OCR and post-correction of historical newspapers and journals

Senka Drobac

Doctoral dissertation, to be presented for public discussion with the permission of the Faculty of Arts of the University of Helsinki, in Hall 6, Metsätalo Building, on the 9th of October, 2020 at 12 o'clock.

ISBN 978-951-51-6511-4 (paperback)
ISBN 978-951-51-6512-1 (PDF)
University of Helsinki
Helsinki 2020

Abstract

The corpus of historical newspapers and journals published in Finland, with more than 11 million pages of historical text, is of great value to the research community. The National Library of Finland (NLF) has OCRed the corpus with ABBYY FineReader, a commercial software that provides OCR models pre-trained on general historical fonts. The estimated accuracy of the OCRed text is between 87 % - 92 % on the character level, which is rather low even for scientific research.

Optical character recognition of printed text commonly reaches over 99 % accuracy for modern Latin fonts. Historical documents, on the other hand, contain a large variety of fonts, can be of poor condition and often are written without an orthographic standard (the same words are spelled differently). All these reasons present a challenge to creating robust and highly accurate OCR models for historical data.

The corpus of historical newspapers and journals published in Finland is particularly challenging because it is written in both the official languages of Finland (Finnish and Swedish) and is printed in two font-families (Blackletter and Antiqua). With two main languages and a large number of different fonts from two font-families, it is not possible to achieve high OCR accuracy with models pre-trained on different materials.

A research group at the NLF has worked on re-OCRing this corpus and they have trained OCR models using the open-source software Tesseract, but only for the Finnish Blackletter part of the corpus. They report high accuracy results (97.64 % on character level) for Finnish Blackletter but also slow performance. For the Antiqua part of the corpus, they reportedly use Tesseract's pre-trained Antiqua model, but they do not report any accuracy results. Also, they have still not published any work done on the material written in Swedish.

In this work, we have explored methods and practices for training high-accuracy OCR models that can be used for efficiently recognizing the entire corpus of historical Finnish newspapers and journals. We selected 13,000 Finnish and 11,000 Swedish text lines from the corpus, of which half are printed in Blackletter and half in Antiqua fonts. After transcribing these lines, we used them for training and testing OCR models with two open-source OCR tool-kits, Ocropy and Calamari. We performed experiments with different training data setups, along with different neural network configurations and architectures. Furthermore, we tested how the

voting mechanism behaves with different OCR models.

Post-correction can further improve the final OCR results, especially in cases when the text, due to material damage or ink bleed, is incomprehensible without a broader context. Therefore, we have also explored different post-correction methods and implemented one of them. We compared the method's effect on OCR results of different accuracy.

The biggest accomplishment of this work is succeeding in training a high-accuracy model that is capable of recognizing both Finnish and Swedish text, as well as Blackletter and Antiqua fonts. Having a mixed model for all the data and not needing to separately perform language or font identification is extremely practical when dealing with such a large corpus.

Furthermore, we found that the results improve when voting with five mixed models, resulting in accuracy between 97.2 % and 98.4 % on the character level, which is up to 11 % better than the current ABBYY results.

Finally, the post-correction experiments showed that, even with a simple automatic method, post-correction can further improve OCR results. Depending on the starting OCR accuracy, the post-correction improved accuracy between 0.1-0.4 %, which is a relative improvement of 0.9-12.5 %.

List of Publications

I Drobac, S., Kauppinen, P., and Lindén, K. (2017). OCR and post-correction of historical Finnish texts. In *Proceedings of the 21st Nordic Conference on Computational Linguistics*, pages 70–76

II Drobac, S., Kauppinen, P., and Linden, K. (2019). Improving OCR of historical newspapers and journals published in finland. In *Proceedings of the 3nd International Conference on Digital Access to Textual Cultural Heritage*. ACM

III Drobac, S. and Lindén, K. (2020). Optical character recognition with neural networks and post-correction with finite state methods. *International Journal on Document Analysis and Recognition*

IV Drobac, S. and Lindén, K. (2019). Optical font family recognition using a neural network. In *Proceedings of the Research Data And Humanities (RDHum) 2019 Conference: Data, Methods And Tools*, page 115, Finland. Studia Humaniora Ouluensia

V Drobac, S., Silfverberg, M., and Yli-Jyrä, A. (2012). Implementation of replace rules using preference operator. In *Proceedings of the FSMNLP 2012*, United States. ACL Anthology

VI Linden, K., Silfverberg, M., Pirinen, T., Hardwick, S., Drobac, S., and Axelson, E. (2012). *HFST - an Environment for Creating Language Technology Applications*. Studies in Computational Intelligence. Springer-Verlag, Germany

VII Linden, K., Axelson, E., Drobac, S., Hardwick, S., Kuokkala, J., Niemi, J., Pirinen, T., and Silfverberg, M. (2013). *HFST — a System for Creating NLP Tools*, pages 53–71. Communications in Computer and Information Science. Springer-Verlag, Germany

Author's Contributions

Publication I: OCR and post-correction of historical Finnish texts

I contributed to the data creation together with the second author Pekka Kauppinen. I performed the OCR experiments and wrote the paper together with the second author Pekka Kauppinen under the supervision of the third author Krister Lindén.

Publication II: Improving OCR of historical newspapers and journals published in Finland

I contributed to the data creation together with the second author Pekka Kauppinen. I performed the OCR experiments and wrote the paper under the supervision of the third author Krister Lindén.

Publication III: Optical character recognition with neural networks and post-correction with finite state methods

I selected and sampled data for transcription, transcribed the Finnish data set, run the experiments and wrote the paper under the supervision of the second author Krister Lindén.

Publication IV: Optical Font Family Recognition using a Neural Network

I developed the system, created the training and test data sets, performed the experiments and wrote the paper under the supervision of the second author Krister Lindén.

Publication V: Implementation of replace rules using preference operator

I developed the system together with second author Miikka Silfverberg and wrote the paper together with the second author, under the supervision of the third author Anssi Yli-Jyrä.

Publication VI: HFST - an Environment for Creating Language Technology Applications

I developed the parallel replace rules, added support for parsing the rules and wrote Section 2.2 under the supervision of the first author Krister Lindén.

Publication VII: HFST—a System for Creating NLP Tools

I performed experiments with replace rules and wrote Section 3.1 together with the second author Erik Axelson under the supervision of the first author Krister Lindén.

Preface and Acknowledgements

My doctoral journey started in 2011 when I got appointed at the University of Helsinki as a CLARA early stage researcher in Krister's HFST team. I started my Ph.D. thesis on a completely different topic: *Hyper-minimization of finite state machines*, with the aim to hyper-minimize Greenlandic morphology. However, in 2015 when I got back from a one-year maternity leave, the problem I've been working on had already been solved and I had to change my research plan. Although the direction of my thesis took a sudden shift, the experience I got working with amazing people has helped me in the remaining part of my journey. Big thanks to Miikka, Anssi, Sam, Erik and Tommi who were there with me during those HFST years.

Around the time I started thinking in which direction to continue, Miikka and Pekka have been working with Krister on OCR post-correction with replace rules. They worked with the corpus of historical Finnish newspapers and journals, which quickly proved to have low OCR quality. I got interested in the idea of trying to create a better OCR for this corpus and have pointed my research in that direction.

Although the OCR field was new to all of us, Krister had managed to successfully guide me through the process. When analyzing intermediate results, he always had interesting theories of why the models were behaving in the way they did and we've spent many hours having interesting discussions together. Of course, sometimes our assumptions were wrong, but also sometimes we had unexpected breakthroughs which made the whole process engaging, fun and interesting. Huge thanks to Krister for mentoring me, I can't imagine getting to this point without his help.

Furthermore, I would like to thank Pekka who put a great effort into creating the training data and doing the early OCR research with us. It was also nice (and surprising) to have someone at the workplace who I can talk to in my mother tongue.

Many thanks also to Mathias who has helped me with getting teaching experience. It was valuable to see how a caring approach has helped students that would otherwise have been left behind. Big thanks also to Jörg for comments on the thesis, but also for doing amazing changes to the department.

I am also grateful for the valuable comments of the preliminary examiners, Peter Z. Revesz and Dana Dannélls.

Lastly, I would like to thank my family for their love and support, without you guys nothing would matter.

Contents

Chapter 1

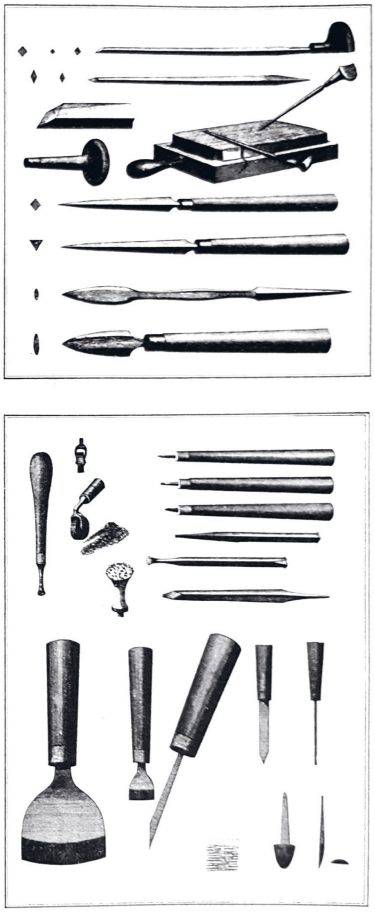
Introduction

Optical character recognition (OCR) is the conversion of images of text into a digital format. Nowadays, OCR of Latin fonts is of such high accuracy that it is considered a solved problem, and many software packages provide free OCR technology in many different languages. However, there is still much research being done on Asian and Middle Eastern scripts, handwritten texts and historical printings.

Historical printings are difficult to recognize because of the large variety of fonts. Font molds were hand-made so glyphs differ from publisher to publisher and also through different editions, as molds were changed through time. Figure 1.1 shows illustrations from [Jäntti, 1940] that illustrate tools used by engravers for font production, as well as engraver work and printing machines in Germany in the 18$^{\text{th}}$ century.

Although font variation is possibly the most important reason for the difficulty of historical OCR, there are other reasons as well. There is great spelling variation in historical printings. For example, although the standardization of the Finnish literary language began in the early 19$^{\text{th}}$ century, Finland did not have an official spelling standard until the mid 20$^{\text{th}}$ century, when *The Dictionary of Modern Finnish (Nykysuomen sanakirja)* was published. Also, historical lexicons are usually not available or are incomplete, which makes automatic OCR post-correction difficult. The quality of the material is another issue. We can find smudged pages, print bleed or parts of pages missing. In historical printings we also have to deal with segmentation issues. Historical books often have marginalia texts, while newspapers frequently changed page layouts. Furthermore, in both books and newspapers, there are drop cap letters that most of the page segmentation software still cannot process correctly.

In this work, we focus on the OCR of historical newspapers published in Finland 1771 - 1929. Although the corpus was recognized several years ago, its accuracy is considered to be too low for regular use. With the development of new technologies and systems, we are now able to train neural models for OCR and post-correction. The custom-made models tailored to the specific data enable us to achieve high accuracy despite the challenges posed by historical variance.



Kaivertajan työvälineet.

(a) Engraver tools

Saksalainen vaskipiirroksen kaivertaja ja painaja n. v. 1700.

(b) Engraver (top) and printing machine (bottom)

Figure 1.1: Two pictures from *Yrjö A. Jäntti, Kirjapainotaidon historia, 1940.* illustrating (a) engraver tools that were used for production of font molds, and (b) engraver work (top) and a printing machine used in Germany in 18$^{\text{th}}$ century (bottom).

1.1 History of OCR

The idea of OCR machines originates in the late 19$^{\text{th}}$ century, with several researchers working on the development of the first mechanical devices. Gustav Tauschek developed and in 1929 patented the “Reading Machine”, which is consid-

ered to be the first mechanical OCR device [Tauschek, 1935]. The illustration of the machine is shown in Figure 1.2. It was a mechanical device, which had a photo-detector (3) that matched letters placed in front of the lens (1), with pre-defined letters on a rotating disk (5) behind the lens. When there was a match, the click work (15) rotated the printing drum (16) to the required letter, which was then printed on a paper.

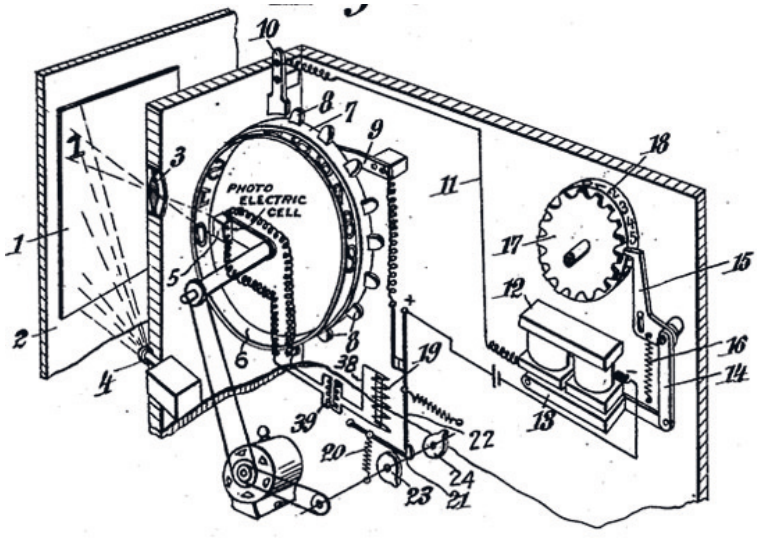


Figure 1.2: The Reading Machine by Gustav Tauschek, an illustration from his patent.

The reading machine was intended for office machinery operated by persons who could not read, as well as for testing and counting securities and bank-notes [Tauschek, 1935]. Furthermore, other mechanical OCR devices were developed in the early 20$^{\text{th}}$ century with different purposes, like the "Optophone" by [d'Albe, 1914] whose aim was to read a text for blind people and the "Statistical Machine" by [Goldberg, 1928], which could convert text to telegraph code.

The development of computers and digital data started the evolution of OCR software. In the 1970s, American inventor Ray Kurzweil founded a company called *Kurzweil Computer Products Inc.*, which developed the "Kurzweil Reading Machine". The main purpose of the machine was to translate printed text into speech, aimed mainly at visually impaired users. It consisted of a control panel, an optical scanner, a computer, and a voice output. The computer had a program that performed OCR, and it was capable of recognizing different font types with different success rates. The maximum reading speed was up to 150 words per minute in the second generation, which increased to 250 words per minute in the third generation. In [Goodrich et al., 1979], the first and the second generation of the machine were

found to produce a very wide error rate range (from under 1 % to over 20 %), depending on the font, source material and quality of the data.

In 1989, David Yang founded the Russian company ABBYY, focused on products for converting paper files to digital data. Encouraged by great demand for OCR software at the time, they released ABBYY FineReader 1 in 1993 , which was met with considerable interest. The FineReader's key advantages were support for all popular fonts and multiple languages, which makes it one of the leading OCR software even today.[ABBYY, 2016]

The era of open source OCR software started in the 2000s. The originally proprietary software Tesseract was released as open-source in 2005 after 20 years of development, by Hewlett Packard and the University of Nevada, Las Vegas. The development of the software has been sponsored by Google since 2006. While the first versions were limited only to the recognition function, version 3 introduced layout analysis and support for many input and output formats.[Wikipedia, 2019c]

In 2007, Thomas Breuel released the first version of *Ocropy*, then called *OCRopus* [Breuel, 2008]. The purpose of the toolkit was to build a system for large-scale digital library applications. They took a modular approach, combining different tools for different stages of the OCR process. Therefore, the system contained modules for pre-processing, layout analysis, text line recognition and statistical language modeling with weighted finite-state transducers. In the first version, they used two tools in the recognition phase and one of them was Tesseract.

Up until 2013, when [Breuel et al., 2013] presented a new version of *Ocropy* with line-level text recognition, all OCR methods worked on a glyph level. The new *Ocropy* used Long short-term memory (LSTM) recurrent neural networks and the Connectionist Temporal Classification (CTC) alignment and loss function [Reul et al., 2019]. This made glyph segmentation unnecessary, and recognition on the line level became the new standard. Tesseract also created a line-level recognizer with LSTM in version 4, published in 2017.

Recently, [Breuel, 2017] proposed a hybrid convolutional-LSTM implementation of OCR systems, which gives much better results than previously used LSTM configurations. They also introduced training on the Graphics processing unit (GPU), which increases training speed. The hybrid CNN-LSTM approach was accepted by [Wick et al., 2018a], who created *Calamari*, a current state of the art tool for OCR. *Calamari* provides custom-defined DNNs (Deep Neural Networks) as combinations of Convolutional neural network (CNN) and LSTM layers, with the CTC loss function. The software supports training and recognition on GPU, as well as some other useful features.

1.2 OCR of Historic Newspapers

Historical newspapers published in Finland 1771 - 1929 have been recognized with commercial software ABBYY FineReader 11, with a character accuracy rate be-

tween 87 % - 92 %, which is rather low for scientific research on this data. Therefore, there is a need to re-OCR the entire corpus to obtain higher accuracy levels. However, to achieve high accuracy on this data, it is not possible to use existing pre-trained OCR models, often provided by commercial software. Rather, it is necessary to create training data by sampling from the corpus and then train specific OCR and post-correction models. This is possible with open-source software.

Lately, much work has been published on OCR of historical printings, especially on OCR of historical books. And while both historical books and newspapers share similar challenges, newspapers are quite specific due to their large font variation. Newspapers in Finland published from the 18th to the early 20th century used two font families (Blackletter and Antiqua) with a large set of fonts. Moreover, they were printed in the two main languages of Finland (Finnish and Swedish). Furthermore, data is not evenly distributed across time. In earlier years, data was more often printed in Blackletter fonts and Swedish language, while in the later data Finnish language and Antiqua fonts dominate. In the transitional period, both languages and both font families were used extensively, occasionally even on the same pages.

The wide variety of data makes it difficult to train a single model that would be able to recognize everything with high accuracy. However, due to the large corpus, which contains almost 5 billion tokens, it is necessary to find a time- and resource-efficient approach for re-OCR. Multiple processing steps for separate language and font families are time-consuming and methodologically complicated, with a higher probability for errors. Therefore, even though it is difficult to achieve a single-step recognition model, it is a preferable option.

In this work, we explore whether it is necessary to have specialized recognition models for individual languages and fonts or if it is possible to perform recognition with one single model. Additionally, we examine different ways to acquire training data, experiment with assorted open source tools and diverse neural network configurations. Finally, we use finite-state tools to create post-correction models to check if we can achieve additional accuracy gains using solely language properties.

1.3 Spell-Checking and OCR Post-Correction

The low OCR accuracy of historical documents led to the exploration and development of post-correction algorithms [Englmeier et al., 2019]. The basic way to correct the OCR output is to use spelling correction. Modern spell-checkers use lexicons of modern language and are not compatible with non-standardized historical language. However, they can be used for spelling normalization, a process that translates all the variants of historical words into their modern versions. Normalization is useful for researchers working on language change who need to find all variants of the same word, making the search easier. However, it is necessary to have the original historical spelling available, with a normalized variant as an additional option.

There are many problems associated with the post-correction of the historical data. It is possible to use spell checkers with historical variants of the language to obtain the true correction of the OCR result. However, for many languages, historical lexicons either do not exist or are incomplete (for example, the Finnish historical lexicon does not include all spelling variations). Some post-correction methods use language models, but many languages do not have sufficient historical textual data to train the language models. There is the additional problem of word ambiguity. Sometimes, it is difficult to decide which variant of the historical word was used without seeing the original text picture or the context in which it appears.

In this work, we train spelling correction error models that we test on OCR results of different quality. Although the goal of our post-correction approach is to get the text in its original historical form, the same methodology can easily be used for normalization. There are also other approaches to post-correction of the OCR data, which we present in more detail with our work in Chapter 6.

1.4 Research questions and hypotheses

The main research questions this thesis addresses are:

1. OCR of Historical Newspapers
    - Is it possible to create one mixed model that could be used to recognize the entire corpus of Finnish historical newspapers, with equal or better accuracy than monolingual specialized models?
    - Is it possible to further improve accuracy with voting?
2. OCR Training Data Creation
    - What is the optimal way of selecting the training data?
3. OCR Post-Correction
    - Is it possible to increase OCR accuracy with post-correction tools implemented with Finite state machines?

1.5 Contributions

The main contributions are:

1. OCR of Historical Newspapers
    - Methods and practices to produce one mixed model that recognizes texts printed in both Swedish and Finnish, as well as the large variety of fonts used in the material from both Blackletter and Antiqua font families.

- The pre-trained OCR model that can be used for recognition or fine-tuning, as well as monolingual Finnish and Swedish models.

2. OCR Training Data Creation

   - Best practices for selection of the training data.
   - Ground truth data sets that consist of line images, ground truth text as well as OCR results of training and test sets. OCR results of training sets are useful for the training of the post-correction models.
   - Methods for separating images containing different font-family texts
   - A neural font-family classifier and a trained model

3. OCR Post-Correction

   - Post-correction methods and practices on the data of different OCR accuracy.
   - Post-correction tools implemented with Finite state machines

1.6 Organization of the Thesis

An overview of the OCR workflow is introduced in Chapter 2. The data and the process of acquiring the data are described in Chapter 3. Evaluation methods are covered in Chapter 4. Training methods for creating OCR models are explored in detail in Chapter 5. Post-processing methods are presented in Chapter 6. Finally, future work is outlined in Chapter 7.

Chapter 2

Introduction to the OCR Workflow

The OCR process involves several different steps, shown in Figure 2.1. A page image is first pre-processed and binarized, which results in a normalized image containing only black and white pixels. The next step is segmentation of the page image into smaller images, each of which contains only one line of text. These line images can be then recognized by a pre-trained OCR model. The model is trained on line images and corresponding ground truth transcriptions. The recognition process outputs lines of text, which are then collected into a single XML file for each page. The post-correction step can either be run immediately after recognition or afterwards.

In this chapter, we will briefly describe each of the steps, while the broader overview of the process can be found in the recent publication by [Reul et al., 2019]. Furthermore, training and post-correction steps are described in detail in Chapter 5 and Chapter 6 respectively.

2.1 Image Pre-Processing

Image pre-processing is an important step in the OCR workflow, as improving image clarity can have a large impact on the OCR result. Pre-processing consists of normalization functions and binarization. While normalization corrects structural defects, either introduced in the scanning process or coming from the original material, binarization simplifies the image and helps to distinguish between printed text and shapes from the document background.

Some of the image normalization operations that are commonly performed are: deskewing, dewarping, denoising or despeckling and cropping the printing area. [Reul et al., 2019]

Deskewing is a process in which an image that is leaning to one side is straightened. Deskewing is usually performed together with other normalization functions

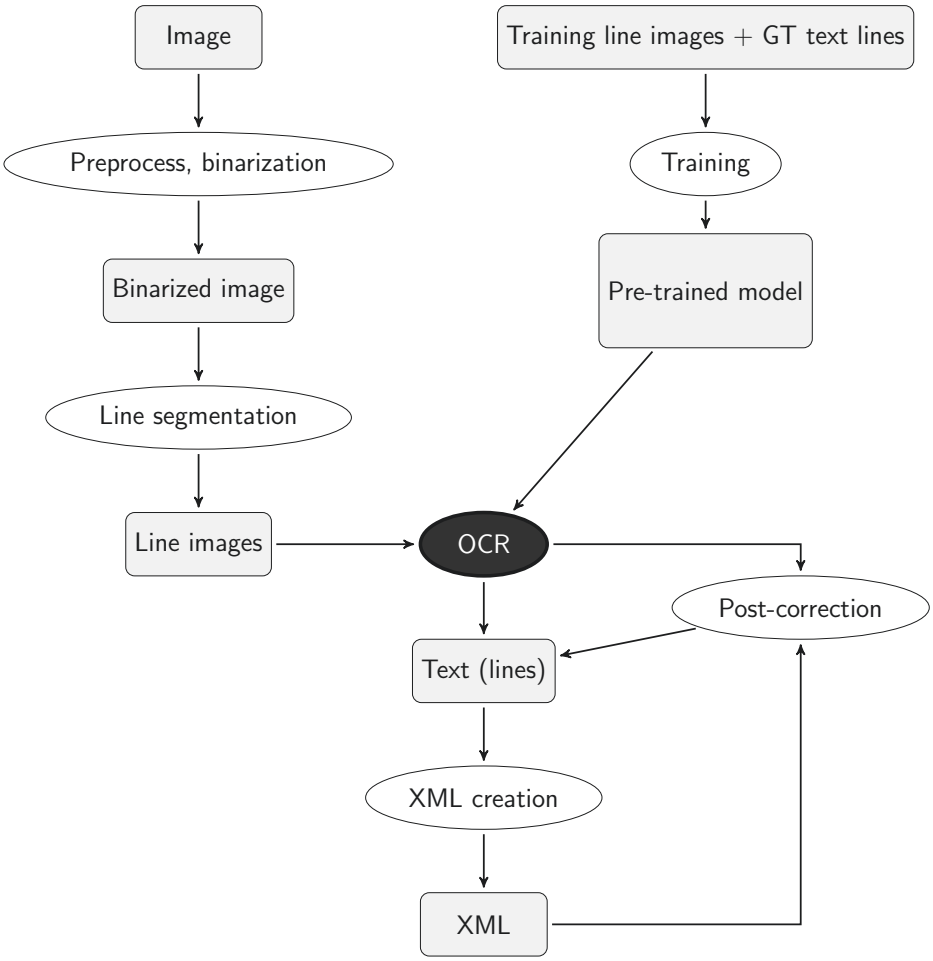


Figure 2.1: Diagram showing a typical OCR workflow. A digital image containing text is first pre-processed and binarized to get it in black and white format. Then the image is segmented, where segments containing lines of text are recognized. Line images of text are then recognized with a model that was pre-trained on images containing lines of text and corresponding ground truth, resulting in digital lines of text. The recognized text can be further post-processed or saved in an XML file and possibly post-processed at a later point.

on the page level, although recent work by [Reul et al., 2019] suggests that deskewing on smaller text regions can lead to better results than deskewing on the page level. This is because regions that skew independently from each other are frequently found in historical printings. These originate from either mistakes in the

printing process or degradation of the material.

Dewarping is the process of digitally manipulating an image such that distorted shapes are returned to normal. Distortion is defined in geometric optics as "a deviation from rectilinear projection; a projection in which straight lines in a scene remain straight in an image" [Wikipedia, 2019a].

Denoising or despeckling refers to removing noise from the signal, in our case an image. Some images contain *salt and pepper noise*, which are pixels in an image that are very different in color or intensity from their surrounding pixels. The other type of noise in images is *Gaussian noise*, where each pixel in an image is changed from its original value. [Wikipedia, 2019b]

Cropping the printing area is useful in cases when an image contains some background area that does not belong to the scanned page. Such an area can be confusing to both the segmentation and the recognition models.

Binarization is a method used to transform the original color or grayscale image into an image containing only black and white pixels. It is used in document processing to identify foreground pixels containing characters or ink and white pixels that belong to the document's background. The binarization is done by selecting an intensity threshold. Then each pixel in the image is set to either black or white, depending on whether it is darker or lighter than the selected threshold. The threshold is usually selected automatically for each image, either by selecting a single threshold on a document level (like in [Otsu, 1979]) or by creating a threshold for each pixel, by using a pixel's contextual information (like in [Sauvola and Pietikäinen, 2000]). The first approach is usually very fast but does not perform very well on documents where illumination is not uniform, like on scanned book pages. The second approach, on the other hand, performs very well but is often slow.

[Shafait et al., 2008] propose a binarization method that is as fast as the global binarization methods, with performance as good as that of local binarization schemes. It is based on Sauvola's local adaptive thresholding techniques, but by using integral images to compute mean and variance in local windows, they managed to produce an algorithm with fast running time. In their paper, they define an integral image of an input image as "the image in which the intensity at a pixel position is equal to the sum of the intensities of all the pixels above and to the left of that position in the original image". This adaptive thresholding approach is used for binarization in the OCRopus (and Ocropy) toolkit. For comparison, Tesseract uses a built-in Otsu global thresholding method.

However, to binarize color images, it is common to first transform them into grayscale. There are many different approaches to doing so, with varying results. For example, [Kanan and Cottrell, 2012] evaluate 13 different grayscale algorithms and investigate how they affect the image recognition algorithm. They find that the algorithm greatly affects recognition results, as well as that certain algorithms better suit different recognition tasks. However, in their work, the image recognition is done directly on the grayscale images, without a binarization step in between.

Also, the image recognition task is different from text recognition, so it is an open question if the same results would necessarily translate to the OCR task.

In recent work [Bouillon et al., 2019] propose a new *grayification* algorithm, which uses luminance and color information to improve the contrast between the foreground and the background, resulting in a grayscale image that better differentiates text from the background. This work is aimed specifically at improving analysis of handwritten historical documents, and the results show that when this method is used before binarization, it improves the results of the leading binarization methods. It works particularly well on historical documents with colored and faded text, which binarization methods usually struggle with. However, the study does not comment on the speed of the method, which is an important factor in mass digitization.

Although binarization has been a standard pre-processing step for OCR for a long time, the development of deep neural networks raises the question of whether it is still a necessary step. Deep neural networks can handle a lot more information than shallow ones, which means that, in theory, they should be able to successfully recognize grayscale images as well. Skipping the binarization would significantly speed up and simplify the OCR process, but the expected downside is that training the OCR models with grayscale images would require more training data. However, these assumptions need to be tested. As OCR with deep neural networks is quite recent, to the best of our knowledge, there has not yet been a study that explores how they perform on grayscale images in comparison to binarized ones.

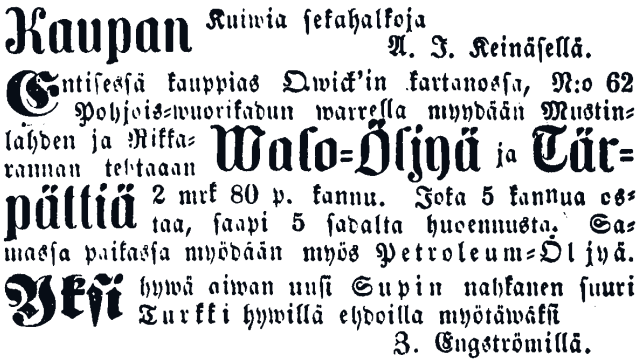


Figure 2.2: An extract from the binarized last page of *Tapio*, No. 47, 1866. It shows a case of difficult line segmentation inside a block.

2.2 Segmentation

Image segmentation is the process of dividing an image into smaller parts. In the OCR process, it is used to identify text regions from the non-text areas, and

further divide them into text lines or even glyphs, depending on the OCR algorithm. Optionally, non-text elements can be further classified (e.g. images, ornaments) and text blocks can be divided into semantic classes (e.g. running text, headings, marginalia) [Reul et al., 2019].

Segmentation of historical documents is a very difficult problem. Historical books often have marginalia, which, when very close to the text, frequently cannot be automatically recognized as a separate block [Reul et al., 2019]. Historical newspapers often changed their layout, with fluctuating number of columns and type of column separators, but were also very creative with the placement of article blocks. In both books and newspapers, we can often find drop caps, initial letters larger than the rest of the text that have proven to be difficult for segmentation tools.

All of the OCR frameworks include automatic segmentation tools. However, due to imperfect results, there are also tools for semi-automatic segmentation (e.g. Transkribus[1] and LAREX[2]), which allow users to achieve complete and accurate segmentation, which is crucial for successful OCR. The choice of an automatic versus a semi-automatic approach largely depends on the goal of the OCR. Since the semi-automatic approach is very time-consuming, it is not suitable for mass recognition of large collections. On the other hand, in order to produce quotable texts, high accuracy is a necessity, and in such projects, it is crucial to have human input at this stage. [Reul et al., 2019]

Figure 2.3 shows an example page from *Helsingin Sanomat*, No.20, 1904. This is a typical case of a page with difficult segmentation. Due to the various widths and sizes of the advertisement boxes on the left of the page, there is a high chance of segmentation mistakes. On the right, there is a running text printed in both Blackletter and Antiqua. The language used in the page is Finnish.

On the other hand, Figure 2.2 contains an extract from the binarized last page of *Tapio*, No. 47, 1866 (the whole page is shown in the next Chapter, in Figure 3.1). The words in large font that spread over two lines of text make it difficult to segment lines and find the correct reading order.

Since segmentation is still an open research problem that requires expertise in image processing and computer vision, in our work we used already existing segmentation produced with Abbyy Finereader 11 and focused on other parts of the OCR workflow, namely training data creation, OCR model training and post-correction.

2.3 Training OCR models

Training is crucial for the success of the OCR process, especially for historical documents. Since historical printings have a large variety of fonts, it is necessary to train models specific to a certain data set. Using pre-trained models for a specific

[1] https://transkribus.eu/Transkribus/

[2] http://www.is.informatik.uni-wuerzburg.de/research-tools-download/larex/

2 — N:o 20 — 1904 HELSINGIN SANOMAT Sunnuntaina lokakuun 23 p.

Jonas Castrénin Asiamajatoimisto Helsingissä Brottsg. 4

N. Bomanin Höyrypuuseppätehdas, TURKU.

Näyttelyhuoneustot: Helsinki, Mikonk. 4. Turku, Linnank. 47.

Huonekaluja, Mattoja, Uutimia y. m.

Varatuomari Yrjö Pulkkisen Lakiasiaintoimisto

Sähkökylpyjä

Privatbanken Helsingissä.

Pohjoismaiden Osakepankki.

Perustettu 1872. Omia rahoja 19,500,000.

Uusia Sävellyksiä.

Maitoa

Höyrylaivoillike.

Ebba Munck

Nord I

Huveja.

Työväenteatteri

Ruotsalainen teatteri.

Folkteatteri.

Pasi Jääskeläinen

Laulu- ja Kanteleekonsertin

Anna Korrie

Laulu-Illan

Eläviä kuvia

Maailman ympäri

Knut Dyblom

Laulu-Illan

Ilta man

Seurahuone.

Konsertti ja Näytäntö

Matkustajat!

Helsingin Sanomat

MINERVA

XI Yleinen luterilainen konferenssi Rostockista.

Helsingin Sanomille.

II.

Ruotsalainen puolue ja kansanvaltaisuus.

Yleisen työt vuonna 1905

Helsinki, lokak. 23 p.

Figure 2.3: The second page of *Helsingin Sanomat*, No.20, 1904.

font and language usually yields poor results when used on a different font or language. For example, models trained on German Blackletter material result in

under 90 % CAR on Finnish Blackletter.

For a long time, models were trained to recognize one glyph at a time. This approach is problematic for two reasons: First, in historical printings, it is often very difficult to segment each glyph due to the aging of the material. Glyphs are either segmented into smaller pieces when the original contours have faded in places, or the contours of the neighboring glyphs have become fuzzy, resulting in two or more glyphs recognized as one. Secondly, it is time-consuming and demanding to create training data on a glyph level for historical fonts. There are tools that create synthetic training data once examples have been identified for each glyph class, but this approach still requires too much manual work to be practical. [Reul et al., 2019]

As mentioned earlier in Chapter 1, [Breuel et al., 2013] introduced an approach that used lines of text for training and subsequently recognition. They used a recurrent neural network with an LSTM architecture together with the CTC loss function, which allowed a sequence of images to be used as input to the network (the input line image is considered a sequence of pixel-length images), and a sequence of characters as the output. This approach has proven to provide better results than the glyph based methods, and has made generation of the training data much easier.

Recently, the line-based approach was further improved by [Breuel, 2017], who got better results with CNN layers added to the neural network. The new CNN/LSTM-hybrid method, with deep neural architecture, has turned out to be highly successful in recognizing historical data and is considered the current state-of-the-art.

While choosing the most efficient approach for training is essential for successful OCR, there are other factors that strongly influence the accuracy of the final results, particularly the choice and size of the training data and the architecture of the neural network. While the choice of the training data is closely connected to the final goal of the OCR project (like with segmentation, projects aiming at highly accurate results might want to invest more time and effort into fine-tuning models for a specific type of text, for example for each book), the necessary neural network depth is determined by the size and type of the data. There are also additional features that can be helpful for achieving high accuracy results, like early stopping, data augmentation, and voting.

Different training methods and approaches are presented in more detail in Chapter 5.

2.4 Recognition and text extraction

The recognition is performed by one or more trained models, after which the result, a collection of plain text lines, is collected either in a plain text file with the correct reading order, or into a more complex data format like XML (Extensible Markup Language) or, of late, the popular JSON (JavaScript Object Notation)

files. Those files usually contain, along with the text, layout information (coordinates and semantic classes of each segment), a confidence measure, and information about languages and fonts used in the document.

The most popular formats used for storing the OCR results are hOCR, ALTO and abbyyXML.

hOCR is an open standard for OCR data representation introduced in [Breuel, 2007]. It uses a combination of XML, HTML (Hypertext Markup Language) and CSS (Cascading Style Sheets) to collect and represent the OCR data in one file.

ALTO[3] (Analyzed Layout and Text Object) is an open XML Schema developed by the EU-funded project METAe. It is commonly used to describe the layout and content of textual materials, like pages of books or newspapers. Often it is used together with a METS (Metadata Encoding and Transmission Standard) file for the description of the entire digitized object (i.e. the entire newspaper, or a book) and creation of references across the ALTO files, each of which represents one page at a time.

AbbyyXML[4] is an XML structure that ABBYY FineReader uses as one of the OCR output formats.

2.5 Post-Processing

Historically, the motivation for the development of the OCR post-correction methods [Englmeier et al., 2019] has come from the inability of OCR methods to produce sufficiently good results for historical documents (WAR < 80 %). In the post-correction stage, the OCRed text is processed with the aim of fixing OCR mistakes, using methods that rely on language properties. This step can be performed immediately after recognition and before the creation of the output XML, where ideally both OCR and post-correction outputs would be preserved, or it can be done at a later stage. Some tools (e.g. Tesseract) have integrated recognition and post-correction steps.

There are many different post-correction approaches, but all of them work on the same principle. First, they recognize possible mistakes and propose correction candidates. Secondly, they need to decide whether to perform the change or not. While some approaches divide these two steps, other methods combine them (see Chapter 6).

The post-correction can be done manually, semi-automatically or fully automatically. Fully automatic post-processing is typically performed considering only the OCRed text without reference to the original image, whereas both manual and semi-automatic approaches usually provide an interface where text and the original image are displayed side by side. In the semi-automatic approach, the tool usually

[3] https://www.loc.gov/standards/alto/

[4] https://www.ocrsdk.com/documentation/specifications/xml-scheme-recognized-document/

points out probable errors and can make suggestions for correction, but then human expertise and language understanding is used in making the decision. Both approaches, since they require human input, are slow and expensive, and humans also make mistakes. According to [Evershed and Fitch, 2014], humans can easily miss graphically similar characters, like i, l, I, 1, 0, O, o, e, c. This can lead to the false perception that due to human effort the text is fully corrected, while there might still be errors left uncaught.

On the other hand, there are fully automated approaches. These are usually complex, specialized methods taking advantage of language modeling and OCR properties (for example common confusions). Some systems use lexicons and create language models when additional data is available. However, there are also approaches using only OCR data and the ground truth used for training OCR models.

Different post-correction methods and approaches are presented in more detail in Chapter 6.

Chapter 3

Data

We are working on the corpus of historical newspapers and journals published in Finland 1771 - 1939. The corpus has been digitized by the National Library of Finland (NLF) and OCRed with ABBYY FineReader (versions 6 - 11). Part of the corpus (1771-1910) has been released for public use[1], part was open from late 2018 until the end of 2020 (years 1918 - 1929). However, all of the data is available for academic research thanks to an agreement between the FIN-CLARIN research infrastructure, NLF and Kopiosto. FIN-CLARIN is providing access to the data via Korp (textual version, search, access images), and a full version[2] via The Language Bank of Finland[3].

The data from that time period is written in a non-standard Finnish. The standardization of the Finnish literary language began in the early 19th century, hence a large part of the corpus that we are working on belongs to the Early Modern Finnish period, which lasted from the early 19th century until the 1890s. In the early years of the standardization process, there was disagreement on whether the standard language should be based on the eastern or western Finnish dialects, and at that time both variants were used in publications. In the second part of the 19th century, academics and writers gradually settled for a compromise where they used mostly the Western idiom but incorporated many elements from the East. [Kauppinen, 2016]

The corpus is diverse in languages and fonts. Apart from Finnish, the data is in large part written also in Swedish, which was the only official language in Finland from the 16th century until 1863, when Finnish attained the status of official language along with Swedish. There are also other languages present to a smaller extent, like Russian, German, English, Latin, etc. Furthermore, the corpus was printed with fonts from both the Blackletter and Antiqua font-families and the fonts and languages are not uniformly divided. Although the earlier data is more often written in Swedish and printed with Blackletter fonts, and the later data

[1] https://digi.kansalliskirjasto.fi

[2] The full version with the original scans is around 50TB in size

[3] https://www.kielipankki.fi/language-bank/

in Finnish with Antiqua fonts, there is a period when both languages and both font-families were used in the same newspapers and even on the same pages.

Figure 3.1 shows an example page from the newspaper *Tapio*, issue No. 47, published in 1866. The page contains both Finnish and Swedish texts. Swedish texts are printed in Antiqua and Finnish in Blackletter fonts. The rest of this newspaper issue contains mostly Finnish text, dominantly printed in Blackletter, with occasional sections printed in Antiqua.

In the rest of this chapter, we present the methods used in acquiring training and testing data. Furthermore, we describe the three data sets that we used in our experiments: Finnish, Swedish, and the one provided by the NLF.

3.1 Acquiring the Data

The corpus of the historical newspapers and journals published in Finland contains OCR results from ABBYY FineReader 11, provided in the METS-ALTO format.

ABBYY FineReader 11 OCR results are of low quality, with CAR between 87 % and 92 %. However, we decided to use ABBYY's segmentation results for acquiring training and testing data. Since all the training and testing data needed to be manually inspected during transcription of the training and test data, we could remove the lines that were segmented in such a way that humans were unable to recognize the text.

The main idea when acquiring the data is to obtain diverse data from the entire corpus. To achieve this, we randomly selected lines of text from the entire corpus, which proved to be superior to the method used by the NLF (described in [Koistinen et al., 2017a] and [Koistinen et al., 2017b]), where they selected 500 full pages of Finnish Blackletter text, of which they used half of the pages for training and the other half for testing. In Publication **I**, we show that with training on around 9,000 lines of Finnish text randomly selected from the corpus, we get better results than when training on more than 50,000 lines segmented from 255 Blackletter pages of the NLF data.

The randomized representation of the corpus is essential for creating a good general model because we then have diverse training data, and can train a model that is able to recognize data across the the corpus. However, when we want to make sure to have enough training data of less frequent fonts and languages as well, we need to acquire more specific data. In such cases, we can use additional tools to automatically identify the data that we are searching for.

To make sure that we have enough training lines of each font-family, we used the font-family recognizer developed in Publication **IV**, which determines whether the line of text is printed in the Blackletter or Antiqua font-family. Although some tools that identify different fonts exist, none of them were ready to use for purpose of font-family recognition. Therefore, it was easier to write a simple neural binary classifier and train it on our data, than to try to figure out how to modify existing tools to

tapahtuneesta waltiopäiwä-miehen waalista, niin kutsutaan ne porwarit jotka eiwät ole waltiuffiesta osalliset kokoontuman Maanantaina 28 pnä tätä kuuta, klo 12 päiwällä woetaksensa ottamaan mainitut waltiuskirjat. Kuopion Raatihuoneesta 20 p. Marraskuuta 1866.

Maistraatin puolesta:
August Westzynthius.

Että Häwen Keisarillinen Majesteettinsa 5 p. wiimeksikuluwuta Lokakuuta antaman armollisen päätöksen kautta estettyötä syystä on päätetyt Herra Läänin Kuwernöörin 6 p. Toukokuuta 1865 antaman säännön, jonka mukaan torikauppa täsä kaupungissa toistaiseksi saatasiin pidettää wuotta torilla, eikun della myöskä maatawaroita kaupungin Kontaritsillakin sillä aikaa wuodesta, jolloin meritiliestä pidetään, laillisen woiman woittaneeksi ja niinmuodoin on seuraawa; se olkoon täten tiedoksi annettu. Kuopion Raatihuoneesta 20 p Marrask. 1866.

Maistraatin puolesta:
August Westzynthius.

Tulewan wuoden henkipano Kuopion kaupungin kansfa tapahtuu Raatihuoneessa seuraawina päiwinä Joulukuusa, nimittäin:

Wanhan kaupungin kansfa: Maanantaina 3 p. klo $^1/_2$4 j. pp. tomtuilta N:o 10—30.
Tiistaina 4 p. klo 9:stä e. pp. tomtuilta N:o 31—41 ja klo $^1/_2$4 j. pp. N:o 42—54.
Keskimäkkona 5 p. klo 9:stä e. pp. tomtuilta N:o 55—67 ja j. pp. N:o 68—84.
Torstaina 6 p. klo 9:stä e. pp. tomtuilta N:o 85—94 ja j. pp. N:o 95—108.
Perjantaina 7 p. klo 9:stä e. pp. tomtuilta N:o 109 wanhan kaupungin loppuun asti, ja j. pp. uuden kaupungin kansfa N:o 1—220.
Lauwantaina 8 p. klo $^1/_2$4 j. pp. N:o 221 loppuun asti, niin myös kuin arenti maisin kansfa.

Ja muistutetaan talon haltioita, että yllä nimitettyinä päiwinä ei ainoastaan ylös antaa omaa perhettänsä, mutta myös ilmoittaa hyyriläisensäkin, joiden, jos isäntä ei tahdo niitä perheensä kirjaan ylösottaa, itse tulee niitä ylösantaa laillisen sakon haastolla. Muutto-kirjatoisesta seurakunnasta muuttuwalle pitää edestuotaman, jos kirja ei jo ole pappilaan heitetty ja myös selittää, minne entinen palwelia on muuttanna.

Koska myös ala-ikäiset lapset nyt tulewat henkikirjoin nimellisesti ylöskotettawaksi, muistutetaan, että niitä nimeltään ylösantaa, niinmyös kuin lapsen syntymä-wuosi.

Hyyri kontrahtia nyt ei tarwitse näyttää, mutta sitä wastaan talollisten, joilla on hyyriläisiä, tulee ilmoittaa, miltä numerolta ne owat sinne muuttaneet, joka tämän kautta tiettäwäksi annetaan. Kuopiossa 15 p. Marrask. 1866.

Carl Aug. Malmberg.

Att Hans Kejserliga Majestät, medelst nådigt utslag af den 5 sistlidne Oktober, på anförda skäl, förklarat Herr Guvernörens öfver länet den 6 Maj 1865 meddelade förordnande, hvarigenom torghandeln i denna stad tillsvidare finge bibehållas å nytorget, med rättighet till försäljning af landtmanna varor å stadens strandtorg den tid af året, som sjökommunikationen fortginge, för laga kraft vunnit och förty böra lända till efterrättelse; derom varder härmedelst till kännedom meddeladt. Kuopio Rådhus, den 20 November 1866.

På Magistratens wägnar:
August Westzynthius.

Sedan några af denna stads borgerskap hos Herr Guvernören öfver länet anfört besvär öfver det inför Magistraten härstädes den 10 i denna månad anstälde val af Riksdagsfullmäktig för Kuopio stad, behagade de i besvären icke deltagande borgare sammanträda å Magistraten måndagen den 26 i denna månad klockan tolf före middagen, för att till bemötande emottagn nämnde besvär. Kuopio Rådhus den 20 November 1866.

På Magistratens wägnar:
August Westzynthius.

Yksityisiä ilmoituksia.

Älytkää!

Kuin allakirjoittaja kadotti kuoleman kautta kumppaninsa kirjapaino-yhtiössä, Tirehtööri A. Mannisen, niin saan täten ilmoittaa, että mainittuun yhtiöön Kuopiossa olisi saisuus tarjona, jos kuka siihen haluaisi ruweta. Laweampia tietoja antaa allekirjoittaja joko kirjeellisesti eli suullisesti, jos kuka haluaa asiaan tarttua. Kuopiossa Marraskuussa 1866.

Fr. Ahlqwist.

Nöyrimmästi pyydetään että muutkin sanomalehdet hywäntahtoisesti julkaisisiwat tämän yllä olewan ilmoituksen.

Kuopion Läänin Maawiljelysseuran Johtokunta kokoontukoon Maanantaisoisa Tiistaina tämän kuun 27 pnä, klo 5 j. pp.

E. Nylander.

Porwarillisen Laulu-seuran

jäsenet olisiwat hywät ja kokoontuisiwat huomenna 25 p. t. k. yleisen kokouksen työwäen poikasten koululla tuumimaan seuraan kokewista aseista nymä maksamattomat suorittamaan jäsen rahansa. Johtokunnan nöyrin toiwo olisi myös että reumanpäiwän olisi hywä ja lähäsoleltaan tuomittaisi tätä seuran kokousten tarkoitusta. Kuopiossa 23 p. Marrask. 1866.

Johtokunta.

Maanantaina 3 p. Joulukuuta antaa allakirjoittaja wiimeisen laulantonsa.

B. B. Broms.

Kakfi taitawaa naulaseppää saawat hyödyttään paikan, jos sitteläan joko kirjallisesti eli suullisesti ilmoittawat allakirjoittajalle, nimiosoitteella: Oulu, Westfoeli.

Henrik Sjöberg.

Suomierowastin ja kaupungin kauppalaisen saatawia kaupungista kantaa tänä wuotena allakirjoittaja, jota asua tullipoitaja Sahlsteinin talosja seuraawina päiwinä ja seuraawasta järjestyksestä:

Maanantaina 10 päiwä tulepaa Joulukuuta e. pp. tomtuilta 1—50, ja j. pp. tomtuilta 51—83.

Tiistaina 11 p. Joulut tomt. 84—120, ja j. pp. tomt. 120 loppuun asti kuin myös unbelta kaupungilta. Kuopiosa Marraskuun 24 p. 1866.

P. J. Igoni.

Joka on ecyplsesså ottanut menneellä wiikolla Tiistaina Raatimies Dahlströmin työwäen saunopuolella ydeen maria-finnbohjan pumpuli hameen walkeilla langoilla kerrattu, yhi tanu pienempi toimn suuremmi, olisi hywä ja teisi sen takaisamaan taloon muutari Johan Hämäläiselle.

Allakirjoittaja ostaa rutkita, jos kuka haluaa niitä myödä. Kuopiosa Marrask. 1866.

E. Grönn.

Se joka tietää waihtanensa huonemmilla paremmat kumni talosti tiwaetta kauppa paikasa tällä wiitella, pyytetään erepe eikasemaan wiimeistään huonemittana samasta paikasa.

Kaupan Kuiwia sekahalkoja

A. J. Keinäsellä.

Entisessä kauppias Dwick'in kartanosa, N:o 62 Pohjois-wenerrantan warrella myydään Muutingsaiden ja Hiffa- **Walo-Öljyä** ja **Tärpättiä** 2 mrk 80 p. kannu. Joka 5 kannua ostaa, saapi 5 salaita huomausto. Samasta paikasa myödään myös Petroleum-Öljyä.

Hyfi hywä oiwan uusi Supin nahkanen suuri Turkki hywillä ehdoilla myödäwäksi

J. Engströmillä.

AVERTISSEMENT.

Tisdagen den 4 December 1866 anställes af undertecknad den andra subscriberade **Musikaliska Solrée meddans.**

J. D. Strunck.

SKANDIA

Lif- och Brandförsäkrings-Bolag

i Stockholm,

emottager försäkringar å lif, på vissa år eller lifstid, med eller utan andel i Bolagets vinst å lifförsäkringsrörelsen, — och afslutar försäkringar mot brandskada å fast- och lösegendom i städer och på landet, — allt till fastställda moderata premier.

Ansökningar emottagas af undertecknad som med beredvillighet, såväl mundtligen som skriftligen, meddelar alla upplysningar rörande försäkringar. Kuopio i Februari 1866.

Alexis Ström.
Drottningsgatan N:o 42.

Måndagen den 3 December 1866, gifver undertecknad sin sista KONSERT.

B. B. Broms.

I fastställd ordning sammanträder pädagogiska föreningen härstädes till årsmöte Onsdagen den 5 nästkommande December kl. VI e. m., hvarvid val af ledamöter i Bestyrelsen för år 1867 skall försiggå och föreningens räkenskaper granskas. Till öfverläggning framställes derefter följande vid nästhållna möte bestämda frågor: 1:o *är det lämpligt, att fruntimmer förordnas till lärare vid gosskolor?* 2:o *är med afseende å nu gällande befordringslagar för lärare något att påminna?* Kuopio den 23 November, 1866.

P. Aschan.

Tvenne dugliga Spiksmeder finna förmånlig anställning om de med det snaraste skriftligen eller mundtligen anmäla sig hos undertecknad, adress Uleåborg & Wesikoski-bruk.

Henric Sjöberg.

Undertecknad boende i arbetskarl Närhis gård emottager och förfärdigar allehanda fruntimmers arbeten.

M. Durchman.

Uti Kellgrenska Ölbryggeriet i Kuopio försäljes Svagdricka à 6 penni p:r kanna, då minst 1 kanna köpes.

North British and Mercantile

Insurance Company
London och Edinburgh,

stiftadt 1809,

konstitueradt under kungligt privilegium och särskilda parlamentsakter.

Tecknadt grundkapital p.steri.		2,000,000
Reservfonden	"	2,233,927 17 / 7 d.
motsvarande tillsammans omkring	S. Rb.	29,500,000.

Undertecknade, hvilka öfvertagit General-Agenturen för Finland, äro befullmäktigade att afsluta försäkringar mot brandskada på fast- och lösegen dom till de billigaste premier.

Förluster regleras promt och liberalt.

Wiborg i Juni 1863. Hackman & Co.

I och för försäkringar uti ***North British & Mercantile Insurance Company*** o prospektälles ch andar upplysningar hoser erh undertecknade agenter, såväl som blanketter till försäkringsanmälningar.

Fredrikshamn	herr	O. L. Lydecken.
Björneborg	„	Georg Wentzell.
Uleåborg	„	M. E. Ringvall.
Tammerfors	„	A. Törngrén.
Kuopio	„	S. Kellgrén.
Joensuu	„	Joh. Wilh. Åkerstedt.
Nikolaistad	„	Carl Gustaf Wolffs Söner.
Åbo	„	G. W. Wilén.
Nykarleby	„	
Jakobstad	„	C. J. Thodén.
Gamlakarleby	„	
Willmanstrand	„	C. Lilius.
Borgå	„	Aug Eklof.
Ekenäs	„	C. G. Lydman.
Helsingfors	„	Carl Fredr. Blomberg.

The Northern Assurance Company

Eld- och lifförsäkringsbolag i London.

emottager fortfarande härstädes genom undertecknad ansökningar om försäkring af lif och egendom, såväl fast som lös, emot särdeles billiga premier som numera bestämmas af mig efter sig företeende omständigheter och meddelas att försäkring tager sin början från den dag premien blifvit till mig inbetald. Anmälnings-blanketter och närmare underrättelser erhållas med beredvillighet af

Fernand Öhmann,
agent i Kuopio.

Till salu: En ny två-sittsig släda hos Sadelmakar Törnqvist.

Kuopiossa,
A. Mannisen perillisten ja Fr. Ahlqwistin kirjapainossa, 1866.

Figure 3.1: The last page from *Tapio*, No. 47, 1866. The page contains Blackletter and Antiqua fonts, as well as both Finnish and Swedish text.

perform the font-family classification task. When working with the clasifier, we were able to analyze already collected data and acquire specific training data that we did not have enough of (in our case, we had to acquire additional Finnish Antiqua lines, which were present in only 25 % of randomly sampled lines). To obtain lines written in a specific language, we used the HeLI language identification software developed by [Jauhiainen et al., 2016]. The software proved to work surprisingly well even on the OCRed text of not very high quality.

The detailed process of acquiring the training and testing data is described in Publications **I** - **III**.

Once when we had acquired the selected line images, we needed to transcribe them to create the ground truth (GT). Transcription can be performed using existing software, usually provided with OCR frameworks. For example, Ocropy has a tool for transcribing line images using an internet browser. However, we developed our own python tool for transcription. It reads and displays a line image and the corresponding OCR text together, as well as an editable line that contains OCRed text that should be corrected for the GT.

Transcription is usually performed in several steps. When starting from zero, it is common to first transcribe a small number of lines which are then used to train an OCR model. That model is then used to OCR the next set of lines for transcription, after which the OCRed text can be used as the starting point for creating the GT. The other method is to use an existing model trained on similar data to transcribe the first lines. This approach is becoming increasingly viable as the number of historical OCR models available is growing. Since we already had the ABBYY's OCR results, we used them to transcribe the lines and train our own first models.

3.2 Finnish Data Set

The first instance of the Finnish data set, created and described in Publication **I**, consists of 9,345 training, 1,038 development and 2,046 test lines with the corresponding ground truth. The lines were randomly picked from the years 1820 - 1939. We denote this training set as *fin-9k*, and the test set as *Test-2k*. In Publication **II** we performed test training on the smaller parts of the same training set and we found models trained on 7,000 lines with Ocropy particularly successful. We call this smaller training set *fin-7k*.

In Publication **II**, we randomly picked 418 lines from the *Test-2k* test set and showed that we can perform tests on this smaller test set without compromising the validity of the results. We manually analyzed this test set to find out that roughly 75 % of the test data is printed in Blackletter fonts and about 25 % in Antiqua, which later proved to be the rule for all the randomly picked Finnish data sets. Since we can see with this test set how the models perform on Blackletter and on Antiqua lines, we call it the *Fine-grained test set*. To make it easier to compare

results, we use the same *Fine-grained test set* in Publication **III** as well.

Furthermore, in Publication **III** we add 4,000 Finnish Antiqua lines to create balanced training and testing sets with an equal number of Blackletter and Antiqua lines. We call this Finnish training set *fin-13k*. We use the data set in 5-Fold Cross Validation and get an average result of five *Balanced test sets*.

Finally, the Finnish data set consists of a total of approximately 13,500 lines with the same ratio of lines printed in Blackletter and Antiqua fonts, and additional fine-grained 418 testing lines.

In Table 3.1, we show in detail the Finnish training and test sets. For each data set we print the number of lines, words and characters, as well as the Blackletter and Antiqua ratio.

Table 3.1: Finnish training and test sets. B:A denotes the Blackletter:Antiqua ratio.

	data-set	# lines	# words	# chars	B:A
training	fin-7k	7,000	30,067	274,181	75% : 25%
	fin-9k	9,345	40,047	364,708	75% : 25%
	fin-13k	13,037	54,941	513,987	50% : 50%
testing	Test-2k	2,046	8,795	81,757	75% : 25%
	Fine-grained	418	1,756	16,622	75% : 25%
	Balanced (1)	1,303	5,575	52,472	50% : 50%

3.3 Swedish Data Set

We started the development of the Swedish data set in Publication **II** when we harvested a total of 6,995 line images and created the corresponding GT. The images were collected from the period 1771 - 1874 when Swedish was still frequently used in the publications. As per typographical division, the Swedish data set consists of about 50 % Blackletter and 50 % Antiqua fonts. At this point, we also created a Swedish test set, consisting of 418 lines, that we used exclusively for testing in both Publication **II** and Publication **III**. The test set is divided into Blackletter and Antiqua subsets, so we call it the *Fine-grained test set*.

In Publication **III**, we expanded the Swedish data set by 5,000 lines, resulting in a total of 11,000 image lines with corresponding GT. We use this large training set for 5-fold Cross Validation, creating 5 distinct *Balanced test sets*.

In Table 3.2, we show the Swedish training and test sets in detail. We used smaller subsets of the training data in different tests, namely *swe-3k*, *swe-6k*, *swe-9k* and *swe-11k*. In the table, we show the number of lines, words, and characters, as well as the Blackletter and Antiqua ratio for each training and test set.

Table 3.2: Swedish training and test sets

	data-set	# lines	# words	# chars	B:A
training	swe-3k	3,351	19,115	144,352	50% : 50%
	swe-6k	6,159	35,288	265,397	50% : 50%
	swe-9k	9,111	52,095	392,021	50% : 50%
	swe-11k	11,094	63,561	478054,	50% : 50%
testing	Fine-grained	418	2,357	17,621	50% : 50%
	Balanced (1)	1,109	6,368	47,791	50% : 50%

3.4 The National Library of Finland's Data Set

The National Library of Finland provided us with 255 pages of transcribed Finnish Blackletter text segmented in 54,087 lines of text. We used this data set in Publication **I** to train and test models with Ocropy, and compare results with the results achieved with models trained on our Finnish data. We also present some of those results in Chapter 5, where we refer to this data set as *nlf*. In Publication **I**, we concluded that we get better results with our data set, therefore we abandoned using this data-set in future work.

Chapter 4

Evaluation

The OCR and post-correction systems are most commonly evaluated using accuracy (or error) rate measures, which calculate a percentage of correct (or incorrect) characters, words or even lines produced by the system. We address these measures in Section 4.1.

Since post-correction systems can both correct OCR errors and produce new errors, to more efficiently evaluate and understand their effect on the OCR results, they are also frequently evaluated with Precision, Recall and F-score measures, which we describe in Section 4.2.

N-fold cross validation is a method commonly used in machine learning to evaluate how neural models behave on different test sets from the data they were trained on. We outline the method in Section 4.3.

The methods listed above are useful for assessing the overall performance of the system, but they do not tell us what sorts of errors models make or how often they occur. To analyze the OCR errors, researchers often use confusion matrices, which we present in Section 4.4. Furthermore, we find useful the line level analysis that we performed in Publication **III** and briefly address here in Section 4.5.

Finally, in Section 4.6 we describe and discuss the methods used to evaluate OCR performance on large collections where GT is not available.

4.1 Accuracy and Error Rates

The OCR systems are evaluated using accuracy and error measurements, either on the character or word level. Those measurements are known as Character Accuracy Rate (CAR), Word Accuracy Rate (WAR), Character Error Rate (CER) and Word Error Rate (WER). In Publications **I** and **III** we also use Line Accuracy Rate (LAR) when we calculate how many lines were completely correct.

The accuracy rates are the percentage of correct tokens in the system output, while error rates are the percentage of errors in the system output. The accuracy rate is calculated as the sum of the number of correct tokens divided by the sum of

correct tokens and errors in the system output:

$$CAR, WAR = \frac{correct}{correct + errors} \cdot 100 \tag{4.1}$$

Similarly, the error rate is calculated as the sum of the number of errors (either on character or word level) divided by the sum of correct tokens and errors in the system output. Again, to get the measure in the percentage form, we multiply it by 100:

$$CER, WER = \frac{errors}{correct + errors} \cdot 100 \tag{4.2}$$

While there is no difference in expressing the results either as an accuracy or error measure (one can easily transform the results between the two by simply subtracting from 100 %), character measures are more straightforward than word-level measures and are better when comparing result outputs from different OCR systems or the results on different test sets.

As we present in Publication **III**, different alignment techniques can influence results when calculating measures on the word level. Some alignment systems use a white-space as a word separator on both OCR and GT level, which can be problematic when the OCR system wrongly produces a white-space. For example, if a word *miracle* is wrongly OCRed as *mira le*, some systems might count this as two word-errors, although only one word is erroneous. They would align *mira* to *miracle* and count this as one error, but they would also recognize *le* as an extra word and add it to the error count. On the other hand, other systems, like the one we use for evaluation in Publications **II** and **III**, first align lines on the character level, inserting an empty string where necessary. Then it is possible to align words using the GT line as a starting point and get a more accurate word error count. This difference between the evaluation systems is particularly noticeable on lines with low accuracy due to poor image quality. In those cases, the word error count can differ drastically with different evaluation systems.

However, word-level measures are commonly used in evaluating the benefits of applying post-correction methods to the OCR results. The idea behind post-processing has been to improve the usability of the OCR results, which often means correcting erroneous words. Therefore, researchers in the post-processing field usually want to report how much the post-processing has improved the OCR results on the word level, presuming that the information on the fully correct words better describes the usability of the corpus.

4.2 Precision, Recall and F-score

When post-correcting OCR results, post-correction systems can, in addition to correcting erroneous words, produce new errors. For example, post-correction systems

that rely on lexicons and dictionaries can replace a correct word with a misspelling or another historical variant.

Since accuracy or error rates do not give us information on such cases, post-correction systems often use precision, recall, and F-score [Manning et al., 1999] to better describe their results. The same evaluation method is also commonly used with binary classification, therefore we use it to evaluate the font-family classifier in Publication **IV**.

For the binary classifier in Publication **IV**, we define True Positive (TP) as correctly predicted Blackletter line, False Positive (FP) as incorrectly predicted Blackletter line, True Negative (TN) as correctly predicted Antiqua line and False Negative (FN) as incorrectly predicted Antiqua line.

In case of the post-correction,[Kettunen, 2015] define TP as a wrongly spelled word that was corrected, FP as a correct word that was changed to a misspelling, FN as a wrongly spelled word that is still wrong after the correction and TN as a correct word that is still correct after correction.

Precision, recall and F-score can be calculated as shown in Equations 4.3 - 4.5:

$$Precision = \frac{TP}{TP + FP} \tag{4.3}$$

$$Recall = \frac{TP}{TP + FN} \tag{4.4}$$

$$F\text{-}score = 2 \cdot \frac{Precision \cdot Recall}{Precision + tRecall} \tag{4.5}$$

4.3 N-fold Cross-Validation

When evaluating the performance of the OCR models, it is good practice to test models on different test sets. Creation of GT data is expensive and researchers usually want to use as much data as possible for the training, so instead of having several data sets that would be used just for testing, researchers usually use different parts of the entire data set for testing, while training models on the rest of the data. In this way, we can see how the neural network behaves when trained and tested on different data, without creating additional GT data. This process is called N-fold Cross-Validation.

In N-fold cross-validation, the data is usually divided into three parts. The biggest chunk, consisting of X % data (if you use X=80 [=4/5], you get 5-fold cross-validation), is used for training, while the rest of the data is divided equally into two small sets that are used for validation and testing. To perform *N-fold* cross-validation, this division is repeated N times, each time with different parts of the original set assigned to the validation and test sets. Although N can be any natural number (smaller than the size of the data set), the most commonly used are 5-fold and 10-fold cross-validation.

Finally, to get an accuracy measure from N tests, we can calculate the mean accuracy and the sample standard deviation over all the test results and estimate a confidence interval of 95 % (statistical significance level of 5 %).

4.4 Confusion Matrices

In many papers, researchers use confusion matrices to show the most common confusions made by an OCR model, together with the number of occurrences of each confusion. The idea behind this approach is to identify the common confusions, but also to see how widely the errors are distributed across the test set. In cases where a small set of confusions accounts for a large number of errors, those can be addressed by, for example, adding specific training data, or focusing post-correction on those cases. However, if a confusion matrix shows that the errors are widely distributed across the test set, we need to use another error analysis method to get more information about the nature of the mistakes.

We calculate confusion matrices in Publications **I** - **III** using Ocropy's error analysis tool to find that the 10 most common confusions cover only 0.3 % (Publication **III**) and 0.8 % (Publications **I** and **II**) of the total characters in the test set. Therefore, in our case, confusion matrices proved to be fruitless as an error analysis approach.

Table 4.1: Distribution of errors across lines on one Finnish balanced test set from Publication **III**. The test set has in total 1303 lines, of which 973 lines (75 %) are completely correct. The first column shows the number of errors per line, the second number of lines containing the corresponding amount of errors, and the third column percentage of those lines in the whole test set.

No. errors per line	No. lines	% of total lines
0	973	75 %
1	190	15 %
2	76	6 %
3	31	2 %
4	13	0.9 %
5	4	0.3 %
...	...	...

4.5 Analysis on a Line Level

When CAR reaches very high levels, most of the data is correctly recognized, which also translates to the line level. For example, in Publication **I**, CAR on the Finnish test set was around 93.5 % and LAR was only 37 %. In Publication **III**, CAR on

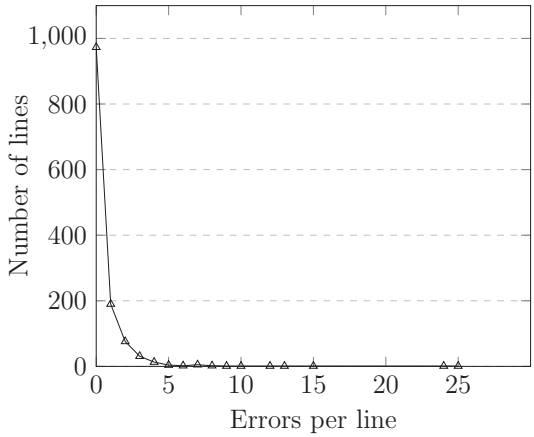


Figure 4.1: Graph showing the error distribution across all lines in a Finnish balanced test from Publication **III**. The x-axis shows the number of errors per line and the y-axis number of lines.

the Finnish test set increased to 98 % range and LAR was over 70 %. Interestingly, the line analysis showed that the errors are not evenly distributed across lines. In Publication **III**, we noticed that most of the lines were either correct or had very few errors, but there were also lines with a high number of errors.

In Table 4.1 and Figure 4.1, we show the error distribution over lines in the 1303-line Finnish balanced test set from Publication **III**. The table shows line counts and percentages for up to 5 mistakes per line, while the graph shows the full distribution. From the table we can see that 973 lines (75 %) had zero errors, 190 lines (15 %) had one error, 76 lines (6 %) had two errors and so on. If we add up the percentages, we can see that only 2 % of the lines from the test set had more than 3 errors per line, and less than 1 % of the lines had more than 5 errors per line. The graph shows the long tail that corresponds to the lines with a large number of errors, reaching even 25 errors per line.

By manually analyzing lines with a large number of errors in Publication **III**, apart from lines that were wrongly segmented or of poor image quality, we found that many of them contain headings and advertisement text printed in specialized fonts. To confirm our speculation that those lines are generally very difficult to recognize, we performed tests on the *Fine-grained test sets*, from which we removed all the lines containing heading and advertisement texts, and found that the results on the trimmed tests were always better than on the original ones. We concluded that we should be able to further increase the OCR accuracy by adding more training data with specialized fonts. However, this needs to be tested in future work.

4.6 Evaluation of Large Digitized Collections

Evaluation of a large digitized collection can give an estimate of the quality of the OCRed text. This information is useful in determining whether the corpora is of sufficiently good quality for research, or if it should be further processed (for example, re-OCRed or post-corrected). There are a couple of different approaches to evaluating a large collection, but they all express results on a word level, assuming that the researchers are interested in the whole words as tokens (for example, they might rely on word frequencies, or examine historical variants of words). In [Tanner et al., 2009], they even focus on evaluating "significant words", which they define as "those content words for which users might be interested in searching, not the very common function words such as "the", "he", "it", etc.".

One approach to evaluating a large collection is to sample a small portion from the entire corpus, transcribe it and evaluate. Evaluating a small subset can give us an estimation of the collection's accuracy, especially if the subset was sampled to contain diverse parts of the collection.

An example of the sample-based approach is described in [Tanner et al., 2009], where 1 % (around 20,000 pages) from the 19th Century British Library Newspaper Database and the Burney Collection were sampled to determine the quality of the OCR text. They transcribed and evaluated two article blocks from each page and searched for the highest OCR accuracy on each page. They chose this approach to determine the upper boundary of the OCR results for each newspaper title, which would indicate the upper usability of the OCR text.

The sample-based approach is especially convenient in cases where there is a strong indication that the collection has poor OCR and would need to be reOCRed. In that case, the GT created in the evaluation process can be used for training new models. Another benefit of this approach is that having a small subset allows us to manually or semi-automatically classify and analyze the data in the subset, which can give us OCR results on different groups of the data, and also help us understand the large collection better. For example, in our case, dividing test lines into Antiqua and Blackletter lines gave us a better understanding of how our OCR models perform on each. We also got a rough estimation of how much of the corpus is written in each font-family. The same approach can be used to examine languages used in the corpus or to analyze data from different periods.

On the other hand, the obvious drawback of the sample-based approach is the cost, as it requires humans, often specialists, to transcribe the data. The other problem is assuring that the sampled data represents the large collection well. This generates questions like: is it better to sample full pages, or to focus on smaller segments? How much data is necessary? Is it better to randomly sample data across the entire collection, or apply stratified sampling? To the best of our knowledge, there is no related work on best practices for sampling OCR data from large historical collections, so there exists no proven recipe for this approach.

The other approach, which is fully automatic, is to check how many words of

the OCR text can be found in a dictionary or a lexicon. A dictionary can be used for this purpose with morphologically simple languages, like English. For example, [Niklas, 2010] use a dictionary to evaluate the OCR quality of British historical newspapers. On the other hand, morphologically rich languages require the use of a lexicon.

[Kettunen and Pääkkönen, 2016] use modern-day lexicons to analyze the recognition rate in Finnish historical newspapers. Since they are aware that modern lexicons cannot recognize historical variants, they perform word frequency analysis and compare their results with the results from transcribed historical documents from the same period. In recent work, [Kettunen and Koistinen, 2019] use a historical variant of a Finnish lexicon to compare OCR results from two OCR engines on 10 years of material from a Finnish newspaper.

The approach of looking for the OCR words in dictionaries and lexicons has an obvious advantage that it is fully automatic and does not require GT. However, it is not very precise. For example, [Reynaert, 2016] gives an example from Dutch newspapers, where the word 'heeren' (eng: lords, gentlemen), due to highly frequent h-b confusion, often gets confused with the valid word 'beeren' (eng: bears or boars). To eliminate the effect of misspellings or erroneous words being recognized as correct, it is good practice to combine the look-up approach with frequency analysis and comparison with other historical documents to get a better estimate of the OCR quality.

The other highly common problem with this approach is that it is difficult to recognize words that are split with a hyphen into two rows, which happens very often in narrow newspaper columns. However, this problem can be approached by using a lexicon that has additional support for hyphenated words or by writing a script that would recognize split words and merge them for the lookup.

Similarly, some attention should be given to the cases where out of vocabulary (OOV) words are very short. In case of a poor OCR quality with many fragmented words, or when the OCR model inserts extra white-space in between the word segments, the evaluation might report a very high OOV count. This property might be a good indicator of pages or sections with poor OCR that would need to be corrected. However, when evaluating the whole collection, those segments might lower the overall accuracy results, leading to believe that the whole collection is of lower usability than it generally is.

Chapter 5

Training OCR Models

In this chapter, we survey the most popular tools used for training and recognition of historical data sets. We describe the CNN-LSTM based neural network used for training models in Calamari and present the network architecture we found the most suitable for our data. Furthermore, we show how selecting the training data and training parameters influences OCR results. Finally, we explore voting mechanisms and their effect on the results.

5.1 Tools

5.1.1 Ocropy

Ocropy (also known as OCROpus or OCRopus 1), is an OCR framework that consists of various Python command-line tools, made for various tasks in the OCR workflow. The toolkit contains tools for binarization and normalization of the images, page segmentation, creation of GT text, generation of synthetic training data, training of OCR models, recognition, evaluation and exporting results to the hOCR format.

Ocropy can also be used through the Ocrocis[1] interface, which has simplified commands and automatically manages the file and folder structure. Ocrocis comes with a detailed tutorial[2] on how to use both Ocropy and Ocrocis for recognition of historical books.

As previously mentioned, Ocropy was the first tool to implement training and recognition on a line level with the 1D bi-directional LSTM network, which both improved recognition accuracy and simplified the creation of the training data. Furthermore, due to good OCR results, enhanced with powerful pre-processing, this open-source toolkit became the go-to OCR framework for historical data.

[1] http://cistern.cis.lmu.de/ocrocis/

[2] http://cistern.cis.lmu.de/ocrocis/tutorial.pdf

However, Ocropy has some flaws. Firstly, it cannot be run on GPU, which makes training and recognition slow. It has support for multi-core training, but even when run on 8 cores in parallel, it is at least 4 times slower than GPU-enabled training tools [Wick et al., 2018a]. Secondly, the shallow neural network that Ocropy uses is limited and, as we showed in Publications **I** and **II**, handles larger or diverse data sets poorly. Furthermore, the training process is long. Ocropy does not provide an option for automatic early stopping that would break the training when the model's accuracy has not improved on a validation set for a certain number of times. In practice, that means that the training process consists of training a large number of steps, frequently saving the intermediate models. After the training process is done, all the models need to be tested on a validation set. The user can define the number of training steps and saving frequency, with the default setting of 1,000,000 steps and saving a model after every 1,000 steps. The training process, consisting of 1,000,000 steps, run on a CPU, can last for several days. On top of that, testing of 1,000 models can also take a long time with larger validation sets. In Publication **I** we used a validation set consisting of about 1,000 lines and testing 1,000 models on those took a couple of days. Thirdly, another problem with training is that the user needs to handle the character set that models learn to recognize. This would not be a big issue if the default settings for the training tool would automatically recognize the alphabet used from the characters found in the GT (which exists as an option). However, by default, the trainer uses a character set for German, which comes with the Ocropy package. In Publication **I** we missed being able to define our own character set, but luckily German and Finnish have a similar alphabet. However, we obtained better results with a customized character set in Publication **II**. All things considered, the training process with Ocropy requires expertise, manual action, and is often quite slow.

Most of the problems were addressed in the latest version published in 2018, called *OCRopus 3*[3]. The new version uses the PyTorch library developed by [Paszke et al., 2019], to provide deep neural networks and GPU support. [Wick et al., 2018a] evaluate Ocropy, OCRopus 3, Tesseract and Calamari tools on two data sets and they report that OCRopus 3 indeed performs much better than Ocropy. However, we never tried to use OCRopus 3 because they also report that Calamari outperforms all the other tools. Therefore, we decided to perform experiments with Calamari.

5.1.2 Calamari

Calamari is a TensorFlow [Abadi et al., 2015] based tool for training, recognition and evaluation of the OCR models. Encouraged by Breuel's experimental results on the combination of deep convolutional networks and LSTMs for character recognition published in [Breuel, 2017], [Wick et al., 2018a] developed the Calamari OCR

[3]https://github.com/NVlabs/ocropus3

tool with a CNN-LSTM hybrid approach, which acts as a replacement for Ocropy's training, recognition and evaluation tools.

In addition to the highly efficient deep CNN-LSTM neural network, Calamari solves all the problems that Ocropy has. First of all, training and recognition can be run on a GPU and in batches which, as reported in [Wick et al., 2018a] and [Wick et al., 2018b], makes both training and recognition multiple times faster than on a CPU. Secondly, the user can define a custom neural network architecture, as long as it consists of CNN and LSTM blocks. This opens the door to experiments with different sizes and numbers of layers, which enables the users to adapt the network to their needs. Furthermore, Calamari has implemented early stopping, a mechanism that stops the training when the model has not improved on a validation set in a provisional number of steps.

Calamari also provides extra features that can improve model accuracy. For example, its creators implemented a data augmentation option, where the training images are slightly altered automatically (for example, slightly rotated or skewed) in every training step so that the model never sees the same image twice. They report that data augmentation improves the accuracy for the models trained on very small training sets. However, it worsens the results when applied on larger training sets. Another useful feature is the voting mechanism. With Calamari, it is possible to perform the recognition with several models, where the voter uses models' confidences to decide which is most probably the correct recognition result. We describe voting in more detail in Section 5.4.

To evaluate Calamari's performance, [Wick et al., 2018a] compare how Ocropy, OCROpus 3, Tesseract 4 and Calamari perform on two historical data sets. They report that Calamari has the smallest error rates on both data sets. For example, on the first data set Calamari provides results with 0.155% CER (0.114% with voting), followed by the second-ranked Tesseract 4 (0.397%), OCRopus 3 (0.436 %) and Ocropy (0.870 %). Furthermore, [Wick et al., 2018b] perform a detailed comparison between Calamari and Ocropy with three different early printed books, using only running text lines (they excluded headings, marginalia and similar). Calamari undeniably outperforms Ocropy in all aspects.

5.1.3 Tesseract

Tesseract [Smith, 2007] is an OCR tool that has been on the market for almost 35 years, of which it has been open-source for the last 15 years. The widely popular version 3 includes layout analysis and in-built pre-processing, along with the collection of high accuracy pre-trained models for many languages. That makes it very popular for the recognition of modern texts. We find it particularly interesting because the research group at NLF has been using Tesseract 3 to train the models for re-OCRing the same corpus as we have. However, through their work and the work of other researchers, we can see some obvious downsides to Tesseract 3.

First, Tesseract 3 is a glyph-based model and the creation of the training data

on a glyph level is more time consuming than on a line level [Reul et al., 2019]. There are tools that generate artificial data sets once all glyphs from each font have been identified, but it is still difficult to collect glyph examples for various historical fonts. The NFL research group has focused on training models for Finnish Blackletter texts, but based on our experience that is the most uniform data set in the corpus font-wise. [Koistinen et al., 2017b] use an existing Antiqua model in addition to the Blackletter model, but do not report accuracy of the Antiqua model alone. The second problem with Tesseract 3 is that it uses the built-in Otsu's binarization algorithm, which performs worse than the one used in Ocropy. [Koistinen et al., 2017b] test additional image pre-processing methods as a pre-step for Tesseract and report improvement with those. However, that improvement comes with a significant performance cost.

The new Tesseract 4 version, which is line-based and supports deep network architectures, was released in 2017. The new version has better speed and accuracy performance ([Tesseract, 2017]), but according to [Reul et al., 2019] and [Wick et al., 2018a], it performs significantly worse than Calamari in recognizing historical texts. Neither Tesseract 3 or Tesseract 4 offer GPU support.

5.1.4 ABBYY FineReader

ABBYY FineReader is the leading commercial OCR software [Reul et al., 2018b]. It has pre-trained models for almost 200 languages, a very powerful segmentation tool and offers tools for semi-automatic post-correction [Reul et al., 2018c].

The newer versions offer support for historical fonts, but due to their high licensing price many public institutions opt for open-source solutions [Kettunen et al., 2018], [Reul et al., 2019]. Furthermore, comparative studies by [Reul et al., 2018c] and [Wick et al., 2018b] have shown that, in terms of accuracy, for historical documents it is better to train one's own models with open source software like Calamari and Tesseract than use ABBYY's pre-trained models.

5.1.5 Other Tools

Besides the most popular OCR tools described above, some other tools and frameworks have been used for the recognition of historical documents.

One such is Kraken[4] [Kiessling et al., 2017], which is a highly modified fork of Ocropy. Its creators altered Ocropy to address some of the Ocropy's limitations, but also to offer support for recognition of Arabic scripts. Therefore, Kraken has implemented support for right-to-left and bi-directional text and experimented with training models in multiple Arabic scripts. Furthermore, they have added a PyTorch back-end to support training deep CNN-LSTM networks [Reul et al., 2019]. In the recent work by [Kiessling, 2019], excellent results were obtained on historical documents (CAR over 98 %), but were not compared with other systems.

[4]https://github.com/mittagessen/kraken

Transkribus[5] is a Handwritten-Text Recognition (HTR) framework developed by the READ (Recognition and Enrichment of Archival Documents) project to help researchers in the humanities recognize handwritten historical documents. The idea behind the project is to provide humanists with an easy-to-use graphical interface where they can load and manage their data, manually segment lines of text and create GT. When they have the training data ready, they can initiate the training of a HTR model with a few mouse clicks. With the trained model, the user can easily recognize pages. Transkribus provides pre-trained HTR models for both handwritten and printed historical texts, which are then used as a starting point for fine-tuning with the training data provided by the user. The training and recognizing are done on READ partner servers in Innsbruck. While the user interface is open source, the HTR tools are not, and therefore we do not know what approach they use for training HTR models.

In Germany, there is the OCR-D[6] coordination project, with 8 project modules focused on various stages of OCR. Their focus is on mass digitization, and thus they prefer all of the OCR steps to be fully automatic. On the other hand, also in Germany, there is the OCR4all [Reul et al., 2019] framework, which collects open-source tools for different stages of the OCR process, but with a focus on recognizing smaller amounts of data (e.g. individual books) with very high accuracy. Therefore, they include both fully automatic and semi-automatic tools. For example, they find it beneficial to use a semi-automatic tool for region segmentation before the automatic line segmentation. Region segmentation in historical documents is still found to be very difficult so the user's input at this stage influences the final OCR results strongly. Also, they provide semi-automatic post-processing tools so that the user can correct any OCR mistakes. They use Calamari for OCR training and recognition.

5.2 Neural Networks

All of the modern OCR tools use neural networks to train OCR models. Using LSTMs (Long Short-Term Memory) for training OCR models became a standard in 2013 when [Breuel et al., 2013] found a way to use LSTMs to train OCR models on pairs of line images and corresponding ground truth. LSTMs [Hochreiter and Schmidhuber, 1997] are highly non-linear recurrent networks with multiplicative gates and additive feedback, which are widely popular for processing sequential data. Recently, [Breuel, 2017] proposed combining the LSTM architecture with deep Convolutional Neural Networks, as they have proved to be highly successful in the image processing field. [Wick et al., 2018a] demonstrate that the CNN-LSTM combination significantly outperforms the LSTM architecture, and we experienced the same effect in our experiments.

[5] https://transkribus.eu/Transkribus/
[6] http://www.ocr-d.de/eng

In this section, we briefly describe the default neural network architectures that Ocropy and Calamari use, and we write about the neural network architecture that we found optimal for our data sets. A detailed explanation of neural network terms and concepts can be found in [Goldberg, 2017].

Ocropy's default neural network consists of one hidden bi-directional LSTM layer with 100 nodes in each direction. It utilizes a learning rate of 0.0001 and a Momentum solver with a momentum of 0.9. They align the output and the ground-truth with the CTC (Connectionist temporal classification, [Graves et al., 2006]) algorithm, and use a heuristic decoder to map the frame-wise network output onto a sequence of symbols. Ocropy does not provide GPU support or batch training. [Breuel et al., 2013]

Calamari's default neural network is shown in Equation 5.1, and consists of two CNN layers with pooling and a ReLU-Activation function (**cnn**), followed by one bidirectional LSTM layer (**lstm**) and an output layer.

The first CNN layer has 40 filters, the second has 60 filters and both layers use one-pixel zero paddings and a 3×3 filter with a stride of 1. The pooling (*pool*) is performed with MaxPooling with a filter size of 2×2. The bi-directional LSTM layer has 200 nodes in each direction, with a dropout rate (*dropout*) of 0.5 applied during training. For training, Calamari uses an Adam solver with an initial learning rate of 0.001 and, like Ocropy, applies the CTC algorithm to align the OCR and GT data. Calamari has support for batches, can be run on GPUs, and its users can define a custom neural network architecture.

$$\begin{aligned} &\mathbf{cnn} = 40 : 3x3, pool = 2x2, \mathbf{cnn} = 60 : 3x3, pool = 2x2, \\ &\mathbf{lstm} = 200, dropout = 0.5 \end{aligned} \tag{5.1}$$

In Publication **III**, we find the optimal size of the neural network for our data set. In order to make sure that the neural network has enough memory to represent our diverse training set, we experiment with different numbers and sizes of layers to get the best 5-fold cross-validation results with the network shown in Equation 5.2.

$$\begin{aligned} &\mathbf{cnn} = 128 : 3x3, pool = 2x2, \mathbf{cnn} = 128 : 3x3, pool = 2x2, \\ &\mathbf{lstm} = 600, dropout = 0.5, \mathbf{lstm} = 600, dropout = 0.5 \end{aligned} \tag{5.2}$$

Similarly to the default Calamari network, we use two CNN layers. However, we got accuracy improvement with deeper CNN layers, so we increased them to 128, which is the maximal size possible with our GPU. We also tried to increase the number of CNN layers followed by MaxPooling, but that decreased the accuracy. We believe that the reason for this lies in the combination of short newspaper lines and too large a dimension reduction with pooling. [Wick et al., 2018a] show that the input image must be at least 2 px long for each character in the GT to have the minimum number of predictions without pooling (they leave a possibility of a

blank prediction in between any pair of adjacent characters). However, the pooling increases the minimal required image width. If one uses two 2×2 pooling layers on training example with 40 characters in the GT, the minimum image width should be $(2 \cdot 2) \cdot 40 \cdot 2\ px = 320\ px$. In the case of three pooling layers, the minimum image width increases to 640 px.

We also experimented with different combinations of the LSTM layers and obtained the best accuracy with a minimum of 1,200 LSTM nodes (in each direction). Due to the restrictions of our GPU, we could not train models with one LSTM layer with 1200 nodes, but we tried several different combinations (for example 3 layers with 400 nodes, or 2 layers with 600 nodes, followed by a dropout after each layer) and did not notice any significant difference in the accuracy. Furthermore, the accuracy did not change significantly when we further increased the LSTM size, so we decided to leave it at 1,200 nodes. We did not notice any improvement with different dropout rates, so we settled with the default 0.5.

5.3 Training Data and Results

The selection of training data is an important factor for training successful OCR models. In our work, we have experimented with different ways of acquiring training data, as well as with training models on different training set sizes and various data combinations.

Table 5.1: Summary of the OCR results published in the Publication **I**. We trained three models with Ocropy: one on our training data (fin-9k), one on the training data provided by the NFL (nfl) and one mixed model on all the data (fin-9k + nfl). We tested the models on our test set (Test-2k) and the NFL test set (Test-nfl).

test/model	fin-9k	nfl	fin-9k + nfl
Test-2k	**93.5**	89.0	93.0
Test-nfl	93.8	93.6	**94.8**

In Publication **I**, we trained models on Finnish data sets. At that point, we had two data sets, one smaller (*fin-9k*) with around 9,000 training lines that we had randomly sampled across the corpus, and which we later found consisted of 75 % Blackletter lines and 25 % Antiqua lines. The larger data set (*nlf*), provided by the NLF, consisted of more than 50,000 Finnish Blackletter lines. Using Ocropy, we trained OCR models on each of the data sets and tested them on two test sets, each similar to the corresponding training set (*Test-2k* and *Test-nfl*). We show the summary of the results from Publication **I** in Table 5.1. The model trained on the smaller set outperformed the model trained on the NLF set on both test sets. The model trained on our training data achieved CAR 93.5 % and 93.8 % and the larger one 89 % and 93.6 % on *Test-2k* and *Test-nfl* test sets, respectively. We also

trained models with both data sets together and the mixed-font model with all the data did worse on our test set (93.0 %), but achieved the best results on the NFL test set (94.8 %).

It is not surprising that the model that was trained on only Blackletter lines (*nlf*) did poorly on *Test-2k*, because that test set includes 25 % Antiqua lines. However, it is interesting that *fin-9k* outperforms the *nlf* model on the *Test-nfl* set, especially because both the *nlf* training and test data were taken from the same 255 pages. Now we could argue that Ocropy's shallow LSTM network learns better on the smaller training set, but training a model on both training sets together brings even more accuracy improvement (+1 %) on the *Test-nfl*. That leads us to the conclusion that it is good to have a more diverse training set and that using mixed-font models can improve overall results.

Table 5.2: Partial results from Publication **II**. The upper table shows results on the Swedish *Fine-grained test set* with monolingual and mixed models. *Swe-3k* is a model trained on 3,000 and *swe-6k* on 6,000 Swedish training lines. The first mixed model *fin-9k + swe-3k* was trained on 9,000 Finnish training lines and 3,000 Swedish lines, while *fin-9k + swe-6k* on 9,000 Finnish training lines and 6,000 Swedish lines. Similarly, the lower table shows results on the Finnish *Fine-grained test set* with monolingual and mixed models. Model *fin-7k* was trained on 7,000 and model *fin-9k* on 9,000 Finnish training lines. The mixed models are the same as in the upper table.

SWE	swe-3k	swe-6k	fin-9k + swe-3k	fin-9k + swe-6k
swe-all	**92.9 / 75**	92.8 / 75	90.6 / 69	90.1 / 67
swe-blackletter	92.8 / 74	**92.9 / 74**	90.8 / 69	90.3 / 66
swe-antiqua	**92.9 / 77**	92.8 / 76	90.4 / 69	90.0 / 67

FIN	fin-7k	fin-9k	fin-9k + swe-3k	fin-9k + swe-6k
fin-all	94.5 / 78	93.9 / 75	**95.0 / 79**	94.4 / 77
fin-blackletter	96.2 / 82	95.7 / 80	**96.3 / 82**	95.8 / 80
fin-antiqua	89.0 / 62	87.9 / 60	**90.7 / 67**	89.7 / 65

In Publication **II**, we continued training with Ocropy on the same *fin-9k* training set and we managed to obtain a small improvement after including a custom alphabet. We also tested different sizes of training sets and found that we obtained better results with less training data. The optimal Finnish model was trained on 7,000 training lines.

We also created about 6,000 Swedish training lines and trained and tested Swedish models. Additionally, we manually divided lines from both Finnish and Swedish test sets on Blackletter and Antiqua lines, to create what we call the *Fine-grained test sets*. We present the most important results from Publication **II** on

the *Fine-grained test sets* in Table 5.2. The results show that Swedish models generally perform a lot worse than Finnish models (CAR on Finnish test sets is in the range 94 - 95 %, while on Swedish test sets it is around 93 %). Finnish Antiqua seems to be the most difficult to recognize, having an accuracy well below Finnish Blackletter. Furthermore, results on the Finnish test set improved with the mixed model, while Swedish results were better with the monolingual model.

Experiments in Publication **II** showed that both Finnish and Swedish monolingual models trained with Ocropy performed better when trained on less data, and the mixed models performed better with less Swedish data (we do not show this in the table, but mixed models with *fin-7k* performed worse than with *fin-9k*). Our suspicion was that Ocropy does not have the capacity to capture a lot of diverse data, and since Swedish data is more difficult to recognize (we managed to achieve lower accuracy than for Finnish data) it is probably also more diverse.

Table 5.3: Unpublished preliminary Calamari results with models trained on the same data sets as in Table 5.2. Additionally, here we show results for the model *fin-9k+nlf* trained on *fin-9k* and *nlf* training sets together, as in Publication **I**. The results are expressed in CAR.

SWE	swe-3k	swe-6k	-	fin-9k+swe-3k	fin-9k+swe-6k
swe-test	93.1	**94.8**	-	91.5	94.1
swe-blackletter	92.9	**94.6**	-	91.2	93.8
swe-antiqua	93.2	**95.0**	-	91.8	94.4

FIN	fin-7k	fin-9k	fin-9k+nlf	fin-9k+swe-3k	fin-9k+swe-6k
fin-test	96.3	**96.4**	93.5	95.5	**96.4**
fin-blackletter	97.6	**97.8**	96.6	96.8	97.5
fin-antiqua	91.8	92.0	90.2	91.2	**92.7**

In Publication **III**, we ran experiments with Calamari, to test how the models behave when trained on deep neural networks.

In Table 5.3 we show the preliminary results with Calamari. We used the same training and test sets as in Table 5.2, with the addition of *fin-7k+nlf*, and we trained the models using Calamari's default neural network. In comparison with the Ocropy results, Swedish results improved +1.9 % and Finnish +1.4 %.

The mixed model trained on *fin-9k* and *nlf* training data also performed better than with Ocropy (+0.5 %). However, that model had the worst accuracy of all the models tested on the Finnish test sets.

The preliminary results clearly showed that Calamari's neural network can handle more data than Ocropy. Since Swedish results were still about 2 % CAR worse than Finnish, we thought that we might benefit from creating more Swedish training data. Similarly, we thought that we might improve Finnish Antiqua results with

more Finnish Antiqua lines as they were underrepresented in the Finnish training set.

Table 5.4: Test results with monolingual and mixed 5-fold models on the Swedish (upper table) and Finnish (lower table) *Fine grained test sets*. The models were trained with Calamari, with a custom neural network shown in Equation 5.2. Results show the mean CAR/WAR values.

SWE	swe-11k	fin-13k + swe-11k
swe-test	**96.6** / 87	**96.6 / 87**
swe-blackletter	96.4 / 85	**96.5 / 86**
swe-antiqua	**96.9** / 87	**96.9 / 87**

FIN	fin-13k	fin-13k + swe-11k
fin-test	95.9 / 89	**97.8 / 90**
fin-blackletter	96.5 / 91	**98.1 / 92**
fin-antiqua	94.2 / 83	**96.4 / 86**

Therefore, in Publication **III** we added 5,000 Swedish lines and 4,000 Finnish Antiqua lines. We also experimented with different neural network architectures to ascertain the optimal size of the neural network for our data. The final results on the *Fine-grained test set* are shown in Table 5.4.

With more training data and a deeper neural network, the Swedish monolingual model improved 1.8 % CAR. The accuracy of the Finnish monolingual model fell by 0.5 % despite a 2.2 % improvement on the Antiqua part. With the increased amount of the Antiqua training data, the model started to perform worse on Blackletter. However, the accuracy of the mixed model improved on all test sets. The mixed model outperformed both Swedish and Finnish monolingual models, as well as all the models from the preliminary testing.

Training one model that performs the best on all test sets is a significant achievement. The goal of our work is to train high-accuracy models to re-OCR the large newspaper collection and it is easier to approach that task with one model that is successful on all the data. Having one mixed model eliminates the need to perform language and font-type classification separately, which would slow down the process and increase the potential for mistakes in each additional step.

Despite the excellent results of the mixed model, we believe that there is still room to improve accuracy. Swedish is still 1.2 % worse than Finnish, and Finnish Antiqua is poor with an accuracy of 96.4 %. We believe that adding more training data could improve the results, but our analysis on the line level showed that, instead of randomly sampling data, at this point we would benefit more from acquiring only lines with the text printed in specialized fonts.

Additionally, it would be interesting to see what effect on accuracy a reduced

number of similar training lines would have. Good quality lines printed in common fonts are currently over-represented in the training set. Another option to add more importance to specialized fonts would be to fine-tune the mixed model by further training on a reduced training set with an equal number of lines printed in common and rare fonts.

5.4 Voting

Voting is an approach that combines outputs from multiple classifiers to get a prediction, which is a proven way to increase accuracy (over individual classifiers). Voting can be performed either on the results from different tools on the same data set or on the results achieved by training multiple models with one tool, with different training parameters or altered training data. If the classifiers produce confidences for all the available labels (characters), it is common to use an aggregation function (for example summation) to decide on the best candidate. Alternatively, majority voting can be used in cases when classifiers output just the top label. [Handley, 1998]

[Handley, 1998] present different voting approaches, as well as the alignment methods that need to be performed before the voting. A recent publication by [Reul et al., 2018a] presents a newer survey of different voting approaches used in different OCR scenarios.

[Reul et al., 2018b] present a confidence voting method on the results from a single tool (Ocropy). For voting, they use models from N-fold cross-validation, trained on a small number of lines for 7 historical books. They align results using ISRI Analytic Tools [Rice and Nartker, 1996], and for every position where the models disagree, they choose the label with the highest sum of confidences across the models. An example of confidence voting is shown in Table 5.5, where three models disagree on a label. Two models give the highest probability to the label "c" and one model to the label "e". However, the sum of probabilities is higher for "e" and that is the label chosen by confidence voting. They also compare their results with the ISRI's sequence voting and always obtain significantly better results (+5 %

Table 5.5: The table shows confidence measures that each model produces for two most probable outcomes "c" and "e". Confidence voting looks at the highest sum value for all voters, so it would predict label "e". In case of majority voting, the result would be label "c" because it is the top result in the most cases.

	c	e
model 1	0.2 %	0.7 %
model 2	0.4 %	0.3 %
model 3	0.5 %	0.4 %
$\sum$	1.1	**1.4**

Table 5.6: CAR on Swedish and Finnish results, without and with voting, with monolingual and mixed models.

	MONOLINGUAL		MIXED	
	single	voting	single	voting
swe-test	96.6	**97.2**	96.6	97.1
swe-blackletter	96.4	**97.1**	96.5	96.9
swe-antiqua	96.9	97.2	96.9	**97.3**
fin-test	95.9	97.8	97.8	**98.1**
fin-blackletter	96.5	98.3	98.1	**98.4**
fin-antiqua	94.2	96.2	96.4	**97.1**

- 10 % CAR) with the confidence voting. Calamari has implemented both voting mechanisms, with the confidence voting being the default.

In Publication **III**, we performed voting tests with Calamari models, using the default confidence voting. We voted on the results produced by monolingual and mixed models from 5-fold cross-validation and compared the result with the mean accuracy of five single models (also shown in the previous section). We present the results in Table 5.6.

On the Swedish test set, we achieve the best overall result when voting with monolingual models (+0.6 % CAR in comparison without voting), but voting with mixed models gives a very slightly worse result (-0.1 %). If we check the Blackletter and Antiqua results, we can see that Swedish Blackletter benefited the most from voting with monolingual models, while Antiqua achieved better results when voting with mixed models.

On the Finnish tests, we achieved the best overall results when voting with mixed models, which generated an improvement of 0.3 % over the mixed single models. With monolingual models, the results with single monolingual models were much lower than mixed models, so the voting improvement of +1.9 % resulted in the same accuracy rate as single mixed models.

The voting results reported in this table are the best OCR results reported on this corpora.

Chapter 6

Post-Processing

Initially we had an OCRed corpus of unsatisfactory quality. It was full of spelling errors and we had previously implemented spell-checkers with unweighted FSTs. After having enhanced our FST machinery for spelling correction with weights to implement a generalized sequence to sequence replacement mechanism, which in terms of machine learning falls under the category of unstructured classifiers, we realized that it would be difficult to make further progress without simultaneous access to the original image, so we needed to embark on a journey of image processing, and for this neural networks had already proven their case. It still remained to be seen, if post-correction had a place when everything had been seen and done.

Generally, OCR post-correction can be considered a spelling correction problem and many research groups have modified and applied spelling correction techniques to improve OCR results. Apart from OCR correction, such methods can also be used to change the historical variations into a standard language, for easier searching or analysis.

Newer approaches to OCR post-correction have taken a specialized approach by exploiting OCR properties. For example, they might use several OCR versions of the same text to obtain the correction candidates, or they can use OCR error patterns to either generate correction candidates directly or train error models.

Different OCR post-correction approaches use different methods, which to some extent overlap between the systems. However, all of them share two steps:

1. Generation of the correction candidates,

2. Decision making to accept or decline correction candidates.

In Figure 6.1, we illustrate the general approach to post-correction, with the two above-mentioned core steps in the center. In the first step, the correction candidates can be generated from the OCR text using pre-trained error models, sometimes also with help from external data. Once the correction candidates are created, the second phase of the post-correction process needs to rank the correction candidates and decide whether to accept or decline the highest-ranked one. The

decision-making step can be performed automatically with the direct use of the language models and lexicons or by models trained for this task. Alternatively, this step can be performed by human experts.

In this chapter, we describe different post-correction systems, with emphasis on sequence-to-sequence methods, which often serve as a basis for other, more advanced methods.

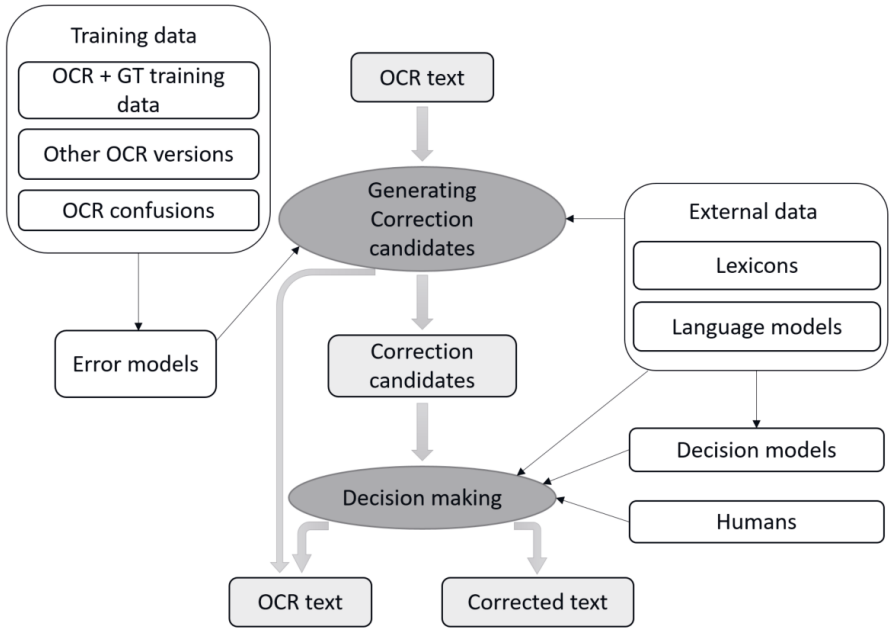


Figure 6.1: A typical OCR post-correction workflow. The OCR text is processed to generate correction candidates. This can be done with the help of the trained error model, lexicons or language models. If there are no correction candidates generated, the OCR text is kept as it is. Once the correction candidates are generated, there is a process of deciding whether to accept the most probable candidate or to keep the original OCR text. This decision can be done with the help of lexicons, language models, computer-generated models trained to make the decision or human input.

In Section 6.1, we present the traditional methods of spelling correction and normalization, and in Section 6.2. we present some more specialized methods. When embarking on our journey to enhance the quality of OCRed text, we chose to implement a general sequence to sequence model as an extension of the concept of spelling correction, which we now apply to the significantly better OCRed text and present in more detail in Section 6.3. The general wisdom, however, is that post-correction is still detached from the original so it will always work best when the output can be corrected with the guidance of a well-standardized norm incorporated

in a lexicon or a language model.

6.1 Spelling Correction and Normalization

Spelling correction can generally be considered a sequence-to-sequence translation problem, in which a sequence of characters with OCR errors is translated into a corrected sequence. The approach commonly includes an error model that contains translation rules, often weighted and context-based. Application of the error model usually includes a language model for output-level character dependencies and a lexicon for determining valid words of a language. Training of a historical language model can be difficult with scarce historical resources and historical lexicons are often unavailable or incomplete.

The earliest approaches to spelling correction used the edit distance as an error model in combination with a lexicon. [Brill and Moore, 2000] improved the approach by providing weights to the edit operations and [Silfverberg et al., 2016] took it to the next level by incorporating the context. [Eger et al., 2016] compare four popular spelling error correction methods, which they test on the tasks of normalization of Twitter data and correcting historical Latin OCR data. In the post-correction task, they use a Latin Language model and a lexicon and obtain the best results with the DirectTL+ system [Jiampojamarn et al., 2009]. Furthermore, they give a general short survey of character-level string-to-string translation models developed in Natural Language Processing (NLP) contexts.

[Silfverberg et al., 2016] present two spelling correction methods, one unstructured and one structured classifier, both implemented using weighted finite-state methods. They too use Twitter data normalization and post-correction of historical document tasks to test and evaluate their methods. The unstructured classifier, based on the work of [Lindén, 2006], creates a set of parallel context-sensitive replace rules, extracted from the training data, that are compiled into a weighted finite-state transducer. The structured classifier is implemented as a composite of unstructured and structured error models, in which the structured model is trained with an averaged perceptron tagger [Collins, 2002]. Both error models are then applied to the input strings and checked against a lexicon to filter out invalid words. Although they report better accuracy results with the structured approach, the unstructured classifier is easier to implement and faster to train.

Post-processing methods usually try to keep the text in the original form. However, when it is necessary to find all the historical variants of a word, it is useful to have a normalized version of the text. Normalization of historical texts is the process of translating historical variants into their standardized versions. It enables easier searching of the corpora, as all historical variants of a word are mapped to one lemma. Most of the spelling correction approaches can be modified for the normalization task, but the problem is that they often require either parallel corpora of the historical and modern variants or expert knowledge to create normalization

rules.

An interesting unsupervised approach (without ground truth training data) to the normalization of historical documents is presented in [Mitankin et al., 2014]. They use the machine learning methodology REBELS [Gerdjikov et al., 2013] to map each historical word not found in the modern dictionary to a ranked list of correction candidates. To select the most likely modern variant, they combine the ranked words with an n-gram model of the modern language. They report an 81.79 % normalization accuracy on the 17th century English historical texts, which is encouraging, but still quite low in comparison to the supervised results (93.96 %). This approach is somewhat similar to the more recent successful unsupervised post-correction methodologies of [Reynaert, 2016] and [Hämäläinen and Hengchen, 2019], so it might be worthwhile to re-visit the algorithm using newer methods and technologies.

Besides normalization, it is possible to create additional data on top of the original OCR text. In Impresso project ("Media monitoring of the past. Mining 200 years of historical newspapers")[1], they provide an interface for exploring and analyzing of historical newspapers, which were enriched with different automatically computed semantic annotations.

6.2 Other Post-Correction Methods

Other, more specialized post-correction methods, unlike the general sequence-to-sequence approach, take advantage of additional techniques and resources specific to the correction of the OCR data. In Publication **III**, we wrote a short survey of the most popular specialized post-correction methods, and in this section, we aim to present them slightly more systematically.

Some post-correction methods use external data sets to train error and language models. This approach is useful when a sufficiently similar external corpus exists, which unfortunately is not always the case with historical material. For example, [Kissos and Dershowitz, 2016] describe the process of correcting Arabic OCR data with poor OCR (70 % WAR on the document level) and report a 9 % WAR improvement. They are correcting modern printed texts, so they have a lexicon to select incorrect words, from which they then create correction candidates with a regression model trained on the OCR confusion matrix. The confusion matrix is created from OCR results of a transcribed set of images. They select the appropriate correction using a regression model based on word features, which they obtain from a language model built from a large, publicly-available text data set.

Similarly, but with historical data, [Généreux et al., 2014] perform post-correction of newspapers printed in German Fraktur between 1910 and 1920. They use a lexicon to find correction candidates and an external corpus together with edit distance and n-gram frequencies to score the candidates and find the best one.

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

Furthermore, [Evershed and Fitch, 2014] perform post-correction on Australian historical newspapers with low accuracy (below 80 % WAR). They also use an external representative text corpus to create a language model by extracting and using n-gram frequency lists. Their error model uses statistically weighted multiple character edits with the addition of a "reverse OCR" procedure. The "reverse OCR" is a procedure for obtaining the confusion cost by generating low-resolution glyph bitmaps for each word pair, with a generic font. The bitmaps are then overlaid in memory and adjusted to find the best alignment, from which they calculate a bit correlation measure. This measure helps obtain a slightly improved final confusion cost.

Manual post-correction of OCR text is perceived to achieve high-quality results, but it is time-consuming, expensive and hard work. Human correctors need to have a good knowledge of the text domain and need to be trained to use correction tools. Also, humans are bad at recognizing similar character confusions. To make manual correction easier and more effective, there is a state-of-the-art web-based tool `PoCoTo` [Vobl et al., 2014], specifically made for the post-correction of OCR results on historical texts. The tool was first developed as a desktop application in the EU project IMPACT[2], and later was adapted for web use. It uses an automated profiling mechanism that calculates which words in an OCR document are probably errors using a combination of statistical and lexical methods. The user interface shows page snippets together with OCR text containing possible errors and also offers possible corrections. Thus it makes it easier for users to correct errors manually.

Recently, a new, improved and fully automated version `A-PoToCo` [Englmeier et al., 2019] has been published. It uses a multi-input OCR approach together with a logistic regression model to rank post-correction candidates. To obtain multiple OCR inputs, different engines are used, or different models created by the same engine, to OCR the same historical document multiple times. To decide whether to accept the highest-ranked correction candidate, they train a logistic regression model, taking the confidence of the highest-ranked candidate and the difference to the confidence of the second-highest ranked candidates as features. They are planning to implement an interactive interface for humans to make the decisions on correction candidates in the next version of the tool, called `A-I-PoToCo`.

In another multi-input OCR approach by [Lund et al., 2013], five binarized versions of the same grayscale image are created using different binarization thresholds (a value between 0 and 255 that determines whether each pixel will be assigned to a white or a black pixel group depending on whether it is below or above the threshold value). Once they have all the candidates, they create a hypothesis lattice and select a single word using a supervised discriminative machine learning tool [Lund et al., 2011].

A completely different approach in post-correction methods is presented in

[2] http://www.impact-project.eu/

[Hämäläinen and Hengchen, 2019] and [Reynaert, 2010]. Both approaches use clustering methods to create correction candidates. [Hämäläinen and Hengchen, 2019] train a Word2Vec [Mikolov et al., 2013] model on the entire OCRed corpus to obtain clusters of the OCR errors together with the real synonyms. Using a dictionary, they check which of the words are correctly spelled, and using the Levenshtein edit distance they group the correct words with all misspelled variants. In [Reynaert, 2010] and further in [Reynaert, 2016], a post-correction Text-Induced Corpus Clean-up tool `TICCL` was created that also clusters all similar words in the corpus, but instead of being grouped semantically, they are grouped by character distances in Euclidean space, calculated and scored using an anagram hashing method. They use external data (lexicon, OCR confusions, and morphological rules) to distinguish between correct words and correction candidates and to adjust the scores. Finally, they select the correction candidate with the highest score.

6.3 Post-Correction with the Unstructured Classifier

In Publications **I** and **III**, we used the unstructured classifier (a classifier without an internal structure) from [Silfverberg et al., 2016] to post-correct our OCR results. The method is simple and easy to implement, and can be adapted to text normalization.

To train the error models, we recognized the OCR training and validation sets with the best OCR models and aligned the results with the corresponding GT on a character level. Due to the technical restrictions, we needed to reduce the size of the aligned training strings, so we split the lines with a white-space as a separator and used word-pairs to train error models. We left the punctuation, as it can also contain errors or be a valuable context. We trained the error models with different thresholds (minimum number of times a substitution must occur in a given context to be included in the ruleset) and picked the best error model after testing on the validation set.

To correct the test data, we composed the OCR strings with the error model to obtain all the possible weighted corrections, where weights correspond to the probability of each correction candidate. For Finnish data, we experimented with using a historical lexicon, which we expanded with leading and trailing punctuation. For all the correction candidates, we checked if they were valid words in the lexicon.

In Section 6.3.1 we present the Finite-state machine background and in Section 6.3.2 we present the results we obtained with this post-correction method.

6.3.1 Finite-State Machines as Unstructured Classifiers

Finite-state machines (FSMs) are practical and efficient at performing diverse pattern matching and string translation tasks and have been used in NLP for more

than 50 years. Finite-state automata (FSA) and transducers (FST) are often used to create language models, which can easily be used in NLP tasks, such as tokenization, shallow parsing, disambiguation, morphological parsing, spelling correction, etc. [Hulden, 2009]. A detailed presentation of FSMs, including formal definitions of the machines and operations, a survey of different FSM algorithms used for language modeling, reinforced with the implementation of the concepts and algorithms, can be found in [Hulden, 2009]. Furthermore, [Karttunen, 2000] addresses methods that use FSMs in NLP.

There are several popular commonly used FSM frameworks and libraries, like proprietary XFST [Beesley and Karttunen, 2003], and open-source *foma* [Hulden, 2009], OpenFST[3] [Riley et al., 2009], and HFST[4].

In our work, we used the Helsinki Finite-State Technology (HFST) framework and library, which is described in detail in [Lindén et al., 2011] and Publications **V** - **VII**. HFST provides open-source finite-state tools for creating morphologies and spellcheckers and, unlike XFST and *foma*, supports the usage of weights. Among other functionalities, HFST supports basic finite-state operations on transducers as well as more complicated concepts like cascade and parallel replace rules (created in Publications **V** - **VI**). We find these properties useful for the creation and application of the post-correction error models, both with and without the usage of a lexicon.

Figure 6.2: An example of an FST. The word *Suomi* (eng. Finland) is on the input side of the transducer and its partitive version *Suomea* on the output side. The ϵ symbol denotes an empty string.

In Figure 6.2, we show an example FST that for input string *Suomi* (eng. Finland) generates the partitive form *Suomea*. FSTs can easily be inverted so that input and output sides change places. The inverted FST from Figure 6.2 would generate for input string *Suomea* the nominative form, *Suomi*. This property is useful in the case of morphological analyzers and generators, where one FST can be inverted either way to serve the desired functionality. Furthermore, FSTs can easily be reduced to FSAs, where they do not create an output, but provide an answer if the input string exists in the language. Taking the input side of the above example would create an FSA that would accept the word *Suomi*. The same principle is applied to create a language acceptor from a morphological analyzer to check if an OCR word is a valid word.

[3]http://www.openfst.org
[4]http://hfst.github.io/

Replace Rules

Replace rules are sets of patterns that describe how the input text changes on the output side. We implemented the compilation of the cascade replace rules into FSTs using the `.r-glc.` operator from [Gerdemann and van Noord, 1999] and preference relations described in [Yli-Jyrä, 2008] in Publication **V**. In Publication **VI** we implement parallel replace rules and describe how the most common replace rules work and in Publication **VII** we test for differences between our results and those of XFST.

The complete set of rules with details and examples is explained in [Beesley and Karttunen, 2003]. In this subsection, we briefly present the basic replace rules in the post-correction context, using the notation from [Beesley and Karttunen, 2003].

A simple right arrow replace rule in Equation 6.1, with regular expressions A and B, denotes that A in the input language maps to B in the output language.

$$A \rightarrow B \tag{6.1}$$

Replace rules can have a context, so the right arrow replace rule in Equation 6.2, with A, B, C and D being regular expressions, denotes that A, when preceded by C and followed by D in the input language, maps to B in the output language.

$$A \rightarrow B \;||\; C \_\_ D \tag{6.2}$$

The contexts can be given in both input and output languages (with operators ||, \/, //, and \\), but combinations when one side of the context is in the input and the other is in the output language are conceptually difficult to understand and prone to errors, especially when combining multiple rules.

We can compile and apply replace rules in a cascade or in parallel. When processed in a cascade, one rule is applied after another so the order of rules is important, especially if preceding rules affect contexts of the following rules. When processed in parallel, all the rules are executed independently at the same time.

Once we compile the replace rules into a transducer, we can apply them to an FSA with the composition operation (`.o.`).

For example, if we compose a replace rule that changes letter `c` into `e` to an automaton containing the erroneously spelled word `Suomca`, as a result we get a correctly spelled word `Suomea`:

$$[\ Suomca\ ] \quad .o. \quad c \rightarrow e$$
$$Result : Suomea$$

Similarly, we can define that `c` changes into `e` only in front of an `a`. If we apply it to an FSA that contains two strings *Barcelona* and *Suomca*, the rule affects only

string *Suomca*, where c occurs just before *a*:

$$[\ Barcelona \mid Suomca\ ] \quad .o. \quad c \ \rightarrow \ e \ || \ \_\_ \ a$$
$$Result : Barcelona \mid Suomea$$

In our implementation, the post-processing error model is basically a list of parallel replace rules which are applied to the OCR strings in the same way, as in the above example. However, when training the post-processing error model, we want to include a statistical measure for each replacement. We can do that with weighted replace rules.

Weighted Replace Rules

Weighted FST and FSA (WFST and WFSA) [Linden et al., 2012], [Mohri and Sproat, 1996] behave in the same fashion as the non-weighted ones, with the difference that all the transitions and states can have a weight assigned. [Mohri and Sproat, 1996] explain how the weights are transferred and combined during finite-state operations.

In the replace rules, we can assign the weights either to the entire rule or just the replacement part. For example, we can assign that *c* changes to *e* with weight 0.5, when followed by *a*:

$$c \ \rightarrow \ e :: 0.5 \ || \ \_\_ \ a \tag{6.3}$$

The replacement weights are determined during training on the pairs of the OCR strings and the corresponding GT.

6.3.2 Results with the Unstructured Classifier

We used the unstructured classifier on the low accuracy OCR text in Publication **I**, and then again in Publication **III** on the OCR text of higher quality.

The main reason to use this particular method in Publication **I** was that it was easy and fast to implement. We chose not to focus on the more complicated post-correction methods, as we decided to concentrate on improving the OCR results first. As shown in Table 6.1, despite its simplicity, the method proved to be effective, consistently improving accuracy on all the test sets, with the improvement ranging between 0.1 % and 0.4 % CAR. We also calculate the relative improvement, which is an improvement on the remaining errors and it ranges between 0.9 % and 7.7 %. The reported results were achieved using a historical lexicon, which proved to work better than without the lexicon.

We used the same method in Publication **III**, to check how the method performs with the starting OCR accuracy about 4.5 % higher than before. In addition to the testing on the Finnish data, we also performed experiments on the Swedish OCR results, training post-correction models on respective monolingual training sets. We show a summary of the results in Table 6.2.

Table 6.1: Summary of the post-correction results from Publication **I**. The table shows the absolute post-correction CAR improvement over the OCR results shown in Table 5.1, and a relative improvement (Δ).

	fin-9k	Δ	nlf	Δ	mixed	Δ
Test-2k	+0.2 %	3.1 %	+0.1 %	0.9 %	+0.3 %	4.3 %
Test-nlf	+0.3 %	4.8 %	+0.4 %	6.3 %	+0.4 %	7.7 %

Table 6.2: Summary of the post-correction results from Publication **III**. The table shows post-correction accuracy improvement of the best OCR results achieved on this data set, shown in Table 5.6. We show the absolute CAR improvement, and a relative improvement (Δ).

	CAR	Δ
swe-test	+0.2 %	6.9 %
swe-blackletter	+0.2 %	6.5 %
swe-antiqua	+0.3 %	11.1 %
fin-test	+0.2 %	10.5 %
fin-blackletter	+0.2 %	12.5 %
fin-antiqua	0	0

This time the model did not perform any changes when we used the lexicon, so we report only the results without the lexicon. However, even without the lexicon, the results are consistently better or unchanged, with CAR improving 0.2 % - 0.3 %. The absolute improvement is the same as or smaller than in Publication **I**, but since the starting OCR accuracy is higher, the relative improvement is larger (6.5 % - 12.5 %).

The analysis of the post-correction results revealed that the Finnish post-correction performed only the correct changes, while the Swedish model performed a total of 19 changes with 2 wrong "corrections". While one of the mistakes was a real error, reducing the 100 % accurate line to a 97.8 % CAR, the other mistake was a change in a line that had very low accuracy (31.2 %) and the change did not make the line worse in a meaningful way. Furthermore, through the analysis, we found that the model performed only punctuation corrections.

Overall gains with this method are not huge, but they are consistent. The error models turned out to be conservative, which is positive since we want to preserve the true historical text. Without seeing the image, it is usually impossible to know if a certain word is a misspelling, a historical variant or an OCR error, so when correcting using the textual context alone, it is important to take a conservative approach in order to keep the text in the original form and to minimize post-correction induced mistakes.

Chapter 7

Conclusions and Future Work

7.1 Conclusions

In this work, we succeeded in training a mixed model that successfully recognizes different languages and fonts found in the corpus of historical newspapers and journals. Furthermore, we have shown that we can obtain even better results when voting with five similar mixed models. With this approach, we have reached a large and significant accuracy improvement over the current OCR results (+5-11 % CAR) and obtained the best recognition results on this corpus with a character accuracy rate around 97 % - 98 %.

Experimenting with post-correction, we have found that even a simple post-correction method can improve the OCR results moderately. We achieved an improvement of up to +0.4 % CAR on the OCR of lower accuracy (93 % - 94 % CAR), with a relative improvement up to 7.7 %. On the OCRed text of higher accuracy (97 % - 98 % CAR), we achieved the absolute improvement of up to +0.3 % CAR, with a relative improvement up to 12.5 %. The approach that we used was a conservative one: it performed very few corrections but most of them were correct.

7.2 Future Work

With our highly accurate mixed models, the first step would be to re-OCR the corpus of Finnish Historical Newspapers and Journals, which motivated this work in the first place. Due to the huge size of the corpus, this is a challenging task, but our preliminary tests have shown that running the process in many parallel batches on a GPU cluster should take a reasonable time. Once we have the new OCR results, we would need to add them to the existing XML files, to preserve the old results and retain compatibility with the current system, but also to preserve the data for potential research. Once we have the re-OCRed text, we could try to perform different post-correction methods. For example, we could try the unsupervised clustering methods, which report excellent results and should be easy to implement.

Although we are happy with the current OCR results, we believe that there is still room for improvement. As we stated earlier, we believe that adding more training data for specialized fonts could result in further improvement. Besides adding new training data, we could try to improve the quality of the existing data sets. Current line segmentation uses rectangular boxes to identify lines of text, which often results in parts of the text being cut off from the line image. Instead of using the current line segmentation, we could adapt the approach used in [Reul et al., 2019]. They suggest first segmenting blocks manually and then using a tight-fit line segmentation, which better preserves the text even in skewed images. Instead of manually segmenting blocks, we could use the existing ABBYY block segmentation.

Another approach to improving the image quality would be to try the Grayification method. As shown in Chapter 2, applying the Grayification method before Ocropy's binarization improves the quality of the binarized image. We would need to test how the addition of this step to the OCR workflow influences OCR accuracy and performance.

Binarization has been a standard procedure in the OCR workflow for a long time. However, with DNNs available, it would be interesting to see if it would be possible to train successful models on grey or even colored images. Would more information in the image require more training data and a larger neural network?

Finally, voting has shown to be a successful mechanism for improving OCR accuracy. We achieved the best results with 5 similar mixed models, but it would be interesting to see if a different voting algorithm would be more suited to voting successfully on more diverse models (for example, combinations of mixed and monolingual ones).

References

[Abadi et al., 2015] Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org.

[ABBYY, 2016] ABBYY (2016). How ABBYY FineReader was born or the story that took place more than 20 years ago. `https://finereaderblog.abbyy.com/how-abbyy-finereader-was-born-or-the-story-that-took-place-more\-than-20-years-ago/`. [Online; accessed 11-Nov-2019].

[Beesley and Karttunen, 2003] Beesley, K. R. and Karttunen, L. (2003). Finite-state morphology: Xerox tools and techniques. *CSLI, Stanford*.

[Bouillon et al., 2019] Bouillon, M., Ingold, R., and Liwicki, M. (2019). Grayification: A meaningful grayscale conversion to improve handwritten historical documents analysis. *Pattern Recognition Letters*, 121:46–51.

[Breuel, 2007] Breuel, T. M. (2007). The hOCR microformat for OCR workflow and results. In *Ninth International Conference on Document Analysis and Recognition (ICDAR 2007)*, volume 2, pages 1063–1067. IEEE.

[Breuel, 2008] Breuel, T. M. (2008). The OCRopus open source OCR system. In *Electronic Imaging 2008*, pages 68150F–68150F. International Society for Optics and Photonics.

[Breuel, 2017] Breuel, T. M. (2017). High performance text recognition using a hybrid convolutional-lstm implementation. In *2017 14th IAPR International Conference on Document Analysis and Recognition (ICDAR)*, volume 1, pages 11–16. IEEE.

[Breuel et al., 2013] Breuel, T. M., Ul-Hasan, A., Al-Azawi, M. A., and Shafait, F. (2013). High-performance OCR for printed English and Fraktur using LSTM

networks. In *2013 12th International Conference on Document Analysis and Recognition*, pages 683–687. IEEE.

[Brill and Moore, 2000] Brill, E. and Moore, R. C. (2000). An improved error model for noisy channel spelling correction. In *Proceedings of the 38th Annual Meeting on Association for Computational Linguistics*, pages 286–293. Association for Computational Linguistics.

[Collins, 2002] Collins, M. (2002). Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms. In *Proceedings of the ACL-02 conference on Empirical methods in natural language processing-Volume 10*, pages 1–8. Association for Computational Linguistics.

[d'Albe, 1914] d'Albe, E. F. (1914). On a type-reading optophone. *Proceedings of the Royal Society of London. Series A, Containing Papers of a Mathematical and Physical Character*, 90(619):373–375.

[Drobac et al., 2017] Drobac, S., Kauppinen, P., and Lindén, K. (2017). OCR and post-correction of historical Finnish texts. In *Proceedings of the 21st Nordic Conference on Computational Linguistics*, pages 70–76.

[Drobac et al., 2019] Drobac, S., Kauppinen, P., and Linden, K. (2019). Improving OCR of historical newspapers and journals published in finland. In *Proceedings of the 3nd International Conference on Digital Access to Textual Cultural Heritage*. ACM.

[Drobac and Lindén, 2019] Drobac, S. and Lindén, K. (2019). Optical font family recognition using a neural network. In *Proceedings of the Research Data And Humanities (RDHum) 2019 Conference: Data, Methods And Tools*, page 115, Finland. Studia Humaniora Ouluensia.

[Drobac and Lindén, 2020] Drobac, S. and Lindén, K. (2020). Optical character recognition with neural networks and post-correction with finite state methods. *International Journal on Document Analysis and Recognition*.

[Drobac et al., 2012] Drobac, S., Silfverberg, M., and Yli-Jyrä, A. (2012). Implementation of replace rules using preference operator. In *Proceedings of the FSMNLP 2012*, United States. ACL Anthology.

[Eger et al., 2016] Eger, S., vor der Brück, T., and Mehler, A. (2016). A comparison of four character-level string-to-string translation models for (OCR) spelling error correction. *The Prague Bulletin of Mathematical Linguistics*, 105:77–99.

[Englmeier et al., 2019] Englmeier, T., Fink, F., and Schulz, K. U. (2019). A-I-PoCoTo — combining automated and interactive OCR postcorrection. In *Proceedings of the Third International Conference on Digital Access to Textual Cultural Heritage*. ACM.

[Evershed and Fitch, 2014] Evershed, J. and Fitch, K. (2014). Correcting noisy OCR: Context beats confusion. In *Proceedings of the First International Conference on Digital Access to Textual Cultural Heritage*, pages 45–51. ACM.

[Généreux et al., 2014] Généreux, M., Stemle, E. W., Lyding, V., and Nicolas, L. (2014). Correcting OCR errors for German in Fraktur font. In *Proceedings of the First Italian Conference on Computational Linguistics (CLiC-It 2014)*.

[Gerdemann and van Noord, 1999] Gerdemann, D. and van Noord, G. (1999). Transducers from rewrite rules with backreferences. In *Proceedings of the ninth conference on European chapter of the Association for Computational Linguistics*, pages 126–133. Association for Computational Linguistics.

[Gerdjikov et al., 2013] Gerdjikov, S., Mihov, S., and Nenchev, V. (2013). Extraction of spelling variations from language structure for noisy text correction. In *2013 12th International Conference on Document Analysis and Recognition*, pages 324–328. IEEE.

[Goldberg, 1928] Goldberg, E. (1928). Statistical machine.

[Goldberg, 2017] Goldberg, Y. (2017). Neural network methods for natural language processing. *Synthesis Lectures on Human Language Technologies*, 10(1):1–309.

[Goodrich et al., 1979] Goodrich, G. L. et al. (1979). Kurzweil reading machine: A partial evaluation of its optical character recognition error rate. *Journal of Visual Impairment and Blindness*, 73(10):389–99.

[Graves et al., 2006] Graves, A., Fernández, S., Gomez, F., and Schmidhuber, J. (2006). Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks. In *Proceedings of the 23rd international conference on Machine learning*, pages 369–376. ACM.

[Hämäläinen and Hengchen, 2019] Hämäläinen, M. and Hengchen, S. (2019). From the paft to the fiiture: a fully automatic NMT and word embeddings method for OCR post-correction. In *Recent Advances in Natural Language Processing*, pages 432–437. INCOMA.

[Handley, 1998] Handley, J. C. (1998). Improving OCR accuracy through combination: A survey. In *SMC'98 Conference Proceedings. 1998 IEEE International Conference on Systems, Man, and Cybernetics (Cat. No. 98CH36218)*, volume 5, pages 4330–4333. IEEE.

[Hochreiter and Schmidhuber, 1997] Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9(8):1735–1780.

[Hulden, 2009] Hulden, M. (2009). Finite-state machine construction methods and algorithms for phonology and morphology.

[Jäntti, 1940] Jäntti, Y. A. (1940). *Kirjapainotaidon historia*. Söderström.

[Jauhiainen et al., 2016] Jauhiainen, T. S., Linden, B. K. J., Jauhiainen, H. A., et al. (2016). Heli, a word-based backoff method for language identification. In *Proceedings of the Third Workshop on NLP for Similar Languages, Varieties and Dialects VarDial3, Osaka, Japan, December 12 2016*.

[Jiampojamarn et al., 2009] Jiampojamarn, S., Bhargava, A., Dou, Q., Dwyer, K., and Kondrak, G. (2009). DirecTL: a language independent approach to transliteration. In *Proceedings of the 2009 Named Entities Workshop: Shared Task on Transliteration (NEWS 2009)*, pages 28–31.

[Kanan and Cottrell, 2012] Kanan, C. and Cottrell, G. W. (2012). Color-to-grayscale: does the method matter in image recognition? *PloS one*, 7(1):e29740.

[Karttunen, 2000] Karttunen, L. (2000). Applications of finite-state transducers in natural language processing. In *International Conference on Implementation and Application of Automata*, pages 34–46. Springer.

[Kauppinen, 2016] Kauppinen, P. (2016). OCR post-processing by parallel replace rules implemented as weighted finite-state transducers.

[Kettunen, 2015] Kettunen, K. (2015). Keep, change or delete? setting up a low resource OCR post-correction framework for a digitized old Finnish newspaper collection. In *Italian Research Conference on Digital Libraries*, pages 95–103. Springer.

[Kettunen et al., 2018] Kettunen, K., Kervinen, J., and Koistinen, M. (2018). Creating and using ground truth OCR sample data for Finnish historical newspapers and journals.

[Kettunen and Koistinen, 2019] Kettunen, K. and Koistinen, M. (2019). Open source Tesseract in re-OCR of Finnish Fraktur from 19th and early 20th century newspapers and journals - collected notes on quality improvement. In *DHN*, pages 270–282.

[Kettunen and Pääkkönen, 2016] Kettunen, K. and Pääkkönen, T. (2016). Measuring lexical quality of a historical Finnish newspaper collection — analysis of garbled OCR data with basic language technology tools and means. In *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC'16)*, pages 956–961.

[Kiessling, 2019] Kiessling, B. (2019). Kraken - an universal text recognizer for the humanities. *Proceedings of the DH*.

[Kiessling et al., 2017] Kiessling, B., Miller, M. T., Maxim, G., Savant, S. B., et al. (2017). Important new developments in arabographic optical character recognition (OCR). *Al-Uūr al-Wusā*, 25:1–13.

[Kissos and Dershowitz, 2016] Kissos, I. and Dershowitz, N. (2016). OCR error correction using character correction and feature-based word classification. In *2016 12th IAPR Workshop on Document Analysis Systems (DAS)*, pages 198–203. IEEE.

[Koistinen et al., 2017a] Koistinen, M., Kettunen, K., and Kervinen, J. (2017a). How to improve optical character recognition of historical Finnish newspapers using open source Tesseract OCR engine. *Proc. of LTC*, pages 279–283.

[Koistinen et al., 2017b] Koistinen, M., Kettunen, K., and Pääkkönen, T. (2017b). Improving optical character recognition of Finnish historical newspapers with a combination of Fraktur & Antiqua models and image preprocessing. In *Proceedings of the 21st Nordic Conference on Computational Linguistics*, pages 277–283.

[Lindén, 2006] Lindén, K. (2006). Multilingual modeling of cross-lingual spelling variants. *Information Retrieval*, 9(3):295–310.

[Linden et al., 2013] Linden, K., Axelson, E., Drobac, S., Hardwick, S., Kuokkala, J., Niemi, J., Pirinen, T., and Silfverberg, M. (2013). *HFST — a System for Creating NLP Tools*, pages 53–71. Communications in Computer and Information Science. Springer-Verlag, Germany.

[Lindén et al., 2011] Lindén, K., Axelson, E., Hardwick, S., Silfverberg, M., and Pirinen, T. (2011). HFST–framework for compiling and applying morphologies. pages 67–85.

[Linden et al., 2012] Linden, K., Silfverberg, M., Pirinen, T., Hardwick, S., Drobac, S., and Axelson, E. (2012). *HFST - an Environment for Creating Language Technology Applications*. Studies in Computational Intelligence. Springer-Verlag, Germany.

[Lund et al., 2013] Lund, W. B., Kennard, D. J., and Ringger, E. K. (2013). Combining multiple thresholding binarization values to improve OCR output. In *Document Recognition and Retrieval XX*, volume 8658, page 86580R. International Society for Optics and Photonics.

[Lund et al., 2011] Lund, W. B., Walker, D. D., and Ringger, E. K. (2011). Progressive alignment and discriminative error correction for multiple OCR engines. In *2011 International Conference on Document Analysis and Recognition*, pages 764–768. IEEE.

[Manning et al., 1999] Manning, C. D., Manning, C. D., and Schütze, H. (1999). *Foundations of statistical natural language processing*. MIT press.

[Mikolov et al., 2013] Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013). Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

[Mitankin et al., 2014] Mitankin, P., Gerdjikov, S., and Mihov, S. (2014). An approach to unsupervised historical text normalisation. In *Proceedings of the First International Conference on Digital Access to Textual Cultural Heritage*, pages 29–34. ACM.

[Mohri and Sproat, 1996] Mohri, M. and Sproat, R. (1996). An efficient compiler for weighted rewrite rules. In *Proceedings of the 34th annual meeting on Association for Computational Linguistics*, pages 231–238. Association for Computational Linguistics.

[Niklas, 2010] Niklas, K. (2010). Unsupervised post-correction of OCR errors. *Master's thesis. Leibniz Universität Hannover*.

[Otsu, 1979] Otsu, N. (1979). A threshold selection method from gray-level histograms. *IEEE transactions on systems, man, and cybernetics*, 9(1):62–66.

[Paszke et al., 2019] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al. (2019). PyTorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, pages 8024–8035.

[Reul et al., 2019] Reul, C., Christ, D., Hartelt, A., Balbach, N., Wehner, M., Springmann, U., Wick, C., Grundig, C., Büttner, A., and Puppe, F. (2019). OCR4all - an open-source tool providing a (semi-) automatic ocr workflow for historical printings. *arXiv preprint arXiv:1909.04032*.

[Reul et al., 2018a] Reul, C., Springmann, U., Wick, C., and Puppe, F. (2018a). Improving OCR accuracy on early printed books by combining pretraining, voting, and active learning. *arXiv preprint arXiv:1802.10038*.

[Reul et al., 2018b] Reul, C., Springmann, U., Wick, C., and Puppe, F. (2018b). 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)*, pages 423–428. IEEE.

[Reul et al., 2018c] Reul, C., Springmann, U., Wick, C., and Puppe, F. (2018c). State of the art optical character recognition of 19th century Fraktur scripts using open source engines. *arXiv preprint arXiv:1810.03436*.

[Reynaert, 2016] Reynaert, M. (2016). OCR post-correction evaluation of early Dutch books online - revisited.

[Reynaert, 2010] Reynaert, M. W. (2010). Character confusion versus focus word-based correction of spelling and OCR variants in corpora. *International Journal on Document Analysis and Recognition (IJDAR)*, 14(2):173–187.

[Rice and Nartker, 1996] Rice, S. V. and Nartker, T. A. (1996). The ISRI analytic tools for OCR evaluation. *UNLV/Information Science Research Institute, TR-96*, 2.

[Riley et al., 2009] Riley, M., Allauzen, C., and Jansche, M. (2009). OpenFST: An open-source, weighted finite-state transducer library and its applications to speech and language. In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics, Companion Volume: Tutorial Abstracts*, pages 9–10. Association for Computational Linguistics.

[Sauvola and Pietikäinen, 2000] Sauvola, J. and Pietikäinen, M. (2000). Adaptive document image binarization. *Pattern recognition*, 33(2):225–236.

[Shafait et al., 2008] Shafait, F., Keysers, D., and Breuel, T. M. (2008). Efficient implementation of local adaptive thresholding techniques using integral images. In *Document recognition and retrieval XV*, volume 6815, page 681510. International Society for Optics and Photonics.

[Silfverberg et al., 2016] Silfverberg, M., Kauppinen, P., and Lindén, K. (2016). Data-driven spelling correction using weighted finite-state methods. In *Proceedings of the SIGFSM Workshop on Statistical NLP and Weighted Automata*, pages 51–59, Berlin, Germany. Association for Computational Linguistics.

[Smith, 2007] Smith, R. (2007). An overview of the Tesseract OCR engine. In *Ninth International Conference on Document Analysis and Recognition (ICDAR 2007)*, volume 2, pages 629–633. IEEE.

[Tanner et al., 2009] Tanner, S., Muñoz, T., and Ros, P. H. (2009). Measuring mass text digitization quality and usefulness. *D-lib Magazine*, 15(7/8):1082–9873.

[Tauschek, 1935] Tauschek, G. (1935). Reading machine.

[Tesseract, 2017] Tesseract (2017). Tesseract: 4.0 accuracy and performance. `https://github.com/tesseract-ocr/tesseract/wiki/4.0-Accuracy-and-Performance`. [Online; accessed 29-Dec-2019].

[Vobl et al., 2014] Vobl, T., Gotscharek, A., Reffle, U., Ringlstetter, C., and Schulz, K. U. (2014). PoCoTo - an open source system for efficient interactive post-correction of OCRed historical texts. In *Proceedings of the First International Conference on Digital Access to Textual Cultural Heritage*, pages 57–61. ACM.

[Wick et al., 2018a] Wick, C., Reul, C., and Puppe, F. (2018a). Calamari - a high-performance Tensorflow-based deep learning package for optical character recognition. *arXiv preprint arXiv:1807.02004*.

[Wick et al., 2018b] Wick, C., Reul, C., and Puppe, F. (2018b). Comparison of OCR accuracy on early printed books using the open source engines calamari and OCRopus. *JLCL*, 33:79–96.

[Wikipedia, 2019a] Wikipedia (2019a). Distortion (optics) — Wikipedia, the free encyclopedia. `https://en.wikipedia.org/wiki/Distortion_(optics)`. [Online; accessed 5-Dec-2019].

[Wikipedia, 2019b] Wikipedia (2019b). Noise reduction — Wikipedia, the free encyclopedia. `https://en.wikipedia.org/wiki/Noise_reduction`. [Online; accessed 5-Dec-2019].

[Wikipedia, 2019c] Wikipedia (2019c). Tesseract (software) — Wikipedia, the free encyclopedia. `https://en.wikipedia.org/wiki/Tesseract_(software)`. [Online; accessed 11-Nov-2019].

[Yli-Jyrä, 2008] Yli-Jyrä, A. (2008). Transducers from parallel replace rules and modes with generalized lenient composition. In *6th international workshop, finite-state methods and natural language processing, FSMNLP-2007. Revised papers*, pages 197–212.