEEE, 2001, pp. 281–285.
- [328] “Zedler Lexikon,” <https://www.zedler-lexikon.de>, accessed: 2019-11-08.
- [329] J. Zedlitz, “Fraktur Model GitHub repository,” https://github.com/jze/ocropus-model_fraktur, accessed: 2019-11-03.
- [330] Y. Zhu, T. Tan, and Y. Wang, “Font recognition based on global texture analysis,” *IEEE Transactions on pattern analysis and machine intelligence*, vol. 23, no. 10, pp. 1192–1200, 2001.
- [331] A. Zramdini and R. Ingold, “Optical font recognition from projection profiles,” *Electronic Publishing*, vol. 6, no. 3, pp. 249–260, 1993.
- [332] —, “Optical font recognition using typographical features,” *IEEE Transactions on pattern analysis and machine intelligence*, vol. 20, no. 8, pp. 877–882, 1998.

Acknowledgements / Danksagung

Viele Personen haben mich in den spannenden, aber auch fordernden letzten vier Jahren unterstützt. Als erstes möchte ich meinem Doktorvater Frank Puppe für seine stets offene Tür danken und für die bedingungslose Bereitschaft, sich spontan und nahezu immer Zeit für umfassende Diskussionen und hilfreiche Anregungen zu nehmen. Außerdem danke ich meinem Zweitgutachter Marcus Liwicki für seine Hilfsbereitschaft und das konstruktive Feedback.

Des Weiteren danke ich meinen Kolleg*innen, die in den vergangenen Jahren nicht nur stets für eine angenehme Arbeitsatmosphäre gesorgt haben, sondern mir auch immer hilfreich mit Rat und Tat zur Seite standen. Besonderer Dank gilt dabei Uwe Springmann, der es mir nicht nur ermöglicht hat, sehr zügig in der spannenden Welt der OCR Fuß zu fassen, sondern auch stets als Diskussionspartner zu technischen und anwendungsbezogenen Fragen auf Augenhöhe zur Verfügung stand. Ebenfalls großer Dank gilt meinem langjährigen Zimmerkollegen Christoph Wick, nicht nur für die zahllosen hilfreichen Diskussionen, sondern auch für seine umgängliche Art.

Ebenfalls großen Anteil am Gelingen dieser Arbeit haben die, u. a. aus dem *Narragonien digital* und dem *Camerarius* Projekt stammenden, zahlreichen “nicht-technischen Nutzer”, deren unermüdliches Transkribieren, Testen und Feedback-geben uns viele Evaluationen und Entwicklungen überhaupt erst ermöglicht haben.

Außerdem möchte ich mich bei meinen Eltern Albrecht und Karin Reul bedanken, die mich stets unterstützt haben. Abschließend gebührt größter Dank meiner Frau Simone und meinen Kindern Lina, Toni und Mats, die mir zu jeder Zeit und speziell in den schweren letzten Monaten stets ein sicherer Rückhalt waren und für den nötigen Ausgleich gesorgt haben.

Würzburg, Januar 2020