CHALMERS
UNIVERSITY OF TECHNOLOGY

Reading Numbers from Annual Reports

Using OCR and NLP to Extract Key Figures from Documents

Master's thesis in Engineering Mathematics and Computational Science

Sara Nordin Hällgren

DEPARTMENT OF MATHEMATICAL SCIENCES

CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2021
www.chalmers.se

Master's thesis 2021

Reading Numbers from Annual Reports

Using OCR and NLP to Extract Key Figures from Documents

SARA NORDIN HÄLLGREN

Department of Mathematical Sciences
Division of Applied Mathematics and Statistics
Chalmers University of Technology
Gothenburg, Sweden 2021

Reading Numbers from Annual Reports
Using OCR and NLP to Extract Key Figures from Documents
SARA NORDIN HÄLLGREN

Supervisor: Oscar Ivarsson, Department of Computer Science and Engineering
Examiner: Philip Gerlee, Department of Mathematical Sciences

Master's Thesis 2021
Department of Mathematical Sciences
Division of Applied Mathematics and Statistics
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2021

Reading Numbers from Annual Reports
Using OCR and NLP to Extract Key Figures from Documents
SARA NORDIN HÄLLGREN
Department of Mathematical Sciences
Chalmers University of Technology

Abstract

This thesis presents methods for extracting key figures from scanned annual reports. A two step approach is suggested, where a classifier locates the desired section and a separate algorithm then proceeds to identify and extract key figures within this context. Optical Character Recognition is carried out using Tesseract 4.1.1. The data consists of 280 annual reports submitted by Swedish companies, for which page labels as well as four different key figures are annotated. For the page classification task, a Random Forest classifier trained on TF-IDF embedded pages is found to achieve a test accuracy of 99.6%. To locate and extract a given key figure, it is found that an approximate string matching algorithm performs best, achieving an extraction accuracy of 92.9% on training documents and 89.6% on test documents. Accurate extraction is hampered by noise, so different image processing techniques are explored. The RCC filter is seen to improve extraction accuracy from 73.8% to 83.8% on a subset of difficult documents. Further improvements could be made by using an image processing technique based on deep learning.

Keywords: annual reports, extract information, reading from tables, optical character recognition, Tesseract, image processing, remove noise, binary images, scanned documents, page classification

Acknowledgements

I would like to thank everyone at AI-Hubben for presenting me with this interesting project, and for keeping me updated on the weather up in Sundsvall (I am writing this in May and I have just been informed there is still some snow left). Even though we have not been able to meet in person, I have enjoyed being part of the team! And thank you to everyone else at Bolagsverket for answering all of my questions - not least to the people at Helpdesk who have been very patient with my software installation requests.
Also, thanks to Oscar for being available to discuss my ideas, and to Ingrid for joining me for zoom coffee breaks - it has been nice to see some friendly faces while working from home. On that note, I want to thank Chris for helping me accommodate an office in our small apartment.

Sara Nordin Hällgren, Gothenburg, June 2021

Contents

List of Figures

List of Tables

1
Introduction

The purpose of this thesis is to examine whether machine learning approaches can be applied to extract key figures from scanned annual reports. A lot of progress has been made in extracting text from scanned documents, as long as the scan quality is good. Identified words and phrases can then be verified using a lexicon, to ensure that the extracted text follows spelling and grammar conventions. Extracting numbers from tables and other structured parts of documents remains more challenging. Furthermore, a number cannot be verified as easily; the best one can do is determine whether the extracted number is unlikely in the given context (for example, a negative number of company employees). The process of extracting text and numbers from an image, such as a photograph or a scanned document, is called Optical Character Recognition. OCR is a central concept in this thesis, and the reader can expect to see this acronym many times.
Developing an optical character recognition program which generalises well to multiple kinds of texts and documents is a very complex task. If sufficient training data is available, one could develop a custom solution for a very specialised task. However, the common option is to use an existing solution. This thesis makes use of the OCR engine Tesseract [26], which has been released for open source and is maintained by Google. Section 2.3 gives an overview of how Tesseract works. Using Tesseract, the aim is to extract numbers from the scanned annual reports; the focus is on key figures which are believed to have a connection to financial crime such as money laundering, tax evasion, and fraudulent activities. The thesis seeks to answer the following central questions:

1. Can the OCR results from Tesseract be improved, and if so, how? To this end, different image processing steps will be tested; the focus is on improved extraction of numbers specifically.
2. How can the desired key figures be extracted from the scanned text mass? Different machine learning models are considered, and a few models are implemented and evaluated.
3. How can the proposed models be continuously improved?

This thesis is intended to be a proof of concept and is thus focused on exploration rather than creating an end-to-end solution. All research has been made on a selection of real annual reports which have been filed by Swedish companies. In order to determine whether an extracted number is actually correct, the expected values needed to be manually annotated. Due to this time constraint, the work focuses on a selection of key figures found within the balance sheet; in the future, it is of course desirable to extract all numbers from the scanned documents. Annual re-

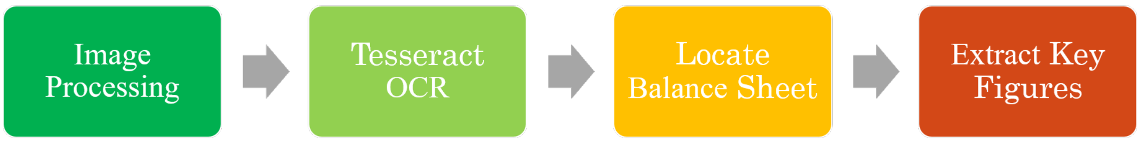


Figure 1.1: Overview of the workflow for this thesis. The first step is to apply image processing to the annual reports, after which the text is extracted using the OCR engine Tesseract. Next, the identified text on each page is used to locate the balance sheet. In the last step, key figures are extracted from the balance sheet.

ports are quite complicated and their layout and design can differ greatly between different companies. For this reason, the task of extracting key figures is broken up into several steps, visualised in Figure 1.1. Image processing can be applied to each scanned document in order to remove noise, after which the text is extracted using the OCR engine Tesseract. The words Tesseract is able to identify on each page are used to locate the balance sheet, which is then searched for the desired key figures. Ideally, for all documents a key figure will be found and extracted, and the extracted value should also be correct. The presence of noise in the scanned documents makes this unlikely, however the level of accuracy in the extracted numbers can be used to compare different image processing methods.

This introductory chapter outlines the background of the thesis, as well as previous work done within the field. It is followed by the Theory chapter, which covers the methods and algorithms that have been used in this process. The Method chapter then explains how and why these methods have been applied. Finally, the last two chapters present Results and Conclusions.

1.1 Background

This Master's thesis is performed in collaboration with Bolagsverket, the Swedish Companies Registration Office. Bolagsverket is a government agency whose primary role is to register companies, changes in companies, and company annual reports. A large amount of documents are submitted to Bolagsverket annually. The number of incoming annual reports is currently close to 600 000 a year, of which around 20% are sent digitally and the rest in paper form.

This volume of annual reports, together with difficulties in handling documents in paper form, complicates the discovery of financial crime. The current system is largely dependent on experienced administrators flagging a certain activity as suspicious, after which the case is handled manually. Bolagsverket is now looking to digitalise parts of this process, as it could be possible for a machine learning model to discovers patterns that are indicative of financial crime. Suspicious cases could then be flagged automatically, allowing administrators to focus on investigating the suspicious cases where their expertise is needed most. A reliable solution for detecting anomalous patterns can therefore lead to a more efficient use of resources within Bolagsverket, and a larger fraction of financial crimes being discovered in an early stage. However, this kind of automated solution is dependent on the numbers being available in a digital format, which motivates the need for this thesis project. It is worth noting that many government agencies use annual reports, either to detect or prevent crime or for other purposes. A robust solution for extracing key figures could therefore also be of use to e.g. Ekobrottsmyndigheten (the Swedish Economic Crime Authority) and Skatteverket (the Swedish Tax Agency). In fact, the Swedish government has introduced a digitalisation strategy with the ambition to be the best in the world at utilising the opportunities created by digitalisation. This initiative will certainly create a need for extracting numbers from many kinds of documents, not just annual reports. The four-step extraction approach introduced in this thesis is in no way specific to annual reports; the same model could instead be trained to extract data from other complex documents such as invoices or tax papers. The author hopes that this work will come of use not just to Bolagsverket, but to the entire Government as part of its digitalisation strategy.

1.2 Previous Work

Here follows a discussion of previous work that is relevant to this thesis project. Initially, two previous attempts at extracting data from annual reports are presented. In general the previous work is either in the form of a proprietary algorithm, where we cannot analyse how it works in detail, or it is focused on extracting data from digital annual reports. For this thesis, a custom approach needs to be developed from scratch. The main difficulty lies in extracting good quality information from structured elements, such as tables, within scanned documents. Previous work in this field relates to preprocessing images for Optical Character Recognition, as well as Detecting and Extracting Tables.

1.2.1 Extracting Data from Annual Reports

Many organisations have an interest in analysing annual reports, so data extraction tools have been developed before. Large Financial Services companies have the resources to develop their own data extraction algorithms, but do not wish to share how their algorithms work. Some companies, such as Infrrd [7], provide data extraction as a service. They claim that extracting full value from the information in annual reports is still a challenge, due to the complexity of the documents. Annual reports come in a variety of different formats and style templates, and important information is presented in tables, graphs, and other containers OCR has difficulty reading. The kinds of information in an annual report are consistent, but how they are presented are not: the format might differ greatly from year to year for one single company. Infrrd offers to solve this data extraction problem for Financial Services companies, for a fee.
Another piece of related work was carried out by Hering [13] in 2017. The author studies U.S. annual reports, which have to be filed with the Securities and Exchange Comission. An algorithm is developed to extract textual information embedded in

these annual reports. The dynamic extraction algorithm is able to identify structural changes within financial statements, and is applied to over 180000 annual reports for descriptive analysis. Annual reports in the U.S. have a much more uniform format than in Europe: official financial statements must be formatted in ASCII or HTML, allowing data to be extracted using regular expressions. The approach taken in this previous study could be applied to the annual reports which are submitted to Bolagsverket digitally, but it does not address the difficulties of extracting data from scanned documents.

1.2.2 Preprocessing Images for OCR

Tesseract [26] is an open-source tool for optical character recognition, which has been requested by Bolagsverket for this thesis project. This section presents previous work focused on improving OCR results, either in general or specifically for Tesseract. It is worth noting that the developers of Tesseract have published their own recommendations for improving the quality of results, which are covered in more detail later in this report.

Sporici et al. [27] found that the accuracy of Tesseract 4.0 can be improved by applying convolution based preprocessing. The authors point out that Tesseract displays great performance on favourable inputs, but is known to be sensitive to certain noise patterns. Most approaches work according to predetermined conditions and apply the same preprocessing to all parts of the image. This kind of approach relies on being interpretable to the human eye, but Sporici et al. propose that it could be more beneficial to preprocess with respect to individual features and shapes. With this in mind, a preprocessing is developed to optimise the performance of Tesseract 4.0, with no aestethic validation. They propose an adaptive preprocessing guided by a reinforcement learning model, in order to minimise the edit distance between the recognised text and the ground truth. The challenge can be formalised as an optimisation problem to discover the best convolution kernels which maximise Tesseract's accuracy over a set of training samples. The task resembles the multi-armed bandit problem with a continuous search space. A five layer convolutional network proves to boost the character level accuracy significantly. It should be noted that the data set used in this study contains blurry images with low contrast; The authors find Tesseract to prefer preprocessed images which have good contrast, even at the expense of partially damaged characters.

A Masters thesis by Björkman [3] investigates the effects of different preprocessing methods on OCR results. The study is focused on detecting counterfeit mechanical components by reading a bar code and validating it, and the data consisted of 100 images of boxes with labels. Different preprocessing methods are used: binarisation using different kinds of grayscale conversion, noise reduction using Gaussian filtering, and image sharpening and cropping. The author defined different levels of errors, and applied the Wilcoxon signed-rank test to check if there is a statistically significant improvement. One of the OCR engines used for the project was Tesseract, and in this setting it was found that cropping, sharpening and Gaussian filtering positively impact the performance. However, these preprocessing steps were only tried individually and not in combination.

1.2.3 Detecting and Extracting Tables

Extensive previous work has been done in the area of detecting and extracting tables from scanned documents. Akmal Jahan and Roshan [19] propose an algorithm for detecting tables, using local thresholds for word space and line height. The authors note that OCR (Optical Character Recognition) methods work well for digitising texts, but face difficulties when the text contains graphics elements such as tables. They also highlight that tables come in three main types: they either have bounding lines, parallel lines, or no lines. Previous algorithms have focused on one of these three types; as a result they are less versatile and can for example struggle with full page tables. Akmal Jahan et al based their algorithm on a few simple assumptions, such that a text line has more characters per line (and thus a higher word space count) than a line within a table. Their data set consisted of around 300 images in different formats such as tiff, jpg, and png. In terms of image preprocessing, they applied binarisation using adaptive threshold, noisy border removal, and dilation. The authors achieve an average table detection accuracy of around 75%, across the three kinds of tables. Based on the data collection method, the algorithm only sees images that actually contain a table. To achieve similar results, pages of interest might have to be chosen using another algorithm.
Babatunde et al. [2] suggest a Hidden Markov Model approach for extracting data from tables within heterogeneous documents. The authors generated a data set consisting of Word, PDF, and HTML files which all contained multiple tables. The Word and PDF documents were converted to HTML format, then an algorithm identifies tables which are subsequently extracted using the Hidden Markov Model. The authors point out three main ways in which the extracted tables can differ from the original: two cells can be merged, a cell can be missing, or an entire row can be missing from the result. The HMM approach yields a per token f1 score of 89% on a test data set. A limitation of this approach is that the original documents need to be converted to HTML; even if this conversion is easily carried out it could lead to a loss of information.
Recently the data extraction field has also seen the emergence of software tools which promise to take care of the entire problem. One such online service is Nanonets [15], which provide a dedicated table extraction tool. This is a black-box approach which might not be applicable to different document types or languages. When handling classified information, a cloud solution is usually not advisable. Another black-box option which can be run locally is Tabula [1], which is made for reading data from PDF tables. It should be noted that the tool only works on text-based PDFs, not scanned documents. Furthermore, a human needs to manually select each table and then preview and export the result. This approach is not very scalable, and it is also uncertain how the model will react to inputs in different languages. If one is restricted from using cloud services, and the documents to be scanned are in an uncommon file format, a custom approach is needed for extracting the data.

2
Theory

This chapter provides the theoretical background for the thesis project. In Section 2.1 there is a brief introduction to annual reports, for readers who are unfamiliar with the terminology. The next section, Section 2.2, covers image processing methods that have been applied to the scanned annual reports in order to remove noise or otherwise make them easier to process. Optical Character Recognition is then used to extract the text from each document; an overview of the OCR engine is provided in Section 2.3.

The OCR engine, Tesseract, outputs the text it has identified on each page. In order to classify different pages within a specific document, or to measure the similarity between two pieces of text, it is beneficial to first convert the text to a numerical representation. These representations can either be based on word counts or learned from existing text; Section 2.4 introduces different ways that text can be represented numerically. Using these vector representations of the text on each page, the pages can then be classified in order to locate the balance sheet. The classification methods used in this thesis are detailed in Section 2.5. Certain metrics are introduced in the interests of quantifying the performance of different classifiers; they are described in Section 2.6. This section also covers similarity measures which have been used to quantify the level of similarity between two pieces of text.

2.1 Annual Reports

This section provides a brief overview of annual reports, including their contents and structure. The terminology introduced here will be central in the rest of the report, but no prior knowledge of accounting is required. The information in this section has been retrieved from Bolagsverket's website [4]. Every limited company (aktiebolag) in Sweden must prepare and file an annual report with Bolagsverket every year. Depending on the size of the company and on the type of company, the annual report must contain different parts such as an auditor's report and a cash flow statement. The annual report of all limited companies must include a director's report, an income statement, a balance sheet, and notes that provide any additional information regarding the other entries.

Annual reports must include a financial statements approval which certifies that the balance sheet and income statement were approved at the shareholder's meeting. If the limited company has an auditor, they must prepare an auditor's report to be submitted to Bolagsverket along with the annual report. The auditor's report should contain the auditor's recommendations; for example whether they can or

cannot recommend that the board of directors be discharged from liability.
Three important financial statements are the income statement, balance sheet, and cash flow statement. An income statement presents a summary of the company's total income and expenses during its financial year. The layout can either be by function or by expenses, where the company can choose the layout they find most suitable for describing their results. Once a layout has been chosen, it should be kept for subsequent income statements. The balance sheet is a summary of the company's assets, equity, provisions, allocations, and liabilities on the last day of its financial year. As the name indicates, this statement should *balance*; the balance sheet total (the total assets) should add up to the same amount as the total liabilities. Furthermore, the year's results from the income statement is also presented in the balance sheet. A cash flow statement shows a company's total amount of funding and capital investment during its financial year. Only large companies are required to submit a cash flow statement. There is no standardised template for annual reports in Sweden, however Bolagsverket have observed around 10 common styles in the incoming documents. Bolagsverket estimates that 90% of annual reports are 8 pages or shorter, but for large holding companies these documents can reach several hundred pages. This large variability in the format and size needs to be accounted for when automatically handling the annual reports.

2.2 Image Processing

This section is based on chapter 3 of the book *Computer Vision: Algorithms and Applications* [29]. Most computer vision applications will only yield good results if images are first properly processed. Such preprocessing operations include binarisation, reducing noise, and increasing sharpness. The most simple kind of image transform manipulates each pixel independently of its neighbours; examples of such operators include brightness and contrast adjustments. Neighbourhood operators are a more sophisticated class of preprocessing metods; here, each new pixel value depends on inputs from the surrounding area. In this thesis, neighbourhood operators are used for two main purposes: removing noise, and changing the shape of objects in binary images using *morphological transformations*. These two concepts are described below.

2.2.1 Removing Noise

It is desirable to remove any noise from the scanned documents, in order to improve the performance of the optical character recognition algorithm. A neighbourhood operator uses a collection of pixel values in the vicinity of a given pixel to determine its final output value. In this thesis, these local operators are used to filter images in order to remove noise. The most common local operator is a *linear filter*, which uses a convolution approach. The output pixel's value is a weighted sum of input pixel values within a small neighbourhood $\mathcal{N}$:

$$g(i,j) = \sum_{k,l} f(i+k, j+l)h(k,l) \tag{2.1}$$

Here, $f(x, y)$ represents the value at position (x, y) in the original image and $g(x, y)$ is the value at position (x, y) in the filtered image. The entries in the weight kernel $h(k, l)$ are usually called filter coefficients. A side effect of the convolution is that the image size is decreased, which might not be desirable in some applications; different border padding schemes can be applied to address this issue. However, since the images used in this thesis are expected to be completely white around the edges, padding is not considered here.

2.2.1.1 Moving Average Filter

The simplest filter is the moving average filter, which simply averages the pixel values in a window of size k. The corresponding weight kernel of size 3×3 is:

$$K = \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \tag{2.2}$$

An extension is the α-trimmed average, where a fixed percentage of the lowest and highest pixel values are rejected when computing the mean. This modified filter is more robust to outliers, but is not beneficial when filtering binary images (since each pixel value is either 0 or 1).

2.2.1.2 Gaussian Smoothing

A Gaussian filter uses a high weight for the target pixel, and decreasing weights for pixels further away. The filter coefficients are taken to be discrete approximations of a two dimensional Gaussian surface. Kernel width and height both need to be positive and odd, and the standard deviation in the x and y directions can be tuned independently. An example 3×3 Gaussian weight kernel has the form:

$$K = \frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} \tag{2.3}$$

2.2.1.3 Median Filtering

The filters discussed previously are all linear, corresponding to convolution with a fixed kernel. However, in some settings better performance can be obtained by using non-linear combinations of neighbouring pixels. Some images might display so called *shot noise*, occasionally displaying large values. In this case, a Gaussian filter will fail to remove the noise; a better filter to use in this setting is the *median* filter. The median filter takes the median of all pixels under the kernel and the central element is replaced with this median value. Since the shot noise value usually lies well outside the true values in the neighbourhood, the median filter is able to filter out such bad pixels. This non-linear smoothing method has a desirable *edge preserving* property, meaning that while filtering out high-frequency noise, is has a lesser tendency to soften edges. Another possibility is to compute the weighted median, where each pixel is used a number of times depending on its distance to the center.

2.2.1.4 Bilateral Filtering

A desirable filter combines the concept of a weighted filter kernel with a better version of outlier rejection. Instead of rejecting a fixed percentage α of pixels, the filter would reject pixels whose *values* differ too much from the central pixel value. This is the essential idea in *bilateral filtering*, where the output pixel value depends on a weighted combination of neighbouring pixel values:

$$\mathbf{g}(i,j) = \frac{\sum_{k,l} \mathbf{f}(k,l) w(i,j,k,l)}{\sum_{k,l} w(i,j,k,l)} \tag{2.4}$$

Where the weighting coefficient $w(i,j,k,l)$ is computed as a product of a *domain kernel* $d(i,j,k,l)$ and a data-dependent *range kernel* $r(i,j,k,l)$. These are defined as follows:

$$d(i,j,k,l) = \exp\Big(- \frac{(i-k)^2 + (j-l)^2}{2\sigma_d^2} \Big) \tag{2.5}$$

$$r(i,j,k,l) = \exp\Big(- \frac{||\mathbf{f}(i,j) - \mathbf{f}(k,l)||^2}{2\sigma_r^2} \Big) \tag{2.6}$$

Their product is the data-dependent *bilateral weight function* $w(i,j,k,l)$:

$$w(i,j,k,l) = \exp\Big(- \frac{(i-k)^2 + (j-l)^2}{2\sigma_d^2} - \frac{||\mathbf{f}(i,j) - \mathbf{f}(k,l)||^2}{2\sigma_r^2} \Big) \tag{2.7}$$

The range kernel measures the intensity similarity to the center pixel. For colour images, the range kernel uses the *vector distance* between the center and neighbouring pixels. This means that a sharp edge in any of the three colour bands signals a change in materials, and thus the pixel's influence is down-weighted.

2.2.2 Morphological Transformations

Non-linear filters are often used to enhance grayscale and colour images, but are also extensively used to process binary images. The most common binary image operations are called *morphological operations*, since they change the shape of the underlying binary objects. To perform such an operation, the binary image is first convolved with a binary kernel. The binary output value is then selected depending on the thresholded result of the convolution. Kernels can have any shape, for example a regular 3×3 box filter or a more complicated disc. When convolving a binary image f with the kernel s, we let

$$c = f \otimes s \tag{2.8}$$

denote the integer-valued *count* of the number of 1s inside each kernel as it sweeps across the image. The the number of pixels the kernel contains is denoted by S. Standard operations used in binary morphology include:

- **Dilation**: $\mathrm{dilate}(f,s) = \theta(c,1)$
- **Erosion**: $\mathrm{erode}(f,s) = \theta(c,S)$
- **Opening**: $\mathrm{open}(f,s) = \mathrm{dilate}(\mathrm{erode}(f,s),s)$

- **Closing**: $\text{close}(f, s) = \text{erode}(\text{dilate}(f, s), s)$

Dilation thickens black segments of an image, much like a pupil dilates. On the other hand, erosion makes objects thinner and removes small islands. The opening and closing operations tend to leave large regions and smooth boundaries unaffected, while removing small objects and smoothing boundaries. The number of pixels added or removed at a boundary is dependent on the size and shape of the kernel s.

2.3 Optical Character Recognition

Optical Character Recognition (OCR) allows a computer to identify text in images without human intervention. International conferences and competitions such as the International Conference on Frontiers in Handwriting Recognition have advanced the research effort within OCR. For this thesis project, optical character recognition is performed using an engine called Tesseract which is described in detail in the following section.

2.3.1 Tesseract

Tesseract is an open-source OCR engine which was initially developed at Hewlett-Packard. It began as a PhD research project, and gained attention for its impressive results during the 1995 UNLV Annual Test of OCR Accuracy. Tessseract was considered as a potential add-on for HP scanners but internal development of the software was stopped in the mid 90's. In 2005 HP released Tesseract for open source, and has been maintained by Google since then. This information regarding Tesseract's history and architecture are retrieved from an article titled ”An overview of the Tesseract OCR engine”, published by the lead developer at Google [26].

2.3.1.1 Architecture

HP had developed proprietary tools for page layout analysis which were used in products, and therefore not released for open source. Tesseract was developed to use these tools and thus never needed its own page layout analysis; it assumes that the input is a binary image with optional defined text regions. Processing is performed following a traditional pipeline, where the first analysis step stores the outline of all image components (which probably made Tesseract the first OCR engine to excel at handling white-on-black text). The outlines are then grouped into *Blobs*.
Blobs are organised into lines of text, which are then analysed to determine if the font is monospaced or proportional. Monospaced text can be easily split into individual words, while proportional text is split using definite spaces and fuzzy spaces. The recognition then proceeds as a two-pass process. In the first pass, the algorithm attempts to recognise each word individually. Each satisfactory word is passed as training data to an adaptive classifier, which then gets a chance to more accurately recognise text lower down on the page. To apply any useful information learned by the adaptive classifier also to the top of the page, a second pass is run over the page. This time, words that were not satisfactorily recognised the first time are recognized again.

2.3.1.2 Finding Lines and Words

Tesseract's line finding algorithm is robust to skewed text, which decreases the need for deskewing pages in the preprocessing stage. Blobs that are smaller than the median height are temporarily filtered out, as they are most likely punctuation or noise. The remaining blobs are more likely to fit a model of parallel, non-overlapping, but potentially sloped lines. Each blob is then assigned to a unique text line, while tracking its slope across the page, greatly reducing the danger of assigning to an incorrect text line in the presence of skew. Once the filtered blobs have been assigned to lines, the baselines are estimated using a least median of squares fit. The filtered-out blobs are then fitted back into the appropriate lines. Finally, partially overlapping blobs are merged.
Once the text lines are identified, a quadratic spline is used to fit the baselines more precisely by a least squares fit. Tesseract then tests the identified text lines to determine if the font is monospaced or not. For a monospaced font all the characters have the same width, and it is thus quite straightforward to split the words into individual characters. Proportional text is far from trivial to split; Tesseract solves this by measuring the gaps and making a final decision after word recognition. The word recognition process includes identifying the blob with the lowest confidence and chopping it in the most promising way. After exhausting potential chops, the word is given to the Associator which tries to build the chopped blobs into candidate characters. Already in 1989, Tesseract displayed better accuracy on broken characters than the competing OCR engines. That early success was thanks to a character classifier which excels at recognising broken characters.

2.3.1.3 Character Classification

An early version of Tesseract used topological features to classify different characters. This approach is theoretically independent of font and text size, but is not robust to real-life issues such as broken characters. The revolutionary idea was that the features in the unknown need not be the same as the features in the training data. The classifier now uses segments of a polygonal approximation as features during the training phase; in recognition, it extracts small features of fixed length from the outline and matches these many-to-one against the training features. This process proved to easily handle recognition of damaged characters, at the expense of a high computational cost. To mitigate this issue, character classification is performed as a two-step process. Initially, the features obtained from the outline of the unknown characters are used to determine a shortlist of known character classes which it might match. After this, the actual similarity is computed and the most likely character is returned.

2.4 Text Representations

This section is based on chapter 6 of the book *Speech and Language Processing* [16]. Words that appear in similar contexts tend to have similar meanings. The book mentions the distributional hypothesis for words, which links their similarity

in distribution to their similarity in meaning. This linguistic hypothesis can be instantiated using vector semantics, where the meaning of a word is represented as a vector, or *embedding*, which can be learned directly from how it is distributed within a text. All natural language processing applications that make use of words meanings are based on these vector representations. In a static embedding each word is mapped to a fixed vector; these are the foundations for more powerful contextualised embeddings such as **BERT**.
Depending on the application, different kinds of word representations will be more or less useful. A good model of word meanings should be able to tell us that certain words are similar in meaning, such as *apple* and *banana*, and that others are antonyms (*hot* is the opposite of *cold*). It should also represent that words can be related without being close in similarity; *coffee* and *cup* are not similar in meaning, but they are nevertheless likely to appear together. A model might even have an inherent understanding of context, allowing it to determine when the word ”Apple” refers to the fruit and when to the software company.
An important baseline model is **TF-IDF**, where the meaning of a word is expressed as a function of the counts of nearby words. Conversely, individual documents can be represented using the counts of words that they contain. This method results in very long embedding vectors that are sparse, i.e. most of the entries are zeros, as most words never appear in the same context as others. Following the discussion of TF-IDF is an introduction to the **word2vec** model family, which can be used to produce short and dense embedding vectors.

2.4.1 TF-IDF

The purpose of a word embedding is to represent words as vectors in such a way that similar words are close to each other. One simple representation is the **term-document matrix** $A \in R^{|V| \times D}$, where each row represents a word from a vocabulary V and each column represents one document in a collection of D documents. The vocabulary can either be fixed or learned from the document collection. Each element a_{ij} contains a count of how many times word number i appears in document j. Thus if the size of the vocabulary is $|V|$, each document can be represented as a vector in $|V|$-dimensional space. Two documents which are similar tend to contain similar words, which means their column vectors will be similar. The same principle holds if one instead extracts the row vectors: they represent the meaning of each word in the vocabulary by the documents it tends to occur in. In this project TF-IDF is used to represent documents and not individual words, so the document flavour is the one discussed below. The motivation for the word representation is very similar and is therefore omitted.
Term-document matrices were originally created as a way for e.g. search engines to find similar documents. Their underlying principle is powerful but leaves some things to be desired: key words might only appear once or twice within a document, while words such as *it* or *the* are very frequent but not important. **TF-IDF** stands for Term Frequency - Inverse Document Frequency and attempts to balance these two conflicting constraints, by applying a weight to each element in the term-document matrix. As a result, the TF-IDF algorithm is a product of two factors. The first one

is the **term frequency**, which counts how often the word w appears in document d. It is possible to use the raw count as the term frequency:

$$\mathrm{tf}_{w,d} = \mathrm{count}(w, d) \tag{2.9}$$

However, a word appearing 100 times in a document does not make it 100 times more likely to be relevant to the meaning of the document. To reflect this nonlinear relation, it is common to use the log_{10} of the term frequency instead (after adding 1, to avoid attempting to take the log of 0):

$$\mathrm{tf}_{w,d} = \log_{10}\Big(\mathrm{count}(w, d) + 1\Big) \tag{2.10}$$

The second factor in TF-IDF gives a higher weight to words that occur only in a few documents. Terms which only appear in a few documents are useful for distinguishing these documents from the rest, while terms that occur frequently across all documents are less helpful. The **document frequency** df_w for a word w is the number of documents it appears in. Distinctive words are emphasised using the **inverse document frequency**, idf, defined using the fraction N/df_w. Here, N is the total number of documents in the collection and df_w is the number of documents that contain the word w at least once. The fewer documents in which a certain word appears, the higher this weight. Because many collections contain a large number of documents, the idf is also usually compressed with a $\log_{10}$ function:

$$\mathrm{idf}_w = \log_{10}\left(\frac{N}{\mathrm{df}_w}\right) \tag{2.11}$$

Both the factors in TF-IDF assume that the data has a natural division into documents, so this measure applies well to a data set of distinct annual reports. With the definitions above, the TF-IDF weight for word w in document d becomes:

$$W_{w,d} = \mathrm{tf}_{w,d} \cdot \mathrm{idf}_w \tag{2.12}$$

This model is quite simple and thus provides a good baseline to which other embeddings can be compared. The calculated weights are deterministic and highly interpretable. However it is important to note that TF-IDF has no understanding of context: a document which has a high weight for the word *Apple* could be either a recipe or a description of the latest MacBook. Furthermore, everything the algorithm learns is on a document level so for certain tasks it might be preferable to use a word level embedding.
The base model produces a weight for each observed word in a set of documents and is thus sensitive to spelling errors; a misspelled word gets its own weight, which increases the length of the embedding vector unnecessarily. This also affects the weight of the original, correctly spelled, word as the algorithm thinks it has seen that word fewer times. This effect can be decreased by correcting the spelling of a text before it is fed to TF-IDF. The same principle holds for different inflections of the same word - *apple* and *apples* get two separate weights. Arguably this is not helpful for representing the meaning of the text, which suggests potential benefits from lemmatising the text (bringing words back to their base form). The basic TF-IDF algorithm can also be extended in different ways; it can for example ignore all

words which do not appear in a fixed vocabulary, or consider word pairs as well as single words.

2.4.2 Word2vec Skip-Gram

TF-IDF weighted count vectors can be used to represent a word as a vector whose dimensions corresponds to documents in a collection. These representations tend to be very long ($|V|$ is around 7000 for the annual reports used here, but can be much larger) and sparse. It is possible to construct more condensed word embeddings with a lower number of dimensions ($d \approx 100$), but as a result each dimension d does not have a clear interpretation. In addition, the entries will be real-valued numbers which can even be negative. Using a more compact word representation requires subsequent models to learn fewer weights, and the smaller parameter space tends to help avoid overfitting. Dense vectors can also be better at capturing synonymy between words such as *car* and *automobile* since they tend to appear in the same contexts, while in TF-IDF these two dimensions are unrelated.
One popular method for computing embeddings is skip-gram with negative sampling (SGNS), which is included in a software package called **word2vec** and is sometimes loosely referred to as word2vec. Word2vec embeddings are static embeddings, meaning that the method learns one fixed representation for each word in the vocabulary. There are other methods for learning dynamic contextual embeddings such as **BERT** and **ELMO**, where the vector for each word changes with the context.
The intuition behind word2vec is that instead of counting word occurences, a classifier is trained on a binary prediction task: "Is word w likely to show up near word v?" The learned classifier weights are then taken to be the word embeddings. Running text can be used as implicitly supervised training data for such a classifier, avoiding the need for annotation of training data. The skip-gram method treats the target word's neighbouring word as positive examples, while randomly sampling other words from the lexicon to get negative samples. A logistic regression classifier (discussed in 2.5.1) is then trained to distinguish these two cases, and the learned weights are used as word embeddings.

2.4.2.1 The Classifier

The classifier used to learn word2vec embeddings is based on a target word w and a context window around this word. Given a tuple (w, c) containing a target word and a candidate context word c (for example (*apricot*, *jam*) or perhaps (*apricot*, *aardvark*)), it will return the probability that c is a real context word:

$$P(+|w, c) \tag{2.13}$$

The computation of this probability is based on embedding similarity: a word c is likely to appear near the target if its embedding is similar to the target embedding. For these two vectors, similarity is measured using the dot product:

$$\text{Similarity}(\mathbf{w}, \mathbf{c}) \approx \mathbf{c} \cdot \mathbf{w} \tag{2.14}$$

This measure of similarity is turned into a probability using the sigmoid function $\sigma(x)$, resulting in:

$$P(+|w,c) = \sigma(\mathbf{c} \cdot \mathbf{w}) \tag{2.15}$$

To make this a valid probability, we must also ensure that the total probability sums to one. Thus the probability that c is not a real context word for w is defined as:

$$P(-|w,c) = 1 - \sigma(\mathbf{c} \cdot \mathbf{w}) \tag{2.16}$$

Equation 2.15 gives the probability for an individual word, but the context window will contain several words. Skip-gram makes the simplifying assumption that all context words are independent, allowing their probabilities to be multiplied. With a context window containing L words $c_{1:L}$, this gives the probability:

$$P(+|w,c_{1:L}) = \prod_{i=1}^{L} \sigma(\mathbf{c}_i \cdot \mathbf{w}) \tag{2.17}$$

In summary, the probability is assigned based on how similar this context window is to the target word. To compute this probability, an embedding is needed for each target and context word in the vocabulary. Because of this, skip-gram ends up learning two embeddings for each word, one for the words as a target and one when the word is considered as a context word. These are stored in the matrices W and C, each containing $|V|$ embeddings.

2.4.2.2 Learning Skip-Gram Embeddings

Skip-gram starts from randomly initialised embeddings and then iteratively shifts the embeddings for each word w to be more similar to the embeddings of words that occur nearby in texts, and less like the embeddings of words that do not occur nearby. We consider a single piece of training data:

... *lemon, a* [tablespoon of **apricot** jam, a] *pinch of* ...

This example has a target word w (*apricot*) and four positive training instances c_{pos} in the context window. In order to train the binary classifier, negative examples are also required. Skip-gram with negative sampling uses more negative than positive examples; for each training instance (w, c_{pos}) we introduce k negative samples (w, c_{neg}). These noise words w_{neg} are chosen according to their weighted frequency:

$$p_\alpha(w) = \frac{\text{count}(w)^\alpha}{\sum_{w'} \text{count}(w')^\alpha} \tag{2.18}$$

A common choice for the weight is $\alpha = 0.75$, which gives better performance than sampling according to the unweighted frequency $p(w)$. For rare words it holds that $p_\alpha(w) > p(w)$, so they will be chosen with slightly higher probability. Intuitively, this will force the embedding to learn about rare words and not just focus on the common ones. Given a set of all positive and negative training instances, and an initial set of embeddings, the goal of the learning algorithm is to maximise the similarity of the (w, c_{pos}) pairs and minimise the similarity of the (w, c_{neg}) pairs.

Considering a single word/context pair (w, c_{pos}) and its k noise words $c_{\text{neg}_1}, ..., c_{\text{neg}_k}$, these two goals can be expressed through the following loss function:

$$\begin{aligned} L_{\text{CE}} &= -\log \left[P(+|w, c_{\text{pos}}) \prod_{i=1}^{k} P(-|w, c_{\text{neg}_i}) \right] \\ &= -\left[\log P(+|w, c_{\text{pos}}) + \sum_{i=1}^{k} \log P(-|w, c_{\text{neg}_i}) \right] \\ &= -\left[\log P(+|w, c_{\text{pos}}) + \sum_{i=1}^{k} \log \left(1 - P(+|w, c_{\text{neg}_i})\right) \right] \\ &= -\left[\log \sigma(\mathbf{c}_{\text{pos}} \cdot \mathbf{w}) + \sum_{i=1}^{k} \log \sigma(-\mathbf{c}_{\text{neg}_i} \cdot \mathbf{w}) \right] \end{aligned}$$

This loss function is commonly minimised using stochastic gradient descent. Recall that the skip-gram model learns two separate embeddings for each word: the target embedding w_i and the context embedding c_i. It is common to simply add these representations together, representing word i with $w_i + c_i$. Alternatively, just the target embedding w_i can be used.

2.4.2.3 Other Static Embeddings

The word2vec continuous skip-gram model described above is just one of many static word embeddings. An extension is **fastText**, which uses subword representations to deal with unknown words and sparsity. Each word in fastText is represented as itself plus the set of all subwords of a certain length. For example, if the subword length is $n = 3$, the word *where* would be represented with `<where>` plus all the individual word parts:

```
<wh, whe, her, ere, re>
```

Here, the boundary symbols $<$ and $>$ are added to keep track of the start and end of the original word. After each word has been given a start and end symbol, and split into chunks in this way, a skip gram embedding is learned for each chunk of length n. After this, the word *where* represented by the sum of the embeddings of the word parts.
Besides word2vec, the most widely used static word embedding is GloVe (short for Global Vectors). This embedding is based on ratios of probabilities from a word-word co-occurrence matrix, which is an alternative to the term-document matrix. Each cell of this matrix contains a count of how many times two given words appear within the same context, where a context can be defined as an entire document or a window of a given size. This allows GloVe to combine the intuition of count-based models with the linear structure used by models like word2vec and fastText.

2.4.2.4 Pretrained Static Embeddings

These more sophisticated embedding models require both expertise, good quality training data, and computational power to train. Because of this, a common approach is to use a pretrained model. This project uses a word2vec embedding of

dimension 100 provided by NLPL [9], which is trained on the Swedish CoNLL17 corpus. The training data consists of Swedish translations of 1000 sentences from the news domain and from Wikipedia.
This project also makes use of pre-trained fastText vectors which are provided by Facebook [10]. The team distributes pre-trained word vectors for 157 languages, trained on the *Common Crawl* and *Wikipedia* corpora. These fastText models are of dimension 300 and were trained using CBOW (Continuous Bag Of Words) with position-weights, character n-grams of length 5, a window of size 5, and 10 negatives.

2.4.3 Transformers

Within the field of Natural Language Processing, sequence transduction can for example refer to language modelling or machine translation tasks. A state-of-the-art sequence transduction architecture, named the Transformer model, was introduced by a team of Google scientists in 2019 [30]. Before then, the dominant models within this field were based on either recurrent or convolutional neural networks. The best performing models also connected the encoder and decoder through a so called attention mechanism. The Transformer model, on the other hand, is only based on attention. It was found to provide better results on well-established translation tasks, be easier to parallellise, and faster to train. In addition, it generalises well to a variety of different tasks.

2.4.3.1 BERT

A popular implementation of Transformers is BERT, which stands for Bidirectional Encoder Representations from Transformers. This language representation model was introduced in 2019 [8]. Unlike other language models at the time, BERT is designed to learn deep bidirectional representations from unlabelled text; this is achieved by jointly conditioning on the left and right context of each word in all layers. As a result, BERT's understanding of a word's context is based both on words appearing before and after it in a sentence. The pre-trained BERT model can be fine-tuned with just one additional output layer, in order to create state-of-the-art models for a wide range of tasks. For example, language inference and question answering tasks can be carried out without any substantial modifications to the underlying architecture.
While the concepts behind BERT are relatively simple, it achieves state-of-the-art results on eleven different natural language processing tasks. When first introduced, the BERT model pushed the General Language Understanding Evaluation (GLUE) benchmark from 72.8% to 80.5%. As of May 2021, the GLUE leaderboard is topped by a team from the Chinese company Baidu, using a language representation called ERNIE. Notably, ERNIE outperforms BERT in several Chinese language tasks. However, the BERT representation is still very popular and has made way for several variants such as ALBERT and RoBERTa.
The National Library of Sweden has trained a Swedish BERT model called KB-BERT [20]. This BERT implementation is found to outperform alternative models, including Google's multilingual M-BERT. The BERT model uses subword representations in a more sophisticated way than fastText: it splits complicated words into

pieces before embedding. This makes the model better at handling new words, and hopefully also more robust for misspelled words.

2.4.4 SentenceTransformer

BERT achieves state-of-the-art performance on sentence-pair regression tasks such as Semantic Textual Similarity, the task of determining how similar two pieces of text are. However, both sentences need to be fed into the network, causing a huge computational overhead. Using BERT to find the most similar pair in a collection of 10 000 sentences requires around 50 million inference computations, which is expected to take around 65 hours. How BERT is constructed makes it unsuitable for semantic similarity search, but also for unsupervised tasks such as clustering.
To solve this issue, Reimer et al. present Sentence-BERT [22], as a modification of the pre-trained BERT network. Sentence-BERT uses siamese and triplet network structures in order to derive semantically meaningful sentence embeddings. These embeddings can then be compared using the cosine similarity. The authors find that this modification reduces the time needed to find the most similar pair from 65 hours with BERT to around 5 seconds with Sentence-BERT, while retaining the same accuracy. A multilingual model capable of understanding Swedish has been implemented by Reimers et al. [21], and is referred to as SentenceTransformer for the remainder of this thesis.

2.5 Classification Methods

After representing each page of an annual report in numerical format, it is of interest to find the key pages in each document. In this project, the interesting pages are the income statement and balance sheet. It is tempting to plug the embedded page straight into a deep neural network, letting it learn complex relationships, and then possibly ending up with an accurate but hard-to-interpret model. First, however, we try two simpler classification models: Logistic Regression and Random Forest. Despite not being as fancy, they have some redeeming qualities: they are less data hungry, cheaper to run, and easier to interpret.

2.5.1 Logistic Regression

This discussion is based on Chapter 5 of *Speech and Language Processing* [17]. Logistic regression is a discriminative classifier which can be used in both the binary and the multi-class setting. Like naive Bayes, it is a probabilistic classifier which makes use of supervised machine learning. Logistic regression has two phases: a training phase when the system is trained using stochastic gradient descent and the cross-entropy loss, and a test phase where $p(y|\mathbf{x})$ is computed and the higher probability label is returned.
We first consider the binary classification setting, where the possible labels are $y = 0$ or $y = 1$. From a labelled training set, logistic regression learns a vector of weights and a bias term. Each weight in the vector $\mathbf{w}$ represents how important that feature is to the classification decision, providing evidence for one of the two classes. To

make a prediction on a test instance, represented by a vector of features $\mathbf{x}$, a weighted sum of the features is computed:

$$z = \mathbf{w} \cdot \mathbf{x} + b \tag{2.19}$$

Here, the resulting scalar z is the weighted sum of the evidence for the class. Nothing in the equation above forces z to lie in the range $[0, 1]$, so it is passed through the sigmoid function (also called the logistic function) in order to become a legal probability:

$$\sigma(z) = \frac{1}{1 + e^{-z}}, \quad \sigma : \mathbb{Z} \to [0, 1] \tag{2.20}$$

To ensure that the probabilities sum to 1, they are defined as:

$$P(y = 1) = \sigma(\mathbf{w} \cdot \mathbf{x} + b) \tag{2.21}$$

$$P(y = 0) = 1 - \sigma(\mathbf{w} \cdot \mathbf{x} + b) \tag{2.22}$$

Given a test instance $\mathbf{x}$, the algorithm can now compute the probability $P(y = 1|\mathbf{x})$. If the resulting probability is larger than the decision boundary 0.5, the instance $\mathbf{x}$ is predicted to belong to class 1:

$$\hat{y} = \begin{cases} 1 \text{ if } P(y = 1|\mathbf{x}) > 0.5 \\ \quad 0 \text{ otherwise} \end{cases} \tag{2.23}$$

Logistic regression is robust to highly correlated features: if two features f_1 and f_2 are perfectly correlated, some of the weight will be assigned to w_1 and some to w_2. When there are many correlated features logistic regression assigns a more accurate probability than naive Bayes, which has overly strong independence assumptions. Logistic regression thus works well on large documents or datasets, and is a common baseline model. However, the model can only account for more complex combinations of features if these are designed by hand.

2.5.1.1 Training the Model

The parameters of the logistic regression model, the weights w_i and the bias b, are learned by minimising a cost function which measures the distance between the system output and the desired output. For logistic regression it is common to use the cross-entropy loss function, and minimise it using stochastic gradient descent. The classifier output is $\hat{y} = \sigma(\mathbf{w} \cdot \mathbf{x} + b)$; the loss function should express how close this is to the correct output y, which is 0 or 1. This is achieved by conditional maximum likelihood estimation: $\mathbf{w}, b$ are chosen to maximise the log probability of the true y labels in the training data, given the observations $\mathbf{x}$. The resulting loss function is the negative log likelihood loss, also called the cross-entropy loss. For a single observation $\mathbf{x}$, the model should learn weights that maximise the probability of the correct label $p(y|\mathbf{x})$. In the binary setting this is a Bernoulli distribution:

$$p(y|\mathbf{x}) = \hat{y}^y (1 - \hat{y})^{1-y} \tag{2.24}$$

It is convenient to instead optimise the logarithm of the probability, which has the same solution since the logarithm is a monotone function. Finally, to turn the log likelihood into a loss function, a sign change is applied:

$$L_{\mathrm{CE}}(\hat{y}, y) = -\log p(y|\mathbf{x}) = -[y \log \hat{y} + (1-y)\log(1-\hat{y})] \tag{2.25}$$

In order to learn optimal weights and bias for the logistic regression classifier, the loss function needs to be minimised. The goal is to find the parameters, commonly denoted by θ ($\theta = \mathbf{w}, b$ in the logistic regression case), which minimise the loss function averaged over all samples:

$$\hat{\theta} = \underset{\theta}{\operatorname{argmin}} \frac{1}{m} \sum_{i=1}^{m} L_{\mathrm{CE}}(f(\mathbf{x}^{(i)};\theta), y^{(i)}) \tag{2.26}$$

The minimisation is commonly performed using the stochastic gradient descent algorithm, discussed in detail in [17]. However, it is not desirable for the model to learn weights which perfectly match the training data; it will almost certainly end up fitting to noisy factors which just happen to correlate with the class labels, leading the model to generalise poorly to unseen data. In order to avoid this overfitting, a regularisation term $R(\theta)$ is added to the objective function 2.26:

$$\hat{\theta} = \underset{\theta}{\operatorname{argmax}} \sum_{i=1}^{m} \log P(y^{(i)}|\mathbf{x}^{(i)}) - \alpha R(\theta) \tag{2.27}$$

Note that in this version, the objective function is rewritten to maximise the log probability instead of minimising the loss, and the factor $1/m$ is omitted as it has no effect on the argmax. The purpose of the regularisation term $R(\theta)$ is to penalise large weights. A model that matches the training data perfectly, but which uses many weights with large values to do so, is penalised more than a model which fits the data a little less well, but does so using smaller weights.
Two common ways to compute the regularisation term $R(\theta)$ are **L2 regularisation** and **L1 regularisation**. L2 reqularisation uses the squared L2 norm of the weight values, $||\theta||_2$, which is the same as the Euclidean distance between the origin and the vector θ. When θ contains n weights, the objective function becomes:

$$\hat{\theta} = \underset{\theta}{\operatorname{argmax}} \sum_{i=1}^{m} \log P(y^{(i)}|\mathbf{x}^{(i)}) - \alpha \sum_{j=1}^{n} \theta_j^2 \tag{2.28}$$

On the other hand, L1 regularisation is a linear function of the weight values. It is named after the L1 norm, which is the sum of absolute values of the weights (also called Manhattan distance). The L1 regularised objective function is:

$$\hat{\theta} = \underset{\theta}{\operatorname{argmax}} \sum_{i=1}^{m} \log P(y^{(i)}|\mathbf{x}^{(i)}) - \alpha \sum_{j=1}^{n} |\theta_j| \tag{2.29}$$

L1 regularisation and L2 regularisation are commonly used in language processing and are also known as lasso regression and ridge regression, respectively. L2 will favour a solution with many small weights, while L1 favours sparse solutions.

2.5.1.2 Multinomial Logistic Regression

In classification tasks with more than two possible classes, the logistic regression algorithm described above is not sufficient. An extension that can be used in this case is multinomial logistic regression, also known by the name **softmax regression**. Here the target y is a variable that ranges over more than two classes, and we want to find the probability that y belongs to a specific class c, $p(y = c|\mathbf{x})$, for each class $c \in C$. This is achieved using the **softmax** function which is a generalisation of the sigmoid function. For a vector $\mathbf{z}$ in $\mathbb{R}^K$, the softmax is defined as:

$$\text{softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{k} e^{z_j}}, \quad i \in [1, K] \tag{2.30}$$

As a result of this computation, $\mathbf{z}$ is mapped to a probability distribution. The input to the softmax function is similar to the input to the sigmoid function in the binary setting, but with a separate weight vector and bias for each class:

$$p(y = c|\mathbf{x}) = \frac{\exp(\mathbf{w}_c \cdot \mathbf{x} + b_c)}{\sum_{j=1}^{K} \exp(\mathbf{w}_j \cdot \mathbf{x} + b_j)} \tag{2.31}$$

The loss function for binary logistic regression, equation 2.25, also needs to be generalised for the multinomial setting. In its original form it contains two terms, one which is nonzero when $y = 1$ and one which is nonzero when $y = 0$. When there are K classes, the loss function for a single sample $\mathbf{x}$ is the sum of the log for K output classes, each weighted by y_k (the probability of the true class):

$$L_{\text{CE}}(\hat{y}, y) = -\sum_{k=1}^{K} y_k \log \hat{y}_k = -\sum_{k=1}^{K} y_k \log p(y = k|\mathbf{x}) \tag{2.32}$$

If i is the correct class, it holds that $y_i = 1$ and $y_j = 0 \ \forall j \neq i$. This means that the loss function simplifies to the log of the output probability corresponding to the correct class:

$$L_{\text{CE}}(\hat{y}, y) = -\log \frac{\exp(\mathbf{w}_i \cdot \mathbf{x} + b_i)}{\sum_{j=1}^{K} \exp(\mathbf{w}_j \cdot \mathbf{x} + b_j)} \tag{2.33}$$

The gradient for a single sample turns out to be very similar to the gradient for binary logistic regression, and the same optimisation process is used.

2.5.1.3 Model Interpretation

Is it often interesting to know not just which label a classifier has assigned, but also why it made that decision; its results should be interpretable. Because the features used in logistic regressions are often human made, one way to get insight into the classifier's decision is to understand what role each feature plays in the decision. Inspecting the magnitude of the weight w_i associated with each feature, or applying a statistical test such as the likelihood ratio test or the Wald test, gives a sense for how important each feature is in the decision making process. To ensure a fair comparison of the weights, the features should be scaled to have the same mean and variance before fitting the model.

2.5.2 Random Forest

Another classic machine learning model is Random Forest, introduced in 2001 [6]. The random forest consists of an ensemble of individual tree predictors, where each tree depends on a sampled vector which is IID for all trees in the forest. As the number of trees in the ensemble increases, the generalisation error for the classifier converges almost surely to a limit. This generalisation error depends on the strength of the individual trees, and on their correlation. The author finds that using a random selection of features to split each node yields good error rates compared to Adaboost.

2.5.3 Multi Layer Perceptron

A Multi Layer Perceptron is a supervised learning algorithm which learns a function $f(\cdot) : R^m \rightarrow R^o$ from a training data set [25]. It can be used for either classification or regression tasks, and is able to learn non-linear function approximations. The multi layer perceptron can have one or more non-linear layers between the input and output layer, also referred to as hidden layers. This is what distinguishes it from the logistic regression algorithm discussed previously. While an advantage to the multi layer perceptron classifier is its ability to learn nonlinear decision boundaries, it does require tuning a large number of hyperparameters such as the number of hidden neurons and layers, and the number of iterations.

2.6 Metrics and Evaluation

This section explains the various metrics that have been used throughout this project. The quality of different classifications has been compared using the Multi-Class $F\beta$ score. which is presented first. This is followed by a discussion of different distance metrics that have been used to quantify the similarity between either texts or multidimensional vectors. The N-gram similarity measure is used for comparing two pieces of text, and the cosine similarity is used for comparing vectors.

2.6.1 Multi-Class $F\beta$ Score

The $F\beta$ score is a metric for evaluating classifiers, which is applicable in the multi-class setting. It is computed as a modified harmonic mean of the precision and recall for each class. This overview of the metric is summarised from [23] and [24]. In order to give a concrete example, the $F\beta$ score will be discussed using the sample confusion matrix presented in Table 2.1, showing true and predicted values for three different classes. Just like in the binary setting, the precision for class A measures how many data points predicted as A actually belong in class A. In the example given here, the value is precision $= \frac{4}{4+6+3} = 0.308$. On the other hand, the recall measures how many data points from class A are predicted correctly. Class A in the example has recall $= \frac{4}{4+1+1} = 0.667$. Similarly, the class specific precision and recall can also be computed for classes B and C; the results are presented in Table 2.2. Using the precision and recall for a single class, its individual $F\beta$ score is:

$$F_\beta = (1+\beta^2)\frac{\text{precision} \cdot \text{recall}}{(1+\beta^2) \cdot \text{precision} + \text{recall}} \tag{2.34}$$

Where β is positive and real. This metric will attach β times as much importance to recall as precision; common choices are $\beta = 2$ and $\beta = 0.5$.

	True: A	True: B	True: C
Predicted: A	4	6	3
Predicted: B	1	2	0
Predicted: C	1	2	6

Table 2.1: An example confusion matrix for a three-class classifier. Each column represents the actual values for class A, B, and C respectively.

	Precision	Recall
Class A	0.308	0.667
Class B	0.667	0.200
Class C	0.667	0.667

Table 2.2: Precision and recall for the three classes, using values from Table 2.1.

2.6.1.1 N-Gram Similarity

An N-gram is a group of N consecutive characters taken from a string. A common choice is $N = 3$, where the string is split into *trigrams*: substrings of length 3. When creating trigrams, each word is considered to have two white spaces prefixed and one suffixed. With this convention, the set of trigrams in the string `cat` is:

$$\text{trigrams}(\texttt{cat}) = \{\texttt{c, ca, at, cat}\} \tag{2.35}$$

The trigram similarity between two strings can be computed by how many trigrams they share, divided by the total amount of trigrams.

$$\text{similarity}(w_1, w_2) = \frac{|\text{trigrams}(w_1) \cap \text{trigrams}(w_2)|}{|\text{trigrams}(w_1) \cup \text{trigrams}(w_2)|} \tag{2.36}$$

As a concrete example, the trigram similarity between the words `cat` and `car` is:

$$\text{similarity}(\texttt{cat}, \texttt{car}) = \frac{|\{\texttt{c, ca}\}|}{|\{\texttt{c, ca, ar, at, car, cat}\}|} = \frac{2}{6} \tag{2.37}$$

2.6.1.2 Cosine Similarity

Cosine similarity is a vector based measure which calculates the Euclidean distance between two vectors [12]. This corresponds to their normalised dot product:

$$\text{cosine similarity}(\mathbf{x}, \mathbf{y}) = \frac{\mathbf{x} \cdot \mathbf{y}}{||\mathbf{x}||||\mathbf{y}||} \tag{2.38}$$

3 Methods

The task of extracting key figures from the annual reports is divided into several distinct steps which can be evaluated separately; the workflow is presented again in Figure 3.1. This chapter presents the problem solving methodology used throughout this thesis, and the individual steps are explained and motivated. Bolagsverket keeps records of all annual reports filed by Swedish companies and a selection of these documents were made available for use in this project.

Section 3.1 gives an overview of the available data, limitations with regard to the data selection, as well as the annotation technique which following steps are dependent on. The image preprocessing methods used to remove noise from the scanned documents is described in Section 3.2. Next, Section 3.3 presents how the text is extracted using the OCR engine Tesseract. As shown in Figure 3.1, the extracted text is then used to locate the balance sheet of each document. Section 3.4 shows how the page classification task is carried out in different ways, from a simple baseline to a neural network, and how these methods are evaluated.

Key figures are then extracted from the balance sheet using a selection of methods, which are presented in Section 3.5. Successful extraction of correct key figures can only be achieved if Tesseract has identified the numbers correctly. If the image preprocessing is successful in removing noise this is expected to increase the quality of the Tesseract output, ultimately leading to a higher accuracy for the extracted key figures. Section 3.6 describes how different preprocessing methods can be compared using the extraction accuracy.

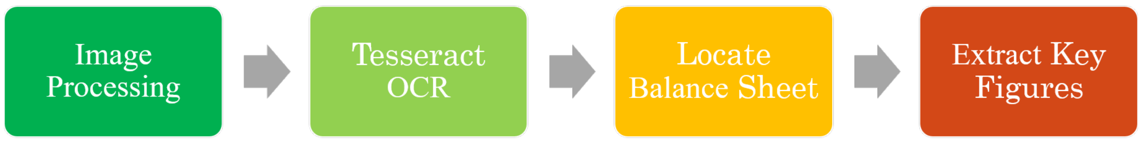


Figure 3.1: Overview of the workflow for this thesis. The first step is to apply image processing to the annual reports, after which the text is extracted using the OCR engine Tesseract. Next, the identified text on each page is used to locate the balance sheet. In the last step, key figures are extracted from the balance sheet.

3.1 Data Set

Bolagsverket receives close to half a million annual reports in paper format every year. In order to limit the time spent training models and annotating data, a subset of annual reports has been chosen. This project uses a selection of 300 annual reports from Swedish companies, received between December 2018 and July 2019. These documents are not in the original condition, meaning that they might be marked with the arrival date to Bolagsverket and/or an internal tracking number. If certain parts of an annual report were missing when it was first submitted, they might have been inserted later. These minor changes to the original condition are not expected to affect the results significantly.

The data set only contains documents which are 8 pages or shorter. According to Bolagsverket, 90% of incoming annual reports fall into this range, however large holding companies can produce annual reports which are several hundred pages long. Since the data set is not representative of those extreme cases, the solutions presented in this thesis are not guaranteed to work for such documents and they might need to be handled separately. This work instead aims to propose a solution for handling the shorter documents. It should also be noted that the provided documents are all from within a quite narrow time span of 7 months. Much older or newer documents might display different design elements such as font or layout, but the contents can also change subject to new regulations. In addition, the scanners used to scan the incoming documents have been upgraded several times. This hardware difference could affect aspects such as noise level and noise distribution, so the solutions suggested in this thesis might not perform as well on documents from other time periods.

Of the provided 300 documents, 20 are excluded since they do not contain the relevant information. Many of the excluded documents are not full annual reports but instead contain one or two pages to be appended to an already submitted document. Of the remaining 280 documents, 25% are held out during the development phase and only used for testing towards the end of the project. Thus the training data consists of 210 documents, and the test set of 70 documents. All scans are binary.

3.1.1 Comparing Training and Test Documents

This section presents a high level comparison between the training and test documents. Some overview statistics for each data set are presented in Table 3.1; each row displays a mean value across the data set. Standard deviations for all five statistics are very similar between the training and test documents, and are omitted for the sake of brevity. As seen in Table 3.1, both training and test documents have a mean length of around 7 pages. Allowing for variability due to the small size of the data sets, they are also similar in mean number of text blocks per page, lines per page, and words per page. The test documents have a higher mean distance between lines of text, possibly a result of different style templates. Since the mean values for all submitted annual reports is not known, the summary statistics presented here can not be put into perspective. Further investigation could reveal whether the selection of 300 documents are representative of annual reports in general.

Mean	Training Documents	Test Documents
Number of pages	7.1	7.2
Text blocks per page	13.7	13.9
Lines per page	29.0	29.7
Words per page	105.2	109.1
Distance between lines	22.4 pixels	25.4 pixels

Table 3.1: Summary statistics for both the training and test data sets. The presented values reflect the mean across an entire data set. Standard deviations for each of the five statistics are omitted for the sake of clarity, but their magnitudes are consistent between the two data sets. Based on these statistics, the training and test data sets appear to be comparable.

3.1.2 Document Annotation

In order to evaluate the performance of the models introduced to classify pages and extract key figures, the correct values first needed to be annotated. For the page classification task, different kinds of pages were labelled with a 2 for *Balance Sheet*, 1 for *Income statement*, or 0 for *Other*. In order to limit the time spent annotating documents, key figures have only been extracted from the balance sheet. The income statement also contains important information which could be extracted as an extension to this project.

The first key figure to be extracted is *summa tillgångar* (*total assets*), since it is an important indicator of a company's financial position which must be stated in an annual report. This value is occasionally written in a slightly larger font than other numbers, which could make it easier to locate. For *total assets*, the company should report the value for the current year but also for the previous year (usually appearing on the same line); both of these values have been annotated. Another key figure within the balance sheet is *summa eget kapital och skulder* (*total liabilities and equity*). The following relationship must always hold, otherwise the balance sheet is invalid:

$$\text{Total Assets} = \text{Total Liabilities} + \text{Total Equity} \tag{3.1}$$

In other words: since the values are equal by definition (*total assets* = *total liabilities and equity*) the same labels can be utilized for both key figures. Table 3.2 provides a more concrete example of how the annotation has been carried out. The table first shows the page labels for an example document which is 6 pages long, where the income statement is on page 3 and the balance sheet covers pages 4 and 5. Next, the correct values for the selected key figures are shown. Sometimes there is an error when scanning and an empty page is inserted in the middle of a document. These empty pages are not removed but instead labelled as *Other*; this is likely to happen again so the page classifier should learn that empty pages are not relevant.

	Label
Page labels	[0, 0, 1, 2, 2, 0]
Total assets, current year	123 000
Total assets, previous year	456 000
Total liabilites and equity, current year	123 000
Total liabilites and equity, previous year	456 000

Table 3.2: Example document annotation scheme. The first line, page labels, is used in the page classification stage. This example document is 6 pages long, with an income statement on page 3 and the balance sheet covering pages 4 and 5. Next, the key figures *total assets* and *total liabilites and equity* were annotated: both have one value for the present and one for the previous year. In total, there are four key figures which the extraction algorithm can be evaluated on.

3.2 Image Preprocessing

The training documents were studied during the annotation stage, and issues such as noise, background colours, and unusual fonts were identified. By processing these difficult documents with Tesseract, it was revealed which aspects of the documents were confusing the optical character recognition software. A selection of difficult-to-read documents were chosen to be used during the image preprocessing step. The purpose was to propose ways that the documents could be processed, in order to give Tesseract a better chance at identifying the text correctly. Tesseract was intially found to struggle particularly with large quantities of noise, so the focus has been on removing noise from the documents.
Looking back at the previous work done within this field, as detailed in Section 1.2.2, Sporici et al. [27] found a convolution based preprocessing to improve Tesseract's accuracy. However, the authors focused on studying blurry documents with low contrast, which are quite different from the binary scanned annual reports. A convolution based approach might be beneficial on these documents as well, but the choice was made to first study simpler techniques to establish a baseline for binary images. Similarly, previous work carried out by Björkman [3] found that a Gaussian filter improved Tesseract's performance. The Gaussian filter is also evaluated here, to determine whether it is useful for binary images. Six different preprocessing techniques have been compared on their ability to remove visual noise, described in Section 3.2.1. Based on the results, promising filter combinations have also been identified. Section 3.2.2 discusses the filter combinations in more detail.

3.2.1 Individual Filters

There are many filters designed to remove noise from images; four common choices are described in Section 2.2.1. Section 2.2.2 presents different morphological operations, however only one of them was suited to the task of removing noise without eroding away the text. The following six image processing techniques have been compared based on their ability to remove visual noise from challenging documents:

1. Moving Average Filter, as described in Section 2.2.1.1.
2. Gaussian Filter, as described in Section 2.2.1.2.
3. Bilateral Filter, as described in Section 2.2.1.4.
4. Median Filter, as described in Section 2.2.1.3.
5. Opening, a morphological operation described in Section 2.2.2.
6. Remove Connected Components (RCC).

Of the image processing methods listed above, the last two are only applicable to binary images. This does not present any issues, since all scanned documents are outputted in a binary format (each pixel is either black or white). The RCC approach has not been described in the Theory section since it is not a mathematical operation. In the RCC algorithm, an image analysis toolkit is used to identify all connected islands of black pixels in a binary image. A loop is then used to remove all objects that are smaller than a given cutoff size, where cutoff size larger than 50 pixels was found to risk removing the dots over the letter 'i'. This more unorthodox approach requires a lot of computational power, but might succeed in removing small grains of noise.
Each image processing technique has been applied to a total of 8 segments of difficult documents, where Tesseract was found to struggle to extract any text at all. These issues were mainly caused by noise, but also by some design choices such as underscores or grey backgrounds behind certain key figures. Based on the results, three of the six processing techniques were seen to remove noise successfully. A more quantitative comparison is carried out in Section 3.6, where it is determined how each method affects the extraction of key figures.

3.2.2 Filter Combinations

After applying each of the six processing methods individually, they were evaluated based on their ability to remove visible noise without corrupting text. Four promising filter combinations were established where the documents are first processed with one method, and then another. The results for these four filter combinations are also evaluated quantitatively in Section 3.6.

3.3 Optical Character Recognition

Before proceeding to classify different pages and reading information from them, text needs to be extracted from the scanned documents. Previous work related to extracting data from annual reports specifically was presented in Section 1.2.1. One technique described there is aimed to extract information from digital documents; while not applicable here, the methods could certainly be used on the annual reports that are submitted digitally. The section also presents a company that offers data extraction as a service. Government agencies face limitations in the services they are permitted to use for complicance and security reasons, but if allowed, the results obtained here could be compared with the paid service.
Section 1.2.1 also presents previous work, related to the more general field of detecting tables and extracting data from them. The first approach describes methods

for locating tables, and could also be used in this setting. However, at least with regard to the short annual reports studied here, locating the data tables is not very challenging since the tables tend to be full page. If Bolagsverket finds other annual reports to mix text and tables within the same page, this previous study should be revisited. Another piece of previous work uses a Hidden Markov Model to extract data from tables, but requires the table to be in a HTML format and is thus not useful for the scanned images. Lastly, two more end-to-end services are described. Since little previous work was found with regard to extracting data from scanned documents, a custom approach has been developed here.
A reliable solution for page segmentation could allow Tesseract to extract specifically the numbers within certain areas of the balance sheet, allowing for extraction of entire tables. However, training a machine vision solution has not been possible for this thesis, due to lack of sufficient quantities of annotated data. A page segmentation solution could be interesting to explore, but considering that the tables occurring in annual reports are very structured (and there is always a description of each number), the added value is not certain. The focus is instead on extracting individual key figures, which have first been annotated manually. For this task, Optical Character Recognition was carried out using Tesseract 4.1.1 using the wrapper pytesseract, version 0.3.7 [14]. Character regognition was carried out with the Swedish language setting enabled. Tesseract requires input images to be in black and white, and they should preferably also be converted to binary before processing. However, since Bolagsverket's scanners only output binary images, this did not need to be taken into account.

3.4 Classifying Document Pages

Once Tesseract has been used to extract text from the documents, the next step is to classify the pages into different categories. Correctly locating the balance sheet means that the algorithm for extracting key figures, described in Section 3.5, needs to search a smaller number of pages with a greater chance of success. This is particularly important for longer documents, which might mention a key figure such as *total assets* on multiple different pages. Directly searching for *total assets* would not only take longer time, there would also be a risk of extracting the wrong value. Since it is known in advance which part of an annual report contains which key figures, this intermediate page classification step has been proposed.
This section discusses how different supervised machine learning techniques can be used to classify pages. As described in Section 3.1.2, each page in a document has been manually assigned one out of three labels: *Balance Sheet* (2), *Income Statement* (1), or *Other* (0). Both the balance sheet and the income statement contain many key figures which reflect a company's performance. While both these page types have been located, within the scope of this thesis it has only been possible to extract numbers from the balance sheet of each document.
First of all a simple baseline page classifier is established using a string search function, detailed in Section 3.4.1. More complicated models are then introduced and compared with this baseline. Because of the relatively small data set available in this project, more data hungry classifiers such as convolutional or deep neural

networks are not suitable here. Classification has instead been performed using Logistic Regression, Random Forest, and a Multi Layer Perceptron. These three types of classifiers all expect a one dimensional array as their input, and not a page full of text. Each page is therefore first represented numerically; this can be achieved in different ways, a selection of which are presented in Section 3.4.2. After this the classifiers themselves are detailed in Section 3.4.3, including how they have been trained and evaluated.

3.4.1 Classify Using Page Titles

Most pages in an annual report have an informative title; this realisation can be used to create a baseline method for the page classification task. The page title tends to have a larger font size and Tesseract does return a proxy for this value: the height of the word's bounding box. However, the height can be distorted by noise and it was deemed more stable to use a string search approach. Starting with the document texts extracted by Tesseract, all empty rows are removed and the text is converted to lower case. The baseline page classifier uses the following simple rules:

```
1. IF line is high up on page
2. AND "balance sheet" at start of line
3. THEN page label = 2
```

Rule number 1 uses the fact that a title tends to be at the top of a page. For this task, the top 20% of non empty rows on each page are said to constitute the top of the page. Rule number 2 ensures that the search phrase is not used within a sentence but at the start of a line, making it more likely to be the page title. The same principle is used to also search for the income statement. With many, if not most, pages having an informative title, this simple baseline classifier is expected to be reasonably successful. However, the prescence of noise can cause a letter to be misinterpreted in the OCR step, and the baseline classifier does not realise that e.g. *Baiance Sheet* refers to the balance sheet.

3.4.2 Embeddings

In order to apply supervised machine learning models to classify page types, each page first needs to be represented numerically. This can be done in many different ways; this thesis evaluates four approaches of various complexity. The first, *TF-IDF*, is based on comparing the relative frequency of each word on a page with how common it is throughout the entire document. Section 3.4.2.1 explains how this embedding is applied. Another word embedding is *word2vec*, which utilizes pre-trained vector representations for each word in a vocabulary. The word2vec vectors are learned from large collections of documents, and two words will have similar embedding vectors if they tend to appear in similar contexts. Section 3.4.2.2 describes how word2vec can be used to obtain page embeddings.
A similar embedding is *fastText*, developed by Facebook, which also has an understanding of word components. If faced with a previously unseen word, fastText can still attempt to provide a numerical representation using pieces of the word. For

example, the word *keytar* might be understood to refer to a musical instrument, even if fastText has not encountered the term before. Section 3.4.2.3 explains how the fastText embedding is utilised. Finally, the fourth and last embedding used to numerically represent document pages is *SentenceTransformer*, described in Section 3.4.2.4. To prevent the classifiers from learning the specific numbers on each page, all numbers are removed before creating the page embeddings.

3.4.2.1 TF-IDF

Each page is first represented numerically using TF-IDF. TF-IDF can either learn a vocabulary from a given set of documents, or use a predefined vocabulary. The predefined vocabulary is preferred, as it will produce replicable results and not be sensitive to the quality of the output from Tesseract. Bolagsverket has published a digital example annual report [5] which is used to create the fixed vocabulary. Basing the vocabulary on only one document is justified since the language used in annual reports is very uniform, and the example document uses the same terminology as real documents. The vocabulary is learned by applying TF-IDF to the example document and then selecting the 100 words with the highest scores. Word pairs, such as *total assets*, are also permitted. A list of stopwords (short words with no inherent meaning, such as *the* and *it*), are explicitly ignored. An existing list of Swedish stopwords is used [18].

TF-IDF can be used directly to obtain an embedding vector for each page in a document. In this setting, each page is represented with a vector of dimension 100, where each entry corresponds to a word in the vocabulary described above. TF-IDF uses a normalised word count; the embedding is described in detail in Section 2.4.1. There are two main weaknesses to this approach: first, TF-IDF has no understanding of how words relate to each other. In the numerical representation, the words *asset* and *assets* will be treated as distinct and independent. For similar reasons, another weakness with TF-IDF in this setting is its sensitivity to spelling errors. If noise causes Tesseract to slightly corrupt a word, it is not recognised by TF-IDF and information will be lost.

3.4.2.2 Word2vec

Word2vec uses pretrained Swedish vectors learned from a large selection of documents, each of dimension 100. Section 2.4.2 explains the word2vec embedding in more detail and also gives more details on the pretrained version used here. The word2vec embedding is not applied to an entire page straight away, as opposed to the TF-IDF embedding. Instead the embedding vector for each word on a page is retrieved, and their average is used to represent the entire page. Note that this requires searching through a given vocabulary of several hundred thousand words, for each word on every page. This makes the word2vec page embedding quite slow to run.

3.4.2.3 FastText

The pre-trained fastText vectors are initally in dimension 300, but are compressed to dimension 100 to match the dimension of the TF-IDF and word2vec embeddings. This provides a more fair basis for comparing the different embeddings. The numerical representation of a page is taken to be the L2 normalised sum of embeddings for all words on the page, just like for word2vec. However, the fastText Python library contains a function to fetch the embeddings without searching through a vocabulary, making the fastText embedding significantly faster to run. FastText is also explained in detail in Section 2.4.2, as a variant of word2vec.

3.4.2.4 SentenceTransformers

For reference, a state-of-the-art sentence embedding is also applied. SentenceTransformer, described in detail in Section 2.4.4, is a powerful tool for numerically representing the meaning of a piece of text. Each page, after having the numbers removed, is embedded as an array of dimension 784 using SentenceTransformer. It should be pointed out that SentenceTransformer is built on the BERT embedding, which has been found to not be robust to misspellings [28]. While this embedding could excel at analysing the sentiment of the auditor's statement to determine whether they have faith in the company's leadership, SentenceTransformer might be too complex for the page classification task.

3.4.3 Classification

Once each page has been represented as a vector, the documents can be fed to a supervised machine learning algorithm. The TF-IDF, word2vec, and fastText embeddings used here all produce embedding vectors of dimension 100. SentenceTransformers, on the other hand, has an embedding dimension of 784. Each of these four representations are fed to three different classifiers, creating 12 classifiers to be trained and evaluated. The focus here is on simple and interpretable classification methods, due to the limited size of the data set and also since the page classification task is not expected to be very challenging. After all, each type of page uses very distinct terminology; for longer and more complex documents, the classification could require a more sophisticated model.
Referring back to Table 3.1, it should be noted that the average document used in this project is around 7 pages long. The income statement is generally one page long and the balance sheet 2 pages long, meaning that the data set is quite well balanced with regard to the different page classes.

3.4.3.1 Training and Evaluating Page Classifiers

As mentioned in Section 3.1, the data has been split into a training part containing 210 documents and a test part consisting 70 documents. In order to present more stable results, 5-fold cross validation has been performed on the training data, and the average scores reported. After this, each model was fitted to the entire training data and evaluated on the test data.

The page classifiers are evaluated first using the simple accuracy score, where they are compared against the accuracy of a naive classifier which always guesses the most common page type. Evaluation is also done using the multi-class Fβ score, described in more detail in Section 2.6.1. The Fβ metric is commonly used for classification tasks; here the weight has been chosen as $\beta = 2$, so recall will be considered twice as important as precision. It is costly to miss a balance sheet page (since this means the key figure extraction is certain to fail), while the main cost of a false positive prediction is the additional computational effort needed to search one more page. If many embedding schemes or classifiers produce similar results, the simplest option will be chosen. A simple method is expected to generalise better to unseen data.

3.5 Extracting Key Figures

After the balance sheet has been identified, the next task is to extract the desired key figures. In order to create the best possible conditions for success, the key figures are extracted from the *known* balance sheet pages, and not the predicted ones. This allows the page classification and number extraction steps to be evaluated and improved individually. A two step process is used to extract key figures: when asked to read the value for *total assets*, the algorithm should first find the text that best matches this key phrase. If a match is found, the y position of the text is noted down. In the next step, the algorithm searches for numbers at the same height y on the page. The two step extraction process is summarised here:

1. `Find best key phrase match`
2. `Extract number(s) from the same height`

Step 1 can be achieved in many different ways. This thesis proposes three methods of increasing complexity: exact string search, approximate string search, and embedding similarity. The first two are based on comparing the text against the key phrase but cannot account for the meaning behind the words. Embedding similarity is introduced as a more sophisticated solution, based on the fact that two phrases with similar meaning will have similar embedding, and thus their cosine similarity will be high. The cosine similarity is described in more detail in Section 2.6.1.2. This approach is expected to reflect that e.g. *sum of assets* and *total assets* refer to the same thing. The embeddings used for this approach were also used in Section 3.4.2 to represent pages of text. Section 3.5.1 describes the process of locating key phrases in more detail.

Once a best match has been identified for the key phrase, Step 2 utilises a deterministic algorithm to extract the corresponding numbers. The algorithm constructed is quite complex, but none of the individual steps are very complicated. It first locates any numbers at the same height as the identified key phrase, performs some clean up steps, and returns a value. This extraction algorithm is presented in Section 3.5.2.

Last of all, it is desirable to evaluate the performance of the different approaches for extracting key figures: some suggestions are made in Section 3.5.3.

With all the learning incorporated in the pretrained embedding models, the extraction process itself does not need to be learned from training documents. A difference in performance between different solutions will therefore only depend on how Step 1 is performed, as Step 2 is the same for all algorithms. Using a deterministic approach for extracting the numbers is thought to be appropriate in this context: an annual report is a structured document, so it is logical to read it in a structured manner. In order to be consistent with the page classification step, extraction results are still presented separately for the training and test documents.

3.5.1 Locating Key Phrases

As described above, a key figure will be extracted by first locating the desired key phrase (e.g. *total assets*) and then extracting the corresponding number(s). A baseline keyword extraction algorithm simply searches for the exact search string on all rows of the balance sheet. This method is then improved using a fuzzy string matching approach which also allows structurally similar strings. This thesis also proposes using embedding similarity to measure the similarity in *meaning* between two pieces of text. In total, five different methods are used to locate key phrases. These five approaches are introduced, in this order, in the sections below:

1. Exact String Search
2. Fuzzy String Search
3. Embedding Similarity using Word2vec
4. Embedding Similarity using FastText
5. Embedding Similarity using SentenceTransformer

3.5.1.1 Exact String Search

Just like in Section 3.4, a baseline is established using a simple string search algorithm. The underlying reasoning is that if one finds the words *total assets* within the balance sheet, the correct number should be next to this phrase (if the OCR output was 100% correct, this would indeed be the case). Searching the text for an exact match to the target string provides a good baseline to improve on.

3.5.1.2 Fuzzy String Search

One can also extend the exact string search by searching approximately for the string, this is commonly called fuzzy string matching. If the search phrase is *summa tillgångar*, it is compared against all row subsets containing two words. As a concrete example, a row in the OCR output might read:

```
summa tillgångar 123 456
```

This row is split into three parts: *summa tillgångar*, *tillgångar 123*, and *123 456*. The search phrase is then matched against these three row subsets and a similarity score is computed. Fuzzy string matching can be performed using different algorithms, the most common one being trigram similarity which is described in Section 2.6. Each key figure is only listed once within the same document, so the best overall match is returned for each document.

3.5.1.3 Embedding Similarity

A more sophisticated approach uses embedding vectors. As discussed in Section 2.4, these pre-trained word vectors convey the meaning behind each word and how they relate to each other. For this reason, embeddings could be used to search the text for certain phrases. The key phrase, e.g. *total assets* is represented numerically as a vector. An algorithm then searches the balance sheet pages for an entry which has a low cosine distance to the target embedding, implying that their meaning is similar. To match against row subsets of the correct length, the line *summa tillgångar 123 456* is once again split into three parts: *summa tillgångar*, *tillgångar 123*, and *123 456*. For each of the three segments an embedding vector is computed, and then the cosine distance to the key phrase embedding is found.
This embedding similarity approach to locating the correct line of text is implemented using three different embeddings: word2vec, fastText and sentenceTransformers. With word2vec, an embedding is retrieved for each word and their average is then computed. If a word is not found in the vocabulary, its embedding is the zero vector. FastText uses the same approach for generating a sentence embedding, but also takes into account different parts of a word. Finally, as the name suggests, SentenceTransformer is made for creating a representation of an entire sentence. These three embeddings are all described in detail in Section 2.4.

3.5.1.4 Similarity Score Cutoff

The fuzzy matching algorithm finds a similarity score between the key phrase and the best match within the scanned text. Similarly, the embedding similarity approach creates a vector representation of both phrases and then computes a similarity score. Two embedding vectors representing similar sentences will be close to parallel, so in this setting the cosine similarity is used. As described previously, each search method will return the best match within a specific document. If a similarity score cutoff is introduced, the algorithm will only return a match if:

$$\text{similarity}(\text{key phrase}, \text{best match}) > \text{score cutoff} \tag{3.2}$$

Applying a score cutoff is thought to force the algorithm to be more selective, only returning a value if the match is of a certain quality. For the most promising extraction algorithm, different similarity score cutoffs are used to study whether this affects the accuracy.

3.5.2 Extracting Key Figures

Tesseract tends to split up tables so that entries on the same line appear on different rows of the output. This complicates the task of extracting numbers corresponding to a certain key word. However, the OCR output provides a bounding box for each identified word; once the target string is found, the algorithm searches for numbers at the same height on the page.
When a matching string is found, using one of the five methods described in Section 3.5.1, the word's y coordinates on the page are saved. An algorithm then looks for any numbers that appear at the same height $\pm$ a tolerance of 50% of the bounding

box height. In annual reports, large numbers are often written with spaces inbetween and are thus identified as different words by Tesseract, e.g. 1 234 567. To remedy this, the algorithm merges words that are close enough together; a tolerance of 30 pixels is used. The search and merge tolerances are fixed for all documents but this could be sensitive to a drastic change in font size. Finally, as the OCR engine outputs a string, a conversion to the float datatype is attempted. When scanning a noisy document, Tesseract occasionally mistakes the noise for punctuation or minus signs. Since the true values can contain decimal delimiters and minus signs, efforts have been made to identify and remove false symbols without corrupting the correct values.

3.5.3 Performance Evaluation

Different extraction algorithms are evaluated using their accuracy score. It should be pointed out that the accuracy is not the only metric that can be used here. Accuracy is chosen since the ideal extraction algorithm will read out each key figure completely correctly. Other metrics, for example measuring the difference between the expected and extracted values, could also be introduced. A total accuracy for the extraction can be computed through $n_{\text{correct}}/n_{\text{docs}}$, where n_{docs} is the number of documents processed and n_{correct} is the number of documents where an exact match was found with the annotated value. However, it is interesting to determine *why* the number extraction fails for certain documents. The total accuracy metric can thus be expanded:

$$\text{tot}_{\text{acc}} = \text{match}_{\text{pct}} \cdot \text{precision} \tag{3.3}$$

Where the two newly introduced metrics are computed as follows:

$$\text{match}_{\text{pct}} = \frac{n_{\text{extracted}}}{n_{\text{docs}}}, \quad \text{precision} = \frac{n_{\text{correct}}}{n_{\text{extracted}}} \tag{3.4}$$

This gives additional information regarding the fraction of documents where a match was found, and in how many of those the identified number was correct. A different search algorithm could return a higher match percentage, but potentially at the expense of a lower precision. The quality of the OCR output can be measured directly with the match percentage, since it is easier to find matching key phrases in a text mass without errors. If the text has been corrupted to a high degree, either by noise or OCR errors, the numbers are also expected to be corrupted. However, while an incorrect word can be checked against a vocabulary, there is no way to recover an incorrectly scanned number. The image processing step is very important since it can prevent these errors.

3.6 Evaluating the Image Processing

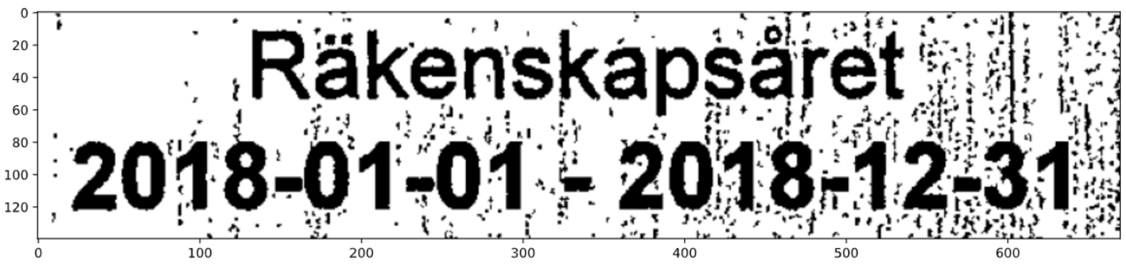
As a final step, this section introduces a quantitative way of measuring whether the image processing actually improves the extraction of numbers. A comparison is carried out using the most successful key figure extraction algorithm found in Section 3.5. Documents are processed in different ways before being passed to Tesseract, after which the extraction algorithm is used to extract the key figures. A filter is considered successful if the average extraction accuracy is significantly increased. A full examination of the filter effects on all 280 documents has not been possible within the scope of this thesis; instead, evaluation is done using 20 documents that were found to be difficult during the extraction phase. Either they are too noisy for the key phrase to be extracted, or the numbers themselves are corrupted by Tesseract due to a design choice such as font or background colour. The average extraction accuracy for the original documents is compared to the extraction accuracy obtained using the processed versions. The result gives an indication as to which filters are suitable for removing noise from the scanned documents.

4 Results

This chapter contains results from the different tasks described in Chapter 3. Results from the image processing step are shown in Section 4.1. After this, Section 4.2 presents results from the page classification, identifying promising page representations and classifiers. Different methods were used to extract key figures from the balance sheet; Section 4.3 displays the results from this task. Last of all, Section 4.4 quantifies how different filters affect the extraction of key figures.

4.1 Image Preprocessing

This section presents a qualitative comparison of different document preprocessing methods. The goal was to remove noise and other potentially confusing elements, to make the text as clear and legible as possible. Preprocessing methods have been evaluated on a selection of both nicely scanned and noisy documents, to ensure that document quality is increased overall. For illustration purposes, each preprocessing technique is applied to one part of a noisy document. This document is chosen since it clearly illustrates the strengths and weaknesses of each filter. Figure 4.1 shows the original version of the image, taken from the front page of an adverse document.



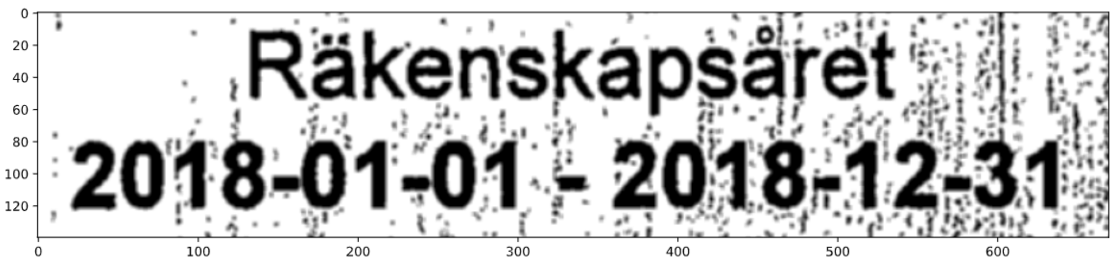
Räkenskapsåret
2018-01-01 - 2018-12-31
0
20
40
60
80
100
120
0
100
200
300
400
500
600

Figure 4.1: A section of a scanned document with no preprocessing applied, taken from the front page of a very noisy document. The text reads *Financial Year 2018-01-01 - 2018-12-31.* While legible to a (Swedish speaking) human, the text is not interpretable by Tesseract because of the streaks of noise, presumably from the scan.

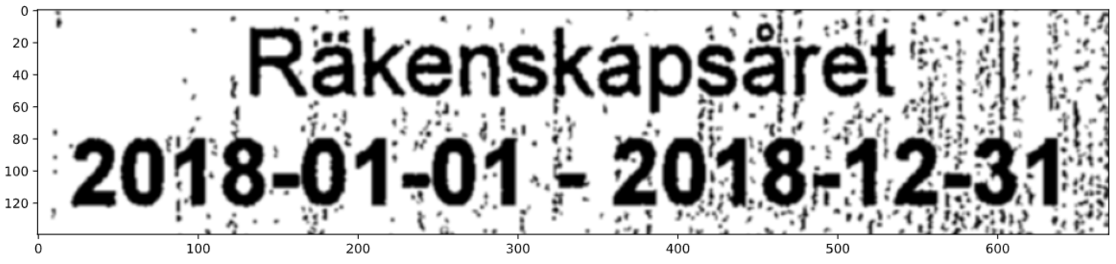
The six preprocessing techniques described in Section 3.2.1 are all applied to the text segment in Figure 4.1. The results are shown in Section 4.1.1 below. In order to limit the number of figures presented here, each filter is shown with the best choice of parameters for the task. Based on how the individual preprocessing techniques perform, promising filter combinations are suggested. The result for these two-step filters are presented in Section 4.1.2.

4.1.1 Individual Filters

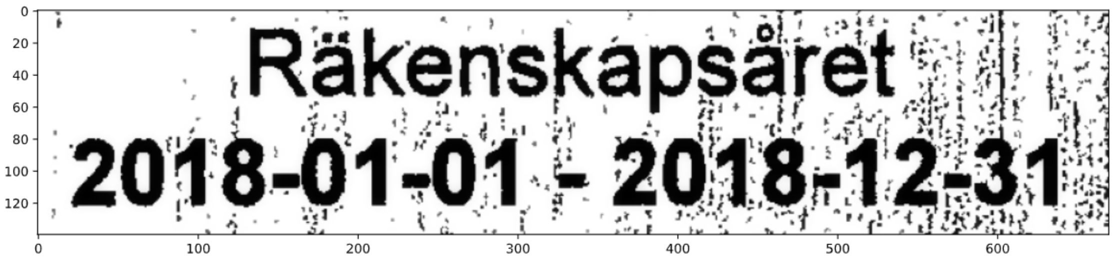
In order to remove noise from the scanned images, six filters have been evaluated: moving average, Gaussian, bilateral, median, opening, and RCC. A qualitative comparison of the filters is performed by studying their effect on parts of documents; here they are all applied to the noisy image shown in Figure 4.1. Results from the first three filters are shown in Figure 4.2, while the rest are presented in Figure 4.3.



(a) Reference text after being passed through a moving average filter with kernel size $s = 3$. The main effect is a smoothing of the entire image, but the filter fails to remove the background noise in this binary image.

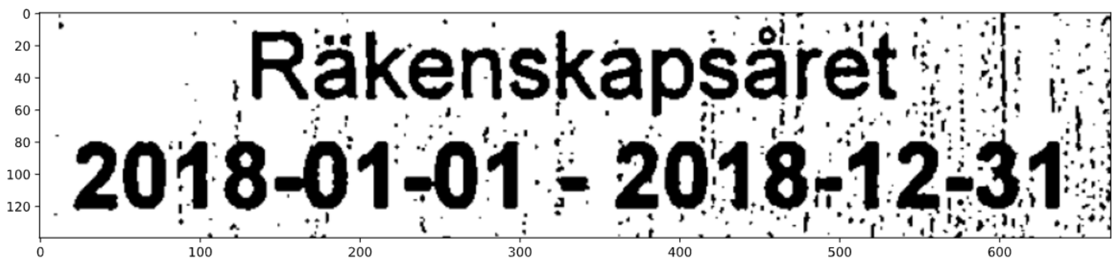


(b) Reference text after being passed through a Gaussian filter with kernel size $s = 3$. The image is blurred, but the noise remains since it is not Gaussian in distribution.

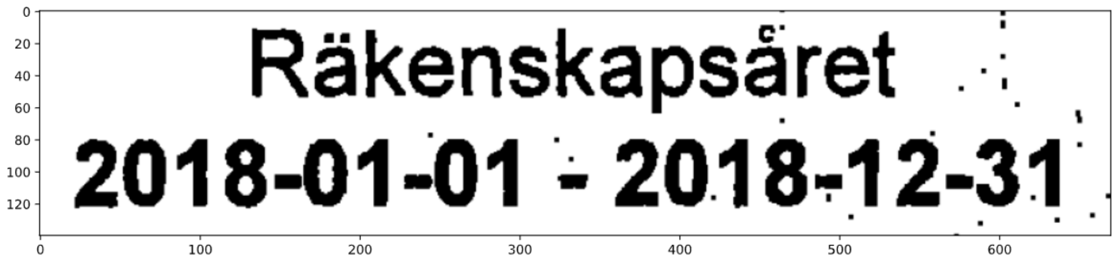


(c) Reference text after being passed through a bilateral filter with *filter_size* = 15, *sigma_colour* = 100, and *sigma_space* = 150. The background noise is somewhat decreased, but the bilateral filter does not appear suited to binary images.

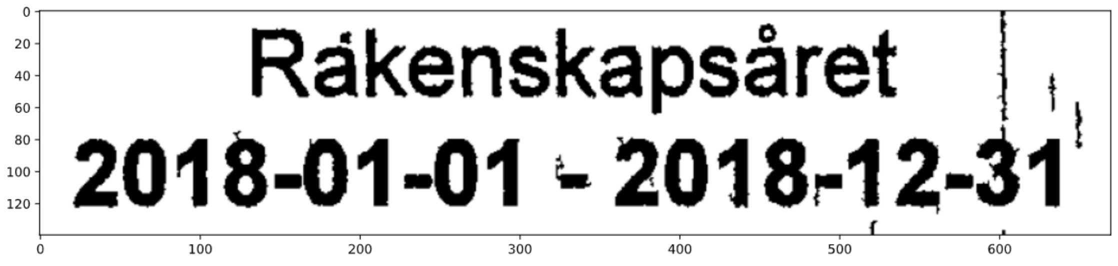
Figure 4.2: Results obtained by processing the example document, shown in Figure 4.1, using three different filters. A moving average filter is shown in 4.2a, a Gaussian filter in 4.2b, and a bilateral filter in 4.2c. Each filter is intended to remove background noise, but they all appear ill suited for binary documents.



(a) Reference text after being passed through a median filter with kernel size $s = 3$. The filter successfully removes smaller speckles of noise, while leaving the text intact.



(b) Reference text after being passed through an opening filter with kernel size $s = 3$. This filter excels at removing noise while leaving large fonts intact, but is found to erase certain elements of thinner fonts (not shown here).



(c) Reference text after being passed through an RCC filter with cutoff size = 40. Most of the noise is removed, but a few streaks remain. The filter also removes one dot above the *ä* and leaves some characters, especially numbers, with rough edges.

Figure 4.3: Results obtained by processing the example document, shown in Figure 4.1, using three different filters. A median filter is shown in 4.3a, an opening filter in 4.3b, and a RCC filter in 4.3c. The latter two filters are more successful in removing noise but also risk corrupting the text, especially for smaller font sizes.

4.1.2 Filter Combinations

Based on the results from preprocessing documents with six different filters, it was decided that the moving average, Gaussian, and bilateral filters are not suitable for filtering binary images. However, the median, opening, and RCC filters are promising and were studied further. In addition, studying how each filter affects the different images, promising filter combinations were identified. Based on the results from the first six filters, shown in Figure 4.2 and Figure 4.3, documents were processed with the following four filter combinations:

1. Median (Opening (kernel size = 3), kernel size = 3)
2. Median (Opening (kernel size = 3), kernel size = 5)
3. Opening (Median (kernel size = 3), kernel size = 3)
4. RCC (Opening (kernel size = 3), cutoff = 25)

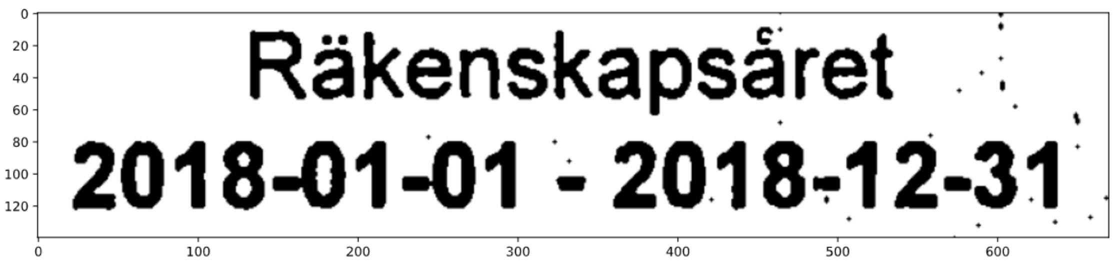
Applying a median filter after the opening filter, as in number 1 and 2 above, appears promising. The opening filter can break up islands of noise, which are then removed or shrunk further by the median filter. A logical next step is to also try the opposite order: option 3 applies an opening filter after a median filter. This preprocessing technique might preserve text in a way which Tesseract finds easier to interpret. Last, it was tried to use a RCC filter after the opening filter. Opening breaks up the noise into smaller pieces, allowing a lower cutoff size for RCC which corrupts the text less. RCC is a very slow algorithm to run; applying it after removing most of the noise with another filter saves computational time.
The results from applying these four filters to the example text are shown in Figure 4.4. All four filters appear promising for removing heavy noise, but at the expense of distorting the text to varying degrees. Particularly for thinner fonts, Tesseract could struggle to interpret text that has been processed by these filters. A quantitative comparison of these 7 preprocessing techniques, and the original documents, is presented in Section 4.4.

4.2 Classifying Document Pages

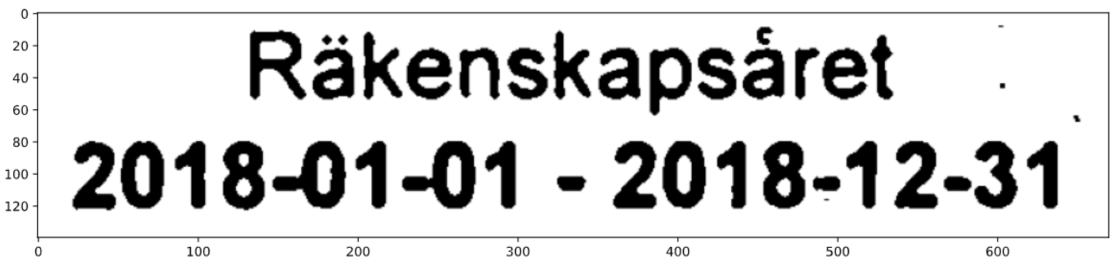
This section presents the results from the page classification step. To start with, each page is represented numerically using four different embeddings of increasing complexity: TF-IDF, word2vec, fastText, and SentenceTransformer. All the embeddings are in dimenision 100 except for SentenceTransformer, which uses embedding vectors of dimension 784. In order to get a visual representation of the four different page embeddings, a PCA representation has been generated for each one. The training data contains around 1500 pages, which have been labelled as follows:

1. 0: Other
2. 1: Income Statement
3. 2: Balance Sheet

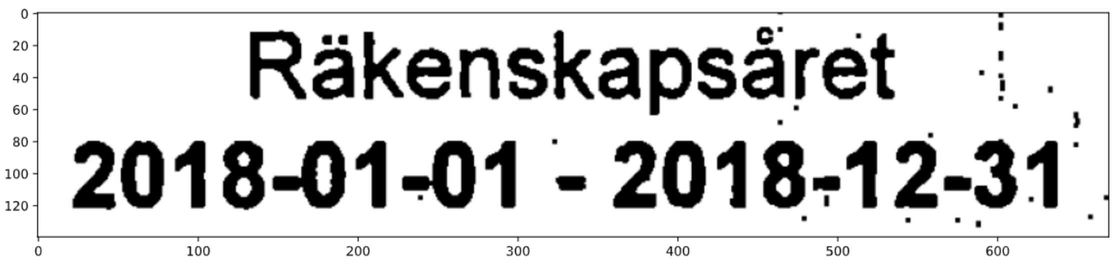
If all three page types appear easily separable from the PCA representations, this implies that the classification task should not require a very sophisticated algorithm. Section 4.2.1 presents the results from computing the first two PCA dimensions for each distinct page embedding. Next, classifiers are trained on these embedded pages, with the aim of successfully distinguishing the page types. As described in Section 3.4, a Logistic Regression, Random Forest, and Multi Layer Perceptron classifier have been applied to each kind of page embedding, resulting in 12 trained classifiers. These are evaluated against a baseline classifier, which looks for e.g. *Balance Sheet* within the title of each page and makes its prediction based on the results. A naive classifier is also introduced, which always guesses the most common page type (0: Other). Results from the classification step are presented in Section 4.2.2.



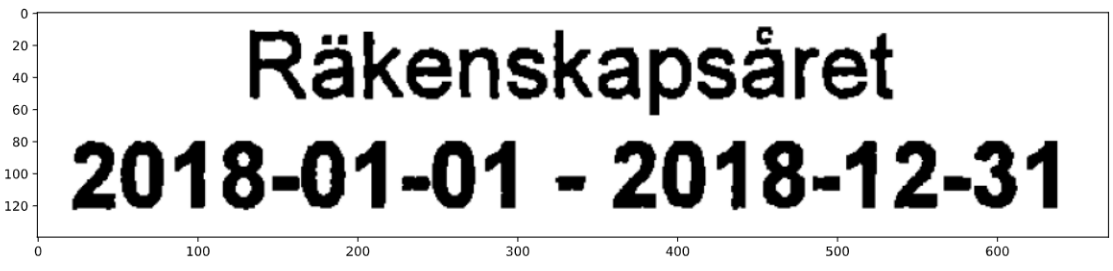
(a) Reference text after being passed through an opening filter with kernel size $s = 3$, and then a median filter with kernel size $s = 3$. Some noise remains in the image.



(b) Reference text after being passed through an opening filter with kernel size $s = 3$, and then a median filter with kernel size $s = 5$. Almost all noise has been removed, but the corners are not as sharp as before.



(c) Reference text after being passed through a median filter with kernel size $s = 3$, and then an opening filter with kernel size $s = 3$. Some noise remains in the image.

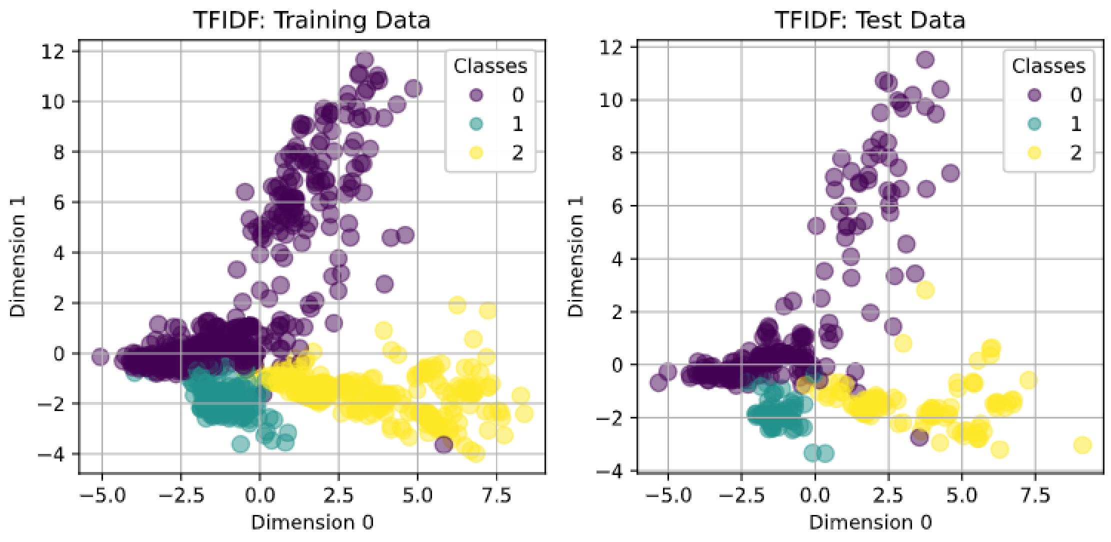


(d) Reference text after being passed through an opening filter with kernel size $s = 3$, and then a RCC filter with cutoff size = 25. No noise remains in the image, and the text is not noticeably distorted.

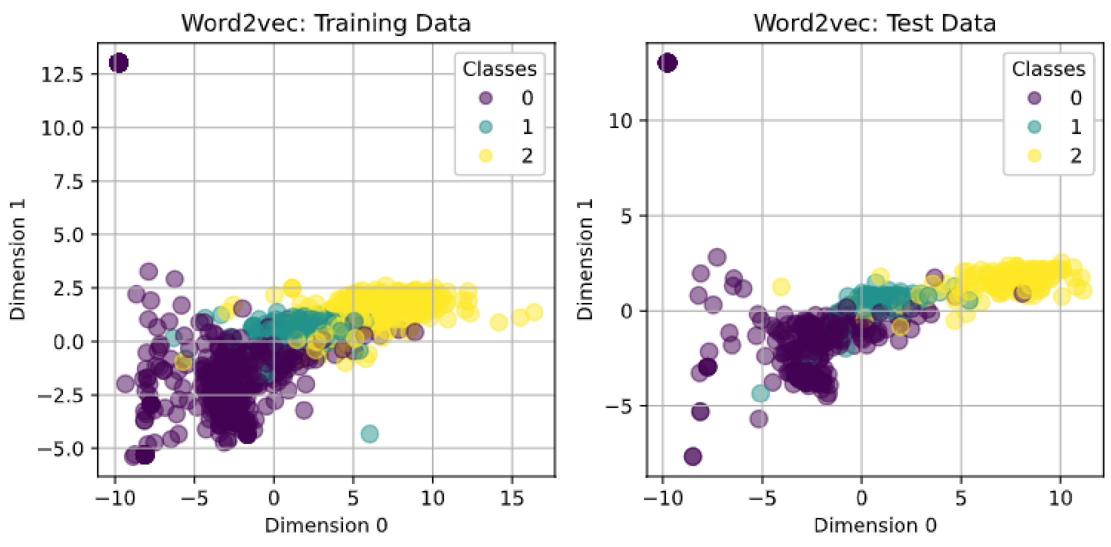
Figure 4.4: Results obtained by processing the example document, shown in Figure 4.1, using four different filter combinations. 4.4a and 4.4b both show a median filter applied after an opening filter, where the median filter has kernel size $s = 3$ and $s = 5$, respectively. The opposite, an opening filter applied after a median filter, is shown in 4.4c. Last of all, 4.4d shows a RCC filter applied after an opening filter.

4.2.1 Visualising the Embeddings

This section gives a visual overview of the different page embeddings. Figure 4.5 shows the first two PCA dimensions for pages embedded using TF-IDF and word2vec. For TFIDF, the page classes appear to be nicely separable in both the training and test case. The page classes are slightly more mixed with the word2vec embedding; however, classification should still be possible using a simple classifier.



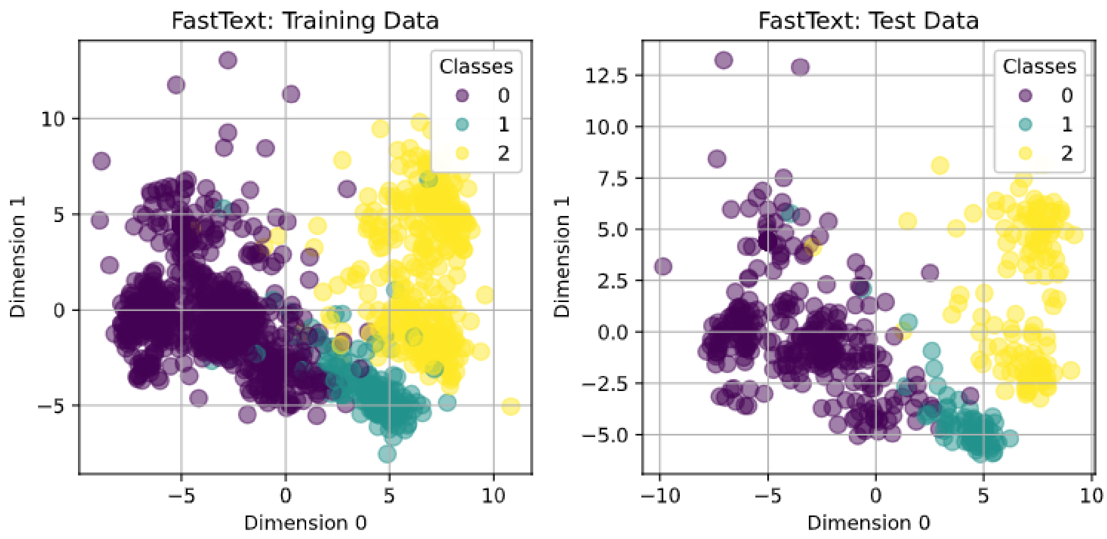
(a) PCA representation of the TFIDF embedded pages; the first two dimensions are shown. The three classes appear to be easily separable in some parameter space.



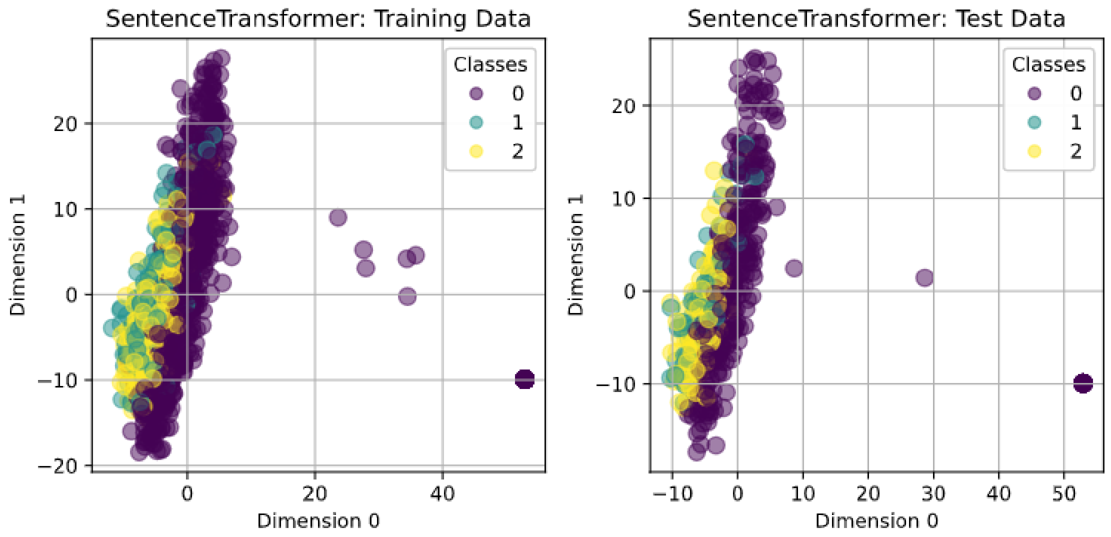
(b) PCA representation of the word2vec embedded pages; the first two dimensions are shown. The three page classes are somewhat separated. Outliers in the upper left corner correspond to empty pages (which were intentionally left in the dataset).

Figure 4.5: The first two PCA dimensions for pages embedded using TFIDF, shown in 4.5a, and using word2vec, shown in 4.5b. In both cases, the PCA transformation was learned from the training data and applied to both training and test data.

Figure 4.6 shows the first two PCA dimensions for pages embedded using fastText and sentenceTransformer. For fastText, the three page classes appear to be mostly separable and there is a high chance of success with a simple classifier. However, the SentenceTransformer embedded pages display a lot of overlap and a more sohisticated classifier might be required to successfully separate the three page types.



(a) PCA representation of the fastText embedded pages; the first two dimensions are shown. The three classes appear to be easily separable in some parameter space.



(b) PCA representation of the sentenceTransformer embedded pages; the first two dimensions are shown. The classes appear quite hard to distinguish, with class 0 slightly separated from the others. outliers in the bottom right corner correspond to empty pages (which were intentionally left in the dataset).

Figure 4.6: The first two PCA dimensions for pages embedded using FastText, shown in 4.6a, and SentenceTransformer, shown in 4.6b. The PCA transformations were learned from the training data and applied to both training and test data.

4.2.2 Classifying Pages

This section presents the results from a total of 14 different page classifiers. The naive classifier always guesses class 0 (*Other*), while the baseline classifier uses the page titles to predict page labels. In addition, 12 more page classifiers were presented in Section 3.4: there are four alternative ways of representing each page, and three different classifier types. With the large amount of classifiers to train, an extensive hyperparameter optimisation has not been implemented. The key parameters used for each classifier were:

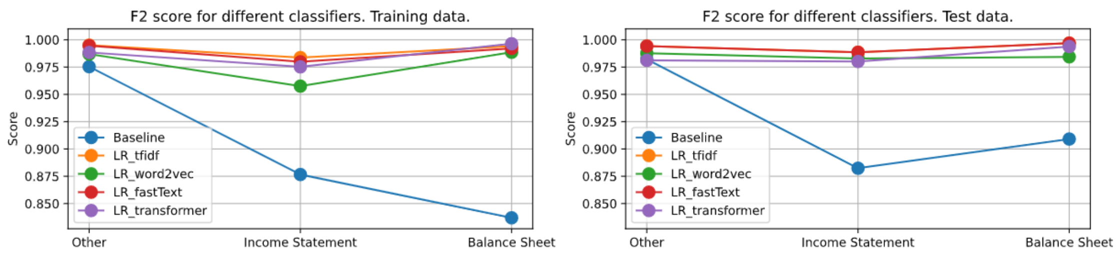
1. LR: l2 penalty, multinomial loss, class weights (0:0.2, 1:0.4, 2:0.4).
2. RF: 100 estimators, minimum 1 samples at a leaf node.
3. MLP: Hidden layer sizes: (100, 50, 30), ReLU activation, lbfgs solver.

The page classifiers are evaluated using two metrics: the simple accuracy, and the F2 score. Accuracies for all classifiers are presented in Table 4.1. For the trained classifiers, the training accuracy represents an average from the 5-fold cross validation. The Random Forest classifier trained on TF-IDF embedded documents achieves the highest accuracy on both the training and test data, and scores similarly on both.

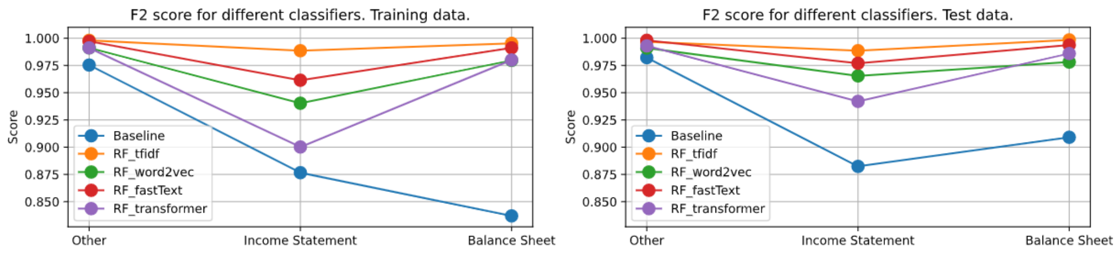
	Train Accuracy	Test Accuracy
Naive Classifier	0.6016	0.6095
Baseline Classifier	0.9276	0.9507
LR-TFIDF	0.9933 ± 0.0030	0.9941
LR-word2vec	0.9832 ± 0.0076	0.9862
LR-fastText	0.9920 ± 0.0050	0.9941
LR-Transformer	0.9886 ± 0.0058	0.9842
RF-TFIDF	**0.9960** ± 0.0039	**0.9961**
RF-word2vec	0.9812 ± 0.0195	0.9842
RF-fastText	0.9906 ± 0.0071	0.9941
RF-Transformer	0.9759 ± 0.0062	0.9842
MLP-TFIDF	0.9899 ± 0.0033	0.9901
MLP-word2vec	0.9879 ± 0.0078	0.9921
MLP-fastText	0.9913 ± 0.0074	0.9961
MLP-Transformer	0.9886 ± 0.0057	0.9803

Table 4.1: Accuracies for different page classifiers. The naive classifier has an accuracy around 60%, while the baseline algorithm achieves over 92% accuracy. All other entries correspond to a page classifier trained on embedded pages. For both the training and test data, a Random Forest classifier trained on TF-IDF embedded pages performs the best. However, all trained models outperform the baseline significantly. Training accuracy is shown ± one standard deviation.

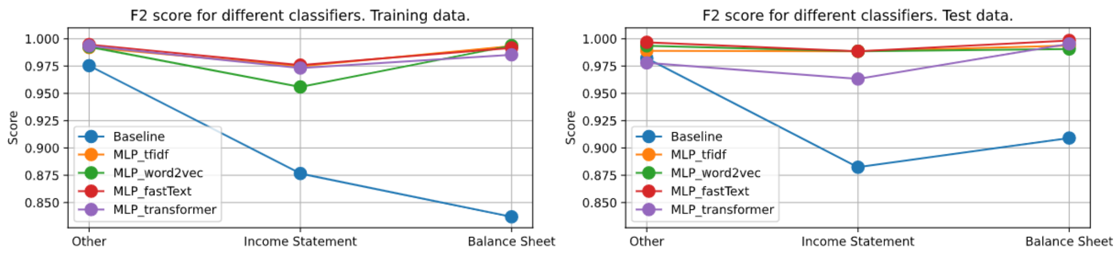
The trained classifers are also evaluated using the F2 scores for each page type. Average values are presented from the cross validation on the training data, after which each classifier is trained on the entire training data set and finally evaluated on the test data. Results are shown in Figure 4.7. As a reference, the scores from the baseline page classifier (using page titles to classify page types) are also shown.



(a) F2 scores for each page class, obtained with the Logistic Regression classifiers. All LR classifiers perform well, and are almost indistinguishable on the test data.



(b) F2 scores for each page class, obtained with the Random Forest classifiers. The best performance is achieved using the TF-IDF embedding, while the Sentence-Transformer embedding gives low scores on both training and test data.



(c) F2 scores for each page class, obtained with the Multi Layer Perceptron classifiers. The classifiers perform similarly on the training data, but the SentenceTransformer embedding does not generalise as well to the unseen test data.

Figure 4.7: Results from the three types of page classifiers. Logistic Regression is shown in 4.7a, Random Forest in 4.7b, and Multi Layer Perceptron in 4.7c. Each plot shows results for all four page embeddings, with the baseline classifier as a reference. For training data, each score represents an average over the cross validation; for test data, the score was calculated directly. Results are similar for train and test data.

As seen in Figure 4.7, many classifiers approach a perfect F2 score of 1.0 for one or more page types; the techniques applied here seem to be sophisticated enough to classify pages very well. In fact, SentenceTransformer displays slightly lower scores on the test data for both the Random Forest and Multi Layer Perceptron classifiers. This might be remedied using hyperparameter optimisation, but else the embedding might not be suitable for this task. The classifier achieving the highest accuracy, the RF-TFIDF classifier, also performs well with regards to the F2 score. For both training and test data, the RF-TFIDF classifier outperforms other Random Forest classifiers. Overall, the TFIDF and fastText embeddings appear most suitable.

4.2.2.1 Logistic Regression Feature Importance

As mentioned in Section 2.5.1.3, the weights learned by Logistic Regression reflect how important each feature is to the classification task. In combination with a word embedding such as TF-IDF, where each dimension corresponds directly to a word in the vocabulary, important insight can be gained into the predictions made by this model. Table 4.2 shows the word with the highest and lowest weight, for each of the three page types. Here, a page containing the word *result* greatly increases the probability that it will be labelled as *Income Statement*. As the income statement discusses the year's financial results, this weight makes logical sense.

	Weight	Word (Swe)	Word (Eng)
Balance Sheet: Highest	$+0.8266$	eget	own
Balance Sheet: Lowest	-0.5568	not	note
Income Statement: Highest	$+0.9807$	resultat	result
Income Statement: Lowest	-0.4844	kapital	capital
Other: Highest	$+0.6301$	belopp	amount
Other: Lowest	-0.9599	summa	sum

Table 4.2: A selection of weights learned by the LR-TFIDF classifier. The highest and lowest weight is presented for each of the three page classes, and the corresponding word from the TFIDF vocabulary is displayed. The first line displays a high positive weight for the word *eget*; if this word appears on a page, it increases the probability that the classifier will predict the page to be *Balance Sheet*.

4.3 Extracting Key Figures

This section presents the results from the key figure extraction. A simple benchmark model uses an exact string search to search the text mass for each key phrase, and then extracts any numbers identified on the same height on the page. This algorithm is extended by using fuzzy string search, which returns the text line which has the highest trigram similarity, described in Section 2.6.1.1, with the key phrase. Section 3.5.1.4 discusses the introduction of a similarity score cutoff; fuzzy matching is here carried out with a score cutoff of 0. An even more sophisticated solution uses embedding similarities between the text mass and the key phrase to locate the correct location in the text. The embedding similarity approach is implemented using word2vec, fastText, and SentenceTransformer.

Results from all five extraction algorithms are presented in Table 4.3. Four key figures have been annotated; the table displays the mean and standard deviation for these four values. The training data and test data are presented separately, to increase comparability with the results in Section 4.2.2. It can be seen that the fuzzy string search algorithm is most successful in extracting key figures, achieving an accuracy of 92.9% on training documents and 89.6% on test documents.

	Training Accuracy	Test Accuracy
Exact String Search	0.6857 ± 0.0221	0.6679 ± 0.0274
Fuzzy String Search	**0.9286** ± 0.0190	**0.8964** ± 0.0292
Word2vec	0.8548 ± 0.0527	0.8179 ± 0.0396
FastText	0.8536 ± 0.0607	0.8250 ± 0.0355
SentenceTransformer	0.8714 ± 0.0430	0.8321 ± 0.0421

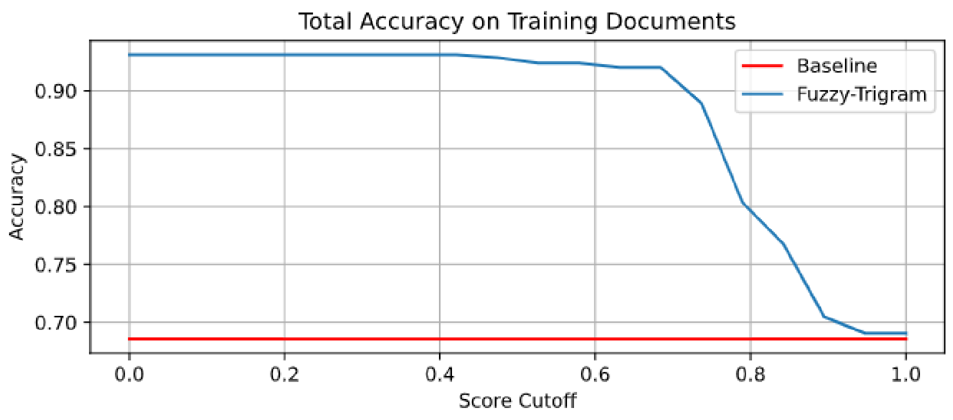
Table 4.3: Accuracies for five different search algorithms used to extract key figures. Each value represents the mean value for all four key figures ± one standard deviation. Results are presented separately for the training and test data sets. The fuzzy string search algorithm performs significantly better than the other approaches, achieving 93% accuracy on training and 90% on test documents.

4.3.1 Similarity Score Cutoff

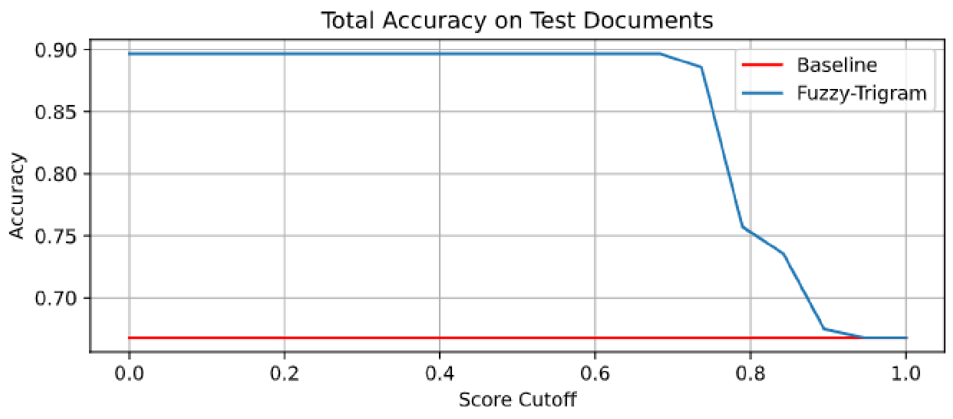
For the most successful extraction algorithm found in the previous step, the effects of a similarity score cutoff is studied. The cutoff will force the algorithm to be more confident about its guesses, and only return a match if it is of a certain quality. Table 4.3 presented a fuzzy string search algorithm using a cutoff = 0, and setting the cutoff = 1 is identical to the exact string search algorithm. Results for training and test documents are shown separately in Figure 4.8. It can be seen that the accuracy decreases, more or less monotonically, as the score cutoff increases. For any given cutoff, the fuzzy algorithm performs better of equal to the baseline exact string search algorithm. It appears that using a score cutoff of 0 is a good choice for achieving a high extraction accuracy.

4.4 Evaluating the Image Processing

Last of all, this section presents the results from the quantitative comparison of different image processing techniques. 20 challenging documents have been preprocessed in a total of 7 different ways. After this, the annotated key figures are extracted using the most promising algorithm, fuzzy string search. Results are presented in Table 4.4. The extraction accuracy obtained with the original documents is also displayed, as a baseline the different filters can be compared to. It is seen that at least on this selection of annual reports, the average extraction accuracy is increased from 73.8% to 83.8% by applying the RCC filter to the documents. Notably, the RCC filter allows the extraction algorithm to find a text match for 100% of key figures, and a high precision is also achieved. These two additional metrics are discussed in more detail in Section 3.5.3. None of the other filters display a significant increase in performance relative to the original documents, but this might not be the case if they are compared on a larger scale.



(a) Average extraction accuracy achieved with fuzzy string search, as a function of similarity score cutoff. Result for training documents.



(b) Average extraction accuracy achieved with fuzzy string search, as a function of similarity score cutoff. Result for test documents.

Figure 4.8: Key figure extraction accuracy as a function of similarity score cutoff, obtained using the fuzzy string search algorithm. Results from training documents are shown in 4.8a, and test documents in 4.8b. 20 linearly spaced cutoffs between 0 and 1 were used. The baseline represents the exact string search algorithm.

Preprocessing	Accuracy	Match %	Precision
None	0.7375 ± 0.0650	0.9500	0.7763
Opening	0.5625 ± 0.0217	0.9500	0.5931
Median	0.7125 ± 0.0415	0.9500	0.7500
RCC	**0.8375** ± 0.0650	1.0000	0.8375
Median-Open3	0.5625 ± 0.0740	0.8750	0.6446
Median-Open5	0.5875 ± 0.0545	0.9250	0.6352
Open-Median	0.6625 ± 0.0415	0.9500	0.6986
RCC-Open	0.6000 ± 0.0354	0.9250	0.6499

Table 4.4: Extraction results obtained for 20 challenging documents, preprocessed using 7 different techniques prior to being passed to Tesseract. The first line shows results obtained using the original documents. Key figures have been extracted using the best algorithm from Section 4.3, fuzzy string matching with cutoff = 0. Each entry represents the mean value obtained across four key figures; accuracies are shown ± one standard deviation.

5
Conclusion

This chapter will revisit the research questions posed at the start of this thesis and determine whether they have been answered. Section 5.1 presents this discussion, and reiterates which parts of the work have been focused on which research question. Possible continuations have also been suggested. Following this, Section 5.2 discusses how the extracted data can be used to identify instances of financial crime. This section also highlights how the finished model can be used by Bolagsverket in their digitisation efforts. Another masters theis project conducted at Bolagsverket in the spring of 2021 aimed to study how financial crime can be identified using key figures from annual reports; the work carried out in that thesis is a natural continuation of the results obtained here.

5.1 Revisiting the Research Questions

In this section, the original research questions are revisited. For the reader's convenience, the questions are presented again in a slightly more condensed format. This masters thesis aimed to answer the following three questions:

1. Can the OCR results from Tesseract be improved, and if so, how?
2. How can the desired key figures be extracted from the scanned text mass?
3. How can the proposed models be continuously improved?

The following sections will discuss these questions in this order. First, Section 5.1.1 discusses how the OCR results from Tesseract can be improved and suggests ways this work can be continued. This research question turned out to be more explorative, but could benefit from larger quantities of annotated data in order to evaluate different preprocessing techniques. Section 5.1.2 discusses the work that has been carried out in order to extract key figures from the scanned documents. To solve this task a two step approach has been suggested, which first locates relevant pages and then identifies and extracts the desired key figures. It was originally thought that advanced natural language tools would be required to answer this question, but in reality classical methods appear to perform just as well. To achieve better performance of the key figure extraction algorithm, improving the image processing step is the most promising path forward. Lastly, Section 5.1.3, discusses how the methods developed for this thesis can be continuously improved. The models should be updated with annotated training data regularly and compared against the benchmarks established here; if performance starts to decline, a new solution might need to be developed.

5.1.1 Improving the OCR Results

This section discusses the project's first research question: *Can the OCR results from Tesseract be improved, and if so, how?*. Certain documents contain so much noise that Tesseract is not able to identify any text at all, as exemplified in Figure 4.1, so different ways of removing noise have been examined. Section 3.2 presented different image processing techniques and evaluated them qualitatively; results were presented in Section 4.1, showing that the Median, Opening, and RCC methods were promising. Combinations of these three filters also succeeded in removing visible noise from the documents. The research question was revisited in Section 3.6, where the preprocessing techniques could be evaluated quantitatively using the extraction accuracy. The results were presented in Section 4.4, showing that accuracy could be increased from 74% to 84% by applying the RCC filter to 20 challenging documents. It has been seen that OCR quality can be improved, and that RCC is a promising tool for the task.

While the results obtained with the RCC filter are encouraging they ought to be evaluated on a much larger scale, which has not been possible within the scope of this thesis. Though care was taken to select the most challenging documents to try filters on, the data set only consists of 300 documents. In order to determine whether a given filter makes a significant difference, they need to be evaluated using large quantities of annual reports from different time periods. This requires the annotation of key figures from many more annual reports; evaluation should of course also be carried out using the average accuracy for *all* desired key figures.

Since a good image processing step has the potential of making all subsequent tasks easier, this is a very promising area for future research. More powerful image processing techniques, based on deep learning, could improve results significantly. The infrastructure needed to evaluate results using the extraction accuracy is now in place; the only missing piece is a sufficiently large quantity of annotated annual reports. When comparing processing techniques, one should keep in mind that the focus for this research question is on improved reading of *numbers* specifically. A filter which corrupts the text slightly but leaves numbers intact is still useful. In order to fully answer the first research question, three new questions should be addressed:

1. Does the RCC filter improve performance when evaluated on a larger scale? A useful filter should increase the extraction accuracy across all key figures.
2. Can more powerful processing methods, such as deep learning, be used to further improve the accuracy? Improvements can be gained be removing noise, but also by enhancing the text.
3. Is it possible to automatically detect which documents need preprocessing? Focusing on those documents would decrease the risk of corrupting a good quality document, and also save computational power.

Whether the RCC filter generally improves Tesseract's accuracy can only be determined after evaluating on more data. RCC should then serve as a baseline against which other processing methods can be evaluated. In fact, all filters examined in this thesis were tested with a small quantity of data and should be evaluated further. It is possible that their hyperparameters such as kernel size could be chosen better; the parameter values presented in this thesis are intended to serve as a starting point.

5.1.1.1 Preprocessing Using Deep Learning

A new research question has been proposed, suggesting that more powerful processing methods could be used to further improve the accuracy. Various deep learning architectures, such as convolutional neural network, could be studied. A particularly promising idea is to try the convolutional based approach used by Sporici et al. in their paper [27]. The aim of their work was to improve the accuracy of Tesseract 4.0; a summary is presented in Section 1.2. If Bolagsverket desires to achieve closer to 100% accuracy for the extracted key figures, a more thorough study of preprocessing techiques should be performed in another Masters thesis. A thesis entirely focused on preparing binary documents for data extraction is thought to have great potential, and could be used for many kinds of scanned documents. Other government branches could also benefit from the results.

Once a baseline performance has been established using the proposed RCC with a cutoff size of 40 pixels, other techniques can be evaluated against it. Ideally, a deep learning technique will be found to increase extraction accuracy more than the RCC filter does. Different image preprocessing solutions could be proposed for different time periods; if a single solution is presented its performance should be evaluated for different time periods. A known issue with deep learning tools in general is the large quantities of data required to train and evaluate them properly. Before a new Masters thesis is proposed, training data needs to be collected from different time periods, and key figures annotated. It has been suggested to use the digitally submitted annual reports to this end, as they can be converted from XBRL to PDF format, but a solution trained on such documents is not likely to generalise well to real scanned documents. Even if efforts are made to add noise to these pristine PDFs, only scanned documents provide a realistic learning setting.

5.1.1.2 Identifying Noisy Documents

One last question has been proposed on the topic of improving the OCR results: is it possible to automatically detect which documents need preprocessing? One suggested approach, as visualised in Figure 5.1, uses the connected components to evaluate the noise level within a scanned document. Using such histograms of connected component sizes it is possible to use e.g. a cutoff, statistical test, or classifier to determine whether the document requires preprocessing before being fed to Tesseract. Detecting documents in need of image processing could also be done in more subtle ways, such as using an image classifier.

Of course, noise is not the only aspect of a document which makes it harder for Tesseract to correctly extract the text. Some annual reports use a very thin font, or use a grey background behind certain figures, and both these design choices appear to complicate the extraction process. Some fonts only appear to present a problem when it comes to certain digits; for example, 3 and 8 are frequently confused if the font is thicker. If certain fonts of font thicknesses are found to consistently cause extraction errors, a method should be created for detecting them automatically before any image processing takes place. It could then be possible to use different image processing techniques on different kinds of documents.

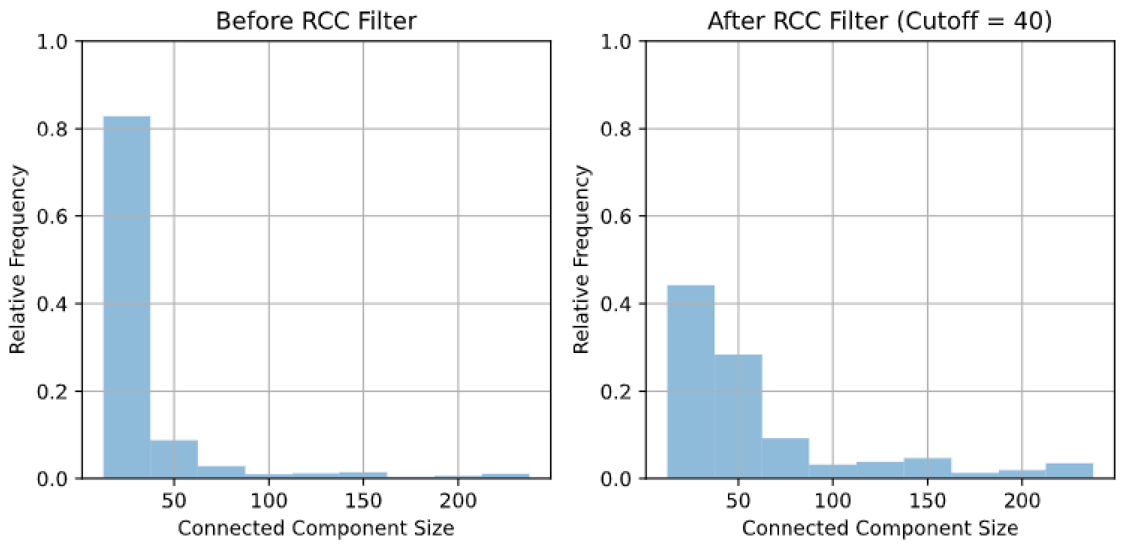


Figure 5.1: Histogram of connected component sizes for a noisy document, before and after the RCC filter is applied. The original document, shows a large fraction of very small components, indicating the prescence of large amounts of noise.

5.1.2 Extracting Key Figures

The second research question, *How can the desired key figures be extracted from the scanned text mass?*, is discussed here. It was initially thought that an advanced machine learning solution would be required for this task, but ultimately a two step approach was found very promising. Splitting the extraction task into two parts, first locating the correct section of the document and then searching it for the desired key figure, increases the interpretability. The page classification step is not very challenging for the documents studied in this thesis. The results in Section 4.2 show that a Random Forest classifier achieves an accuracy of 99.6% on unseen test documents. This classifier, applied to TF-IDF embedded pages, also achieves high F2 scores on both training and test documents. Next, different approaches were used to extract annotated key figures from the balance sheet pages. The fuzzy string search algorithm was found to perform best here, achieving an average accuracy of 89.6% on the test documents in Section 4.3. Based on the results in Section 4.3.1, introducing a similarity score cutoff does not improve the accuracy of the fuzzy matching algorithm.

The relatively simple fuzzy matching algorithm is thought to outperform the more sophisticated embeddings exactly because they are *too* sophisticated for the task. If part of the key phrase is corrupted, as often happens in the OCR step, the embedding similarity algorithms tend to instead choose a similar phrase, where the words are intact. Fuzzy string search, on the other hand, still finds the correct phrase as long as it is not corrupted beyond recognition. It is believed that the Levenshtein distance, or edit distance, would be a more appropriate choice in this setting. This measure corresponds more closely to the kind of errors (insertions, changes, and deletions of individual characters) that Tesseract tends to cause. However, the Levensthein distance is significantly more computationally intense.

Over all, the two step approach suggested here is thought to be a good solution for extracting key figures. A well trained page classifier ensures that the extraction algorithm only has to process one or a couple of pages, decreasing computational load and increasing chances of success. However, some questions remain: for example, is this approach scalable to longer documents? Even if the image preprocessing technique is perfected, a 500 pages long annual report might present new challenges for both the page classification and number extraction algorithms. Tesseract works quite well for extracting the key figures, but is it the optimal tool for the task? In order to address these questions and also move towards an end-to-end key figure extraction algorithm, the following continuations are suggested:

1. Extend the page classifier
2. Classify documents
3. Extract textual information
4. Explore Tesseract alternatives

5.1.2.1 Extending the Page Classifier

While this thesis has served as a proof of concept, showing that it is possible to extract key figures with reasonable accuracy, the end goal is to extract all key figures for each document. To this end, the page classifiers will need to be able to handle more kinds of pages than just *Balance Sheet*, *Income Statement*, and *Other*. For example, some companies submit a cash flow statement which can be analysed. If one truly wants to extract all key figures within the annual report, the page classifiers need to be taught to recognise all kinds of pages that can conceivably appear within an annual report. Whether this approach is scalable depends on the number of distinct page types. Furthermore, the simple classifiers applied here might no longer be suitable for a more complex classification task.

5.1.2.2 Classifying Documents

The documents used in this thesis are quite uniform. Most short annual reports come from smaller companies, who tend to use the same programs for bookkeeping and exporting report. In addition, the data is taken from a relatively short time span. If the data set is expanded with a different kind of annual reports, such as the really long documents submitted by large holding companies, the entire extraction pipeline might fail. If this occurs, a solution could be to introduce a document classifier, turning the extraction algorithm into a three step process. Depending on the number of clusters identified, the classifier could either consist of a static rule checking the length of each document, or a more sophisticated machine learning solution. Subsequent steps should then be adjusted by class: for example, another page classifier might need to be developed for types of documents which have not been studied in this thesis, such as very long documents. While adding significant complexity, separate handling of different document types might prove to be the best path forward.

5.1.2.3 Extracting Textual Information

Another possible continuation is to not only extract key figures from annual reports, but also textual information. However, so far the focus for the image preprocessing methods has been to improve the extraction of numbers from the documents. If the text is to be used, it should be studied whether the proposed image processing techniques are suitable for text as well as numbers. A rudimentary approach for evaluating the success of text extraction is to measure the quantity and quality of extracted words. For a very noisy document, few words will be identified and applying a suitable filter can increase the word count. It is also important to measure the text quality; one could for example count how many of the extracted words appear in the dictionary. When a word is mistaken for another valid word, this is called a *legal error*. More sophisticated forms of OCR post correction could be carried out by e.g. analysing the grammatical structure of each sentence. If the text does not make sense within the given context, a legal error may have occurred.

5.1.2.4 Tesseract Alternatives

This project has exclusively used Tesseract for extracting text from the scanned annual reports. While this OCR engine gives good results, it is not certain to be the optimal tool for the task. The results obtained in this thesis, as well as the suggested continuations, all aim to achieve the best possible results using Tesseract. While the many possible parameter choices for Tesseract have not been studied in depth, it is possible that doing so could improve results. Version 4.1.1. of the OCR engine is used for this thesis, but version 5 will soon be released and should also be evaluated. There are also many entirely different methods for extracting text from images, from other OCR engines to paid data extraction services. Some of these are discussed in Section 1.2.
It has also been suggested to develop a custom model for extracting numbers from the text, using image analysis and a classifier for different characters. The proposed model would locate the name of a certain key figure, e.g. *total assets* within the scanned image, and then directly extract the corresponding number at that height on the page. Giving the number extractor a specific area to extract numbers from might achieve a higher accuracy. Such a solution increases the interpretability of the extraction process, as it would be possible to inspect what regions of the document the model is extracting numbers from. Tesseract does provide this information as well, but it acts more like a black box.
In order to train a combined machine learning solution for key figure detection and extraction, training data would need the key figures to be annotated by value as well as coordinates on the page. The annotation work in itself is therefore quite complicated unless a specific tool is created. The extraction model would need to be able to handle different fonts and font sizes, both for the text and for the numbers. It is believed that an extraordinary amount of work would be required to produce a custom model which can compete with Tesseract in this setting. Even if a custom model is developed for extracting key figures from tables, an OCR engine like Tesseract should still be used if one wishes to extract the text, as this is where its strength lies.

5.1.3 Continuous Model Improvements

How can the proposed models be continuously improved? This is the last research question of the thesis, and will be discussed here. Previous sections have already discussed how the presented solutions for image preprocessing and key figure extraction can be extended in order to improve performance and address a wider selection of extraction tasks. This section instead focuses on how the existing solutions can be managed to make sure they remain useful for a long period of time. The usefulness of the chosen image preprocessing technique, such as the RCC algorithm or a deep learning method, should be continuously monitored.

Once a benchmark has been established, as described in Section 5.1.1, the algorithm should be compared to this benchmark on a yearly basis. If performance starts to decline significantly, different parameter settings should be considered; if this fails, a new image processing solution might need to be developed. Validation requires annotated data. The author recommends selecting a representative sample of 100 documents each year, and then selecting a few key figures to annotate for each document. The usefulness of the preprocessing solution is then evaluated by comparing the extraction accuracy from processed documents and from the original documents.

The key figure extraction algorithm should also be continuously monitored. Specifically, the page classifier is the only active machine learning component in the extraction task, and might perform worse if the input data changes significantly. After extending the models to classify more kinds of pages and evaluating on a wider selection of documents, as described in Section 5.1.2, another classifier might be found to outperform Random Forest. In either case, the chosen classifier should be evaluated against the benchmark on an annual basis.

One suggestion is to use the same 100 documents chosen to evaluate the image processing; if the page types are annotated, they can also be used to evaluate the page classifier. The classifier should be retrained using both old and new data, and then evaluated on a separate set of test documents. If its performance deteriorates significantly relative to the benchmark the model might need to be replaced. As time passes, the structure of documents might change significantly. The model's performance might be more reliable if it is only trained on recent documents, for example the last 10 years.

Initially, the TF-IDF vocabulary used to represent each page was learned from the training documents and then applied to test documents. This approach was abandoned for two reasons: firstly, learning from scanned documents means that potential errors from the OCR process can be introduced into the vocabulary. Secondly, it is not guaranteed that the language used in the training documents is representative. If the vocabulary is learned from a small subset of documents, it can come to include names of people, places or companies, which is not desired. Learning the vocabulary from an example document solves both these problems, but if Bolagsverket updates the example annual report [5], a new vocabulary should be learned from the new document. Once pages have been classified, the algorithm for locating key figures is not expected to need updating with time. Since the suggested solution is based on approximate string matching, a completely deterministic algorithm, no deterioration in performance is expected. However, if a more sophisticated model is introduced it could need to be retrained regularly.

5.2 Using the Results

Last of all, this section discusses how the data extracted from the annual reports can be used by various government agencies. Another thesis at Bolagsverket this spring is focused on using key figures from annual reports to detect financial crime. Different characteristics or patterns can be used to flag annual reports which have a high probability of financial crime. One approach is to determine whether the set of numbers extracted from a document are compliant with Benford's law, which states the distribution of leading numbers. One could also use more sophisticated machine learning techniques such as LSTM to study how a specific company does over time, in search of suspicious patterns. For full details, the reader is referred to the masters thesis conducted by Kristoffer Ahlm at Umeå Universitet (unpublished).
This thesis has had a clear focus on extracting numbers from the annual reports, but as mentioned previously it is possible to also extract textual information. Different kinds of analysis can be carried out on the extracted text than on the numbers. The CFIE (Corporate Finance Information Environment) in the UK [11] aims to analyse the narratives of annual reports using NLP tools. By analysing the language used in e.g. the auditor's statement and the CEO statement, one can attempt to determine whether the company is being honest about its finances. This kind of analysis might only be of interest for larger companies, as their annual reports tend to be longer and contain more text that can be analysed. For sentiment analysis, ideally a context aware embedding such as BERT or SentenceTransformer should be used. The pretrained versions used in this thesis could be fine tuned to the task, and with access to sufficient training data custom embedding could be learned from scratch.

Bibliography

[1] Manuel Aristarán. *Tabula*. GitHub repository. 2012. URL: `https://github.com/tabulapdf/tabula`. (accessed: 2021-02-24).

[2] Florence Babatunde, Bolanle Ojokoh, and Samuel Oluwadare. "Automatic Table Recognition and Extraction from Heterogeneous Documents". In: *Journal of Computer and Communications* 03 (Jan. 2015), pp. 100–110. DOI: `10.4236/jcc.2015.312009`.

[3] JACOB BJÖRKMAN. "Evaluation of the Effects of Different Preprocessing Methods on OCR Results from Images with Varying Quality". Degree Project in Computer Science and Engineering. Stockholm, Sweden: KTH Royal Institute of Technology, 2019.

[4] Bolagsverket. *Contents of an Annual Report*. Dec. 19, 2018. URL: `https://bolagsverket.se/en/bus/business/limited/annual-reports/contents-of-an-annua`.

[5] Bolagsverket. *Exempel på Årsredovisning för Företag som Haft Verksamhet*. Jan. 21, 2019. URL: `https://bolagsverket.se/ff/foretagsformer/aktiebolag/arsredovisning/delar/exempel-pa-arsredovisning-for-foretag-som-haft-verksamhet-1.14475`.

[6] Leo Breiman. "Random Forests". English. In: *Machine Learning* 45.1 (2001), pp. 5–32. ISSN: 0885-6125. DOI: `10.1023/A:1010933404324`.

[7] Mark Clark. *Uncovering the Data Extraction Challenge of Annual Reports*. Feb. 8, 2020. URL: `https://www.infrrd.ai/blog/uncovering-the-data-extraction-challenge-of-annual-reports`.

[8] Jacob Devlin et al. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. 2019. arXiv: `1810.04805 [cs.CL]`.

[9] Murhaf Fares et al. "Word vectors, reuse, and replicability: Towards a community repository of large-text resources". In: *Jörg Tiedemann (ed.), Proceedings of the 21st Nordic Conference on Computational Linguistics, NoDaLiDa*. Linköping University Electronic Press. ISBN 978-91-7685-601-7, May 2017.

[10] Edouard Grave et al. "Learning Word Vectors for 157 Languages". In: *Proceedings of the International Conference on Language Resources and Evaluation (LREC 2018)*. 2018.

[11] Mahmoud El-Haj. *Analysing UK Annual Report Narratives using Text Analysis and Natural Language Processing*. URL: `https://www.lancaster.ac.uk/staff/elhaj/docs/GlasgowTalk.pdf`. (accessed: 2021-05-14).

[12] Hamed Hassanzadeh et al. "A Supervised Approach to Quantifying Sentence Similarity: With Application to Evidence Based Medicine". In: *PLOS ONE*

10.6 (June 2015), pp. 1–25. DOI: 10.1371/journal.pone.0129392. URL: https://doi.org/10.1371/journal.pone.0129392.

[13] J. Hering. "The Annual Report Algorithm: Retrieval of Financial Statements and Extraction of Textual Information". In: 2017.

[14] Samuel Hoffstaetter. *pytesseract 0.3.7.* Dec. 5, 2020. URL: https://pypi.org/project/pytesseract/.

[15] Nano Net Technologies Inc. *Automatic Table Extraction: Capture relevant information from any table structure from any document.* URL: https://nanonets.com/table-extraction. (accessed: 2021-02-24).

[16] Dan Jurafsky and James H. Martin. "Speech and Language Processing – An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition". In: 3rd Edition Draft, 2020-12-30. Chap. 6: Vector Semantics and Embeddings. URL: https://web.stanford.edu/~jurafsky/slp3/.

[17] Dan Jurafsky and James H. Martin. "Speech and Language Processing – An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition". In: 3rd Edition Draft, 2020-12-30. Chap. 5: Logistic Regression. URL: https://web.stanford.edu/~jurafsky/slp3/.

[18] *Keywords.* https://github.com/ekorn/Keywords. Accessed: 2021-02-08.

[19] Akmal Jahan MAC and Roshan Ragel. "Locating Tables in Scanned Documents for Reconstructing and Republishing (ICIAfS14)". In: (Dec. 2014). DOI: 10.1109/ICIAFS.2014.7069552.

[20] Martin Malmsten, Love Börjeson, and Chris Haffenden. *Playing with Words at the National Library of Sweden – Making a Swedish BERT.* 2020. arXiv: 2007.01658 [cs.CL].

[21] Nils Reimers and Iryna Gurevych. "Making Monolingual Sentence Embeddings Multilingual using Knowledge Distillation". In: *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing.* Association for Computational Linguistics, Nov. 2020. URL: https://arxiv.org/abs/2004.09813.

[22] Nils Reimers and Iryna Gurevych. *Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks.* 2019. arXiv: 1908.10084 [cs.CL].

[23] Boaz Shmueli. *Multi-Class Metrics Made Simple, Part I: Precision and Recall.* July 2, 2019. URL: https://towardsdatascience.com/multi-class-metrics-made-simple-part-i-precision-and-recall-9250280bddc2.

[24] Boaz Shmueli. *Multi-Class Metrics Made Simple, Part II: the F1-score.* July 3, 2019. URL: https://towardsdatascience.com/multi-class-metrics-made-simple-part-ii-the-f1-score-ebe8b2c2ca1.

[25] Skikit-Learn. *Neural Network Models (Supervised).* URL: https://scikit-learn.org/stable/modules/neural_networks_supervised.html#multi-layer-perceptron. (accessed: 2021-05-20).

[26] Ray Smith. "An Overview of the Tesseract OCR Engine". In: *Proc. Ninth Int. Conference on Document Analysis and Recognition (ICDAR).* 2007, pp. 629–633.

[27] Dan Sporici, Elena Cușnir, and Costin-Anton Boiangiu. "Improving the Accuracy of Tesseract 4.0 OCR Engine Using Convolution-Based Preprocessing".

In: *Symmetry* 12.5 (2020). ISSN: 2073-8994. DOI: 10.3390/sym12050715. URL: https://www.mdpi.com/2073-8994/12/5/715.

[28] Lichao Sun et al. *Adv-BERT: BERT is not robust on misspellings! Generating nature adversarial samples on BERT*. 2020. arXiv: 2003.04985 [cs.CL].

[29] Richard Szeliski. "Computer Vision: Algorithms and Applications". In: 2nd Edition Draft, 2021-03-27. Chap. 3: Image Processing. URL: http://szeliski.org/Book/.

[30] Ashish Vaswani et al. *Attention Is All You Need*. 2017. arXiv: 1706.03762 [cs.CL].