FACULTY OF INFORMATION TECHNOLOGY CTU IN PRAGUE

Assignment of bachelor's thesis

Title:	Extracting Structured Data from Czech Invoices
Student:	Milan Vu
Supervisor:	Ing. Jiří Novák, Ph.D.
Study program:	Informatics
Branch / specialization:	Knowledge Engineering
Department:	Department of Applied Mathematics
Validity:	until the end of summer semester 2021/2022

Instructions

1. Study the existing approaches for natural language processing (e.g., NameTag2) and optical character recognition (Tesseract OCR, GOCR, CuneiForm, Kraken, etc.).
2. Analyze the current open-source (e.g., invoice2data) and commercial (rossum.ai, docsumo.com, nanonets.com, etc.) tools used for extraction of data from invoices.
3. Implement a tool for the labelling of essential parts of invoices. Use the labelling tool to create training data for a machine learning model.
4. Existing applications commonly require a predefined template for every invoice to extract the data. Research the possibilities of automated processing of new documents without a template if they are similar to already processed data. Design and implement a new solution for invoices in the Czech language.
5. Test the accuracy of recognition, training/prediction speed of the machine-learning model, and the solution's usability.
6. Discuss the dis/advantages of the solution in comparison to the existing tools.

Electronically approved by Ing. Karel Klouda, Ph.D. on 20 October 2020 in Prague.

FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Bachelor's thesis

Extraction of Structured Data from Czech Invoices

Milan Vu

Department of Applied Mathematics
Supervisor: Ing. Jiří Novák, Ph.D.

June 27, 2021

Acknowledgements

I would like to thank my family and friends for support and my thesis supervisor Ing. Jiří Novák, Ph.D. for guidance during writing this thesis.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended. In accordance with Article 46 (6) of the Act, I hereby grant a nonexclusive authorization (license) to utilize this thesis, including any and all computer programs incorporated therein or attached thereto and all corresponding documentation (hereinafter collectively referred to as the "Work"), to any and all persons that wish to utilize the Work. Such persons are entitled to use the Work in any way (including for-profit purposes) that does not detract from its value. This authorization is not limited in terms of time, location and quantity. However, all persons that makes use of the above license shall be obliged to grant a license at least in the same scope as defined above with respect to each and every work that is created (wholly or in part) based on the Work, by modifying the Work, by combining the Work with another work, by including the Work in a collection of works or by adapting the Work (including translation), and at the same time make available the source code of such work at least in a way and scope that are comparable to the way and scope in which the source code of the Work is made available.

In Prague on June 27, 2021 . . . . . .

Czech Technical University in Prague
Faculty of Information Technology

Citation of this thesis

Vu, Milan. *Extraction of Structured Data from Czech Invoices.* Bachelor's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2021.

Abstrakt

Tato práce se zabývá implementací použitelného řešení pro extrakci strukturovaných dat z českých faktur. V první řadě je prostudování veřejně dostupných (open-source) a komerčních řešení této problematiky a také prostudování veřejně dostupných nástrojů a modelů, které jdou potenciálně použít při vytváření vlastního řešení. Implementační část této práce je rozdělena do dvou částí: nejdříve je potřeba vytvořit nástroj pro označení klíčových hodnot na českých fakturách a tímto nástrojem vytvořit trénovací data pro evaluaci a trénování modelů našeho řešení. Druhou částí implementace je návrh a konstrukce modelu, který dokáže extrahovat strukturovaná data z českých faktur.

Klíčová slova OCR, Strojové učení, Automatizace faktur, Extrakce dat, Extrakce klíčových informací, Rozpoznání textu, Detekce Textu

Abstract

The goal of this thesis is to implement a usable solution for extraction of structured data from Czech invoices. The first part of this thesis is to study open-source and commercial solutions addressing this problem and study publicly available tools and models that could potentially be used for said solution.

The implementation part of this thesis is split in two parts: firstly, implementation of a tool for labeling Czech invoices and using this tool to create a training data set for evaluation and training of machine-learning models. Secondly, design and implementation of a solution that can extract structured data from Czech invoices.

Keywords OCR, Machine Learning, Invoice Automation, Data Extraction, Invoice Capture, Key Information Extraction, Text Recognition, Text Detection

Contents

1 Introduction **1**
1.1 Motivation and objectives 1

2 Analysis **3**
2.1 Commercial Invoice Data Capture solutions 3
2.1.1 Testing commercial Invoice Data Capture solutions . . . 3
2.1.2 Image Pre-processing 5
2.1.2.1 Image representation 5
2.1.2.2 Noise reduction 6
2.1.2.3 Morphological operations 6
2.1.2.4 Threshholding 7
2.1.2.5 Image resizing 7
2.2 Text Detection 7
2.2.1 Tools and models for Text Detection 8
2.3 OCR / Text Recognition 9
2.3.1 OCR output formats 9
2.3.2 Technologies used in OCR engines 10
2.3.3 OCR Engines 11
2.3.4 Commercial OCR engines 13
2.4 Structured Data Extraction 13
2.4.1 Supervised machine learning classification models 15
2.4.1.1 Ensemble models 17
2.4.2 Exploratory analysis 18
2.4.3 Natural Language Processing 19
2.4.3.1 Tools for NER and Lemmatisation 20
2.4.4 Metrics for evaluation and comparison 20
2.5 Labeling tool technologies 22

3 Design **23**

3.1 Functional Requirements of the Labeling tool 23
3.2 Non-functional Requirements of the Labeling tool 24
3.3 Implementation steps 24

4 Labeling tool implementation 27
4.1 Front-end 27
4.1.1 Home page/Invoices page 27
4.1.2 Upload page 29
4.1.3 Editing page 29
4.2 Back-end 30
4.2.1 Endpoints 30
4.2.1.1 Simple Endpoints 31
4.2.1.2 Complex Endpoints 32

5 Invoice Data Capture model implementation 33
5.1 Feature engineering 33
5.1.1 Applying Natural Language Processing 34
5.2 Building data extraction model 34

6 Evaluation and Comparison 35
6.1 Text Detection and Image Pre-processing 35
6.1.1 Tesseract Text Detection method 35
6.1.2 CRAFT 36
6.1.3 EAST 37
6.1.4 Best Text Detection 37
6.2 Comparing Machine Learning Models 37
6.2.1 Hyperparameter tuning 38
6.2.2 Most important features 39

7 Conclusion 41

Bibliography 43

A List of used abbreviations 49

B Contents of the enclosed media 51

List of Figures

2.1 hypatos.ai demo 4
2.2 rossum.ai demo 5
2.3 Architecture of a traditional RNN 11

3.1 Design diagram of the application 25

4.1 Front-end homepage 28
4.2 Front-end editing page 29
4.3 Database schema 31

6.1 Random Forest Classifier Hyperparameter tuning 39
6.2 Gradient Boosting Classifier Hyperparameter tuning 39

List of Tables

6.1 Tesseract OCR accuracy by image pre-processing techniques with Tesseract's native Text Detection 36
6.2 Tesseract OCR accuracy by image pre-processing techniques with CRAFT Text Detection 36
6.3 Tesseract OCR accuracy by image pre-processing techniques with EAST Text Detection 37
6.4 Machine-learning classification models comparison by accuracy . . 38
6.5 Feature importances from Random Forest Classifier 40

Chapter 1

Introduction

The field of Artificial Intelligence (AI) and machine-learning (ML) has seen great success in recent years. New research and state-of-the-art models keep being published at a rapid pace. Thankfully, a large number of these works are being implemented into publicly available tools, especially for the fields of Natural Language Processing (NLP) and Computer Vision, as seen on paperswithcode.com[1], for example. With such resources, it is not surprising that we have seen a surge of new companies built around addressing the extraction of structured data from invoices (further referred to as Invoice Data Capture), such as rossum.ai, docsumo.com, nanonets.com, hypatos.ai and many more, in the year of 2021.

Not to say that solutions for Invoice Data Capture were not here before, but existing open-source and traditional applications commonly require a predefined template for every invoice to be able to extract structured data. Such solutions have been here quite some time, but were not widely adapted due to the lack of usability. New commercial ML based solutions do not require any template because they are able to learn from invoices processed beforehand. With enough learning, the models could, in theory, extract any kind of an invoice, opening the possibility of automating the invoice registration process.

1.1 Motivation and objectives

The cost of processing invoices is quite high for certain companies. An article[2] by rossum.ai estimate template based commercial solutions at $1.03 per invoice and $38,333 annually for the implementation, while estimating their ML based solution at $0.45 per invoice with the implementation annual cost at $45,833. The cost per invoice might not seem high, but from the yearly costs of the implementations, it is apparent that their main clientele are large corporations, since smaller companies could never afford such prices, even if they process a much smaller number of invoices.

The goal of the thesis is therefore to study the possibility, design and implementation of a usable and freely accessible solution for template-less and ML based Invoice Data Capture for Czech invoices, which even small companies can afford to maintain.

It is evident that we need a good training data set of labeled invoices in order to properly evaluate and train ML models. Unfortunately, we could not find a good source of labeled Czech invoices, nor a good tool for labeling them, therefore a crucial part of this work is the creation of our own labeling tool and training data.

Chapter 2

Analysis

In this chapter, we firstly analyze commercial solutions for the task of Invoice Data Capture. Next, we split the main task into three sub-tasks, namely: Text Detection, OCR / Text Recognition and Structured Data Extraction, which is needed in order to find the right technologies and tools for our solution. Lastly, we analyze technologies that can be used for the labeling tool mentioned in section 1.1.

2.1 Commercial Invoice Data Capture solutions

As of 2021, the number of companies offering Invoice Data Capture solutions is still growing. The website aimultiple.com[3] showcases comparisons between 21 most popular ones, of which, we picked 3 to talk about:

- rossum.ai
- docparser.com
- hypatos.ai

While rossum.ai was listed because it is a successful startup from the Czech Republic, docparser.com and hypatos.ai were listed because of an impressive record of customers from the Big 4 and other Fortune 500 companies. Because these companies offer ML based solution, the performance of their products get better over time and therefore are hard, or rather unfair to compare. There are however, some studies[4] cross-comparing products in their pre-trained form.

2.1.1 Testing commercial Invoice Data Capture solutions

Some of the commercial solutions now offer a free trial of their products. Due to this fact, we registered for a free trial with hypatos.ai and rossum.ai. While

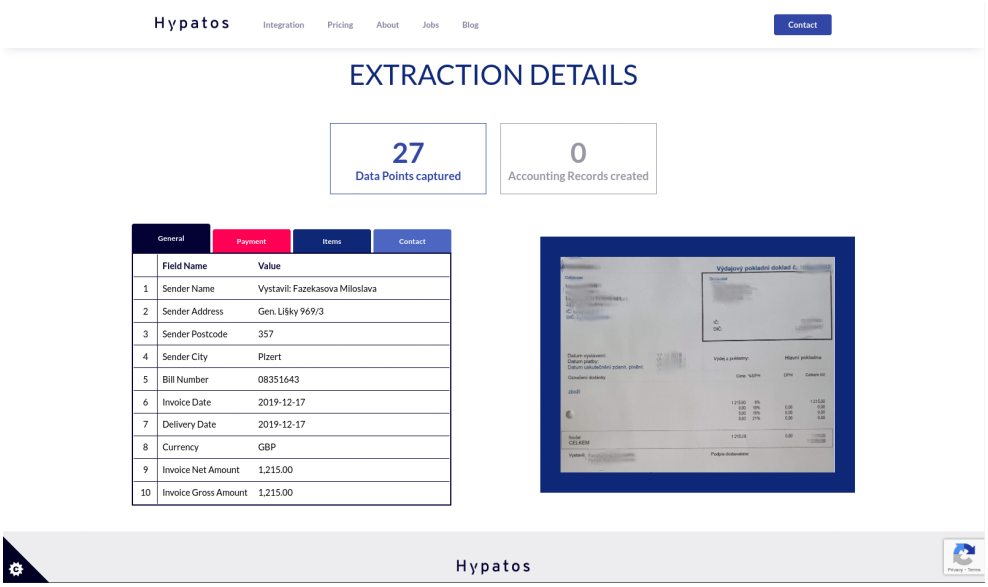

Figure 2.1: hypatos.ai demo

hypatos.ai offers a basic demo where we can upload only a single invoice at a time with the need of a captcha check between each invoice, rossum.ai enables us to upload a bigger number of invoices.

hypatos.ai
The solution offered by hypatos.ai (2.1) is able to identify a variety of values on an invoice, such as City, Address, Postcode, E-mail, Phone number and some more. Among all the extracted information, hypatos.ai also attempts to extract all item values on an invoice, which help predict the Gross Amount, however it seems to struggle with Gross Amounts where the value is 0 (in case of invoices that confirm payment). The solution also struggles with extracting Invoice ID and ICO (the solution does not attempt to identify ICO, but VAT number, but since both values can be used to track down a specific company in the Czech Republic, they are interchangeable and therefore deemed the same in this work). Due Date is extracted perfectly on every invoice tried as expected.

rossum.ai
The first impression of rossum.ai's solution is the impressive user interface (2.2). Although this solution does not offer extraction of Geo-location and Contact information like hypatos.ai, it does a better job at identifying other key values. Due Date, ICO (or VAT) and Invoice ID is extracted perfectly in most cases. Rossum.ai also extract all item values on an invoice, but does a

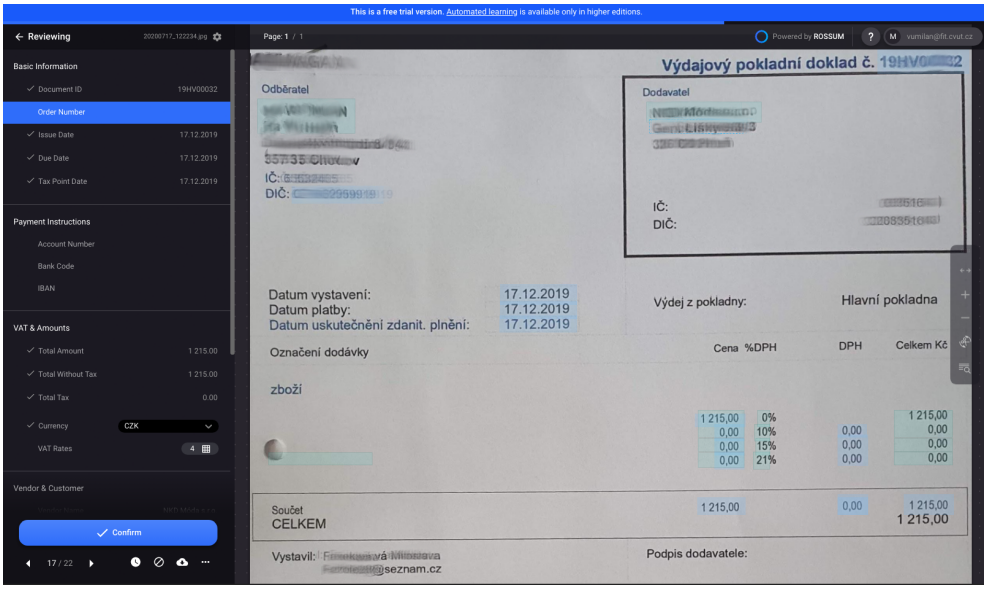

Figure 2.2: rossum.ai demo

significantly better job, which also results in a better accuracy for the Gross Amount. Just like hypatos.ai however, it struggles with Gross Amounts with the value 0.

Both commercial solutions for Invoice Data Capture were impressive, but rossum.ai's solution was better overall.

2.1.2 Image Pre-processing

Before applying the Text Detection and Recognition models, it is usually important to somehow pre-process the images we want to feed to those tools. The effects of applying image pre-processing can vary, which we inspect in the Experimentation chapter.

2.1.2.1 Image representation

RGB

RGB is a color model in which red, green, and blue are added together in various ways to produce other colors. In computer graphics, this color model come in the form of 3 channels (red, green, blue) where each channel contains 8bit pixels. The final color range we can get with this design is therefore 256^3. Nowdays, RGB is the most popular method of representing the color spectrum.

Grayscale image
A grayscale image, unlike RGB, has only one color channel. Each channel consists of 8bit pixel, can hold values from 0 to 255. The values of the pixel in this case represent brightness, where the value 0 is pitch black and value 255 is bright white. One way of converting RGB into grayscale is through weighted means of the color channels, such as

$$G(x, y) = 0.3 \cdot R + 0.59 \cdot G + 0.11 \cdot B.$$

Binary image
A binary image is an even simpler image representation than grayscale, where each pixel can only hold the value of 0 or 1. RGB images are usually converted to a binary image by converting to grayscale first, and then applying one of the threshhold methods described further on.

2.1.2.2 Noise reduction

Image noise reduction is a process that attempts to remove noise from an image, which usually occurs in low quality photos. The difficulty of this task is to remove noise, but keep all details of the image, which is difficult to distinguish.

Median filter
The idea of median filter[5] is to run through each pixel of an image and replace the value of the pixel with the mean of all neighbouring values. The number of neighbouring pixels is determined by a square "window" with a length that can be set, but has to be an odd number. Should an edge be inside the window, instead of mean, the most dominating value is chosen for the center pixel.

2.1.2.3 Morphological operations

Morphological operations[6] is a collection of non-linear operations related to the shape or morphology of features in an image. Morphological operations were firstly defined for binary images, later extended to grayscale images and now support even color images. The operations work with two images - the input image I and image S, which is used as a kernel, called the structural element.

Dilation and Erosion
Dilation and Erosion are two fundamental operations of Morphological image processing. In essence, dilation adds pixels around objects in an image, while erosion removes pixels around objects. The number of added or removed pixels depend on the size and shape of the structuring element S.

2.1.2.4 Threshholding

Thresholding is a method of image segmentation and can be used to convert grayscale images to binary images. The conversion starts by choosing a threshhold value τ. Obtaining a binary image happens when for each pixel p in a grayscale image, which has a value from 0 to 255, we set the value of p to 0 if $p < \tau$ or to 1 when $p \geq$.

Otsu's method
Real world images are often very different, which makes choosing a universal threshhold very difficult. The Otsu's method[7] approaches the problem differently, where for each image, we calculate an optimal binarization threshhold separably.

2.1.2.5 Image resizing

Image resizing is a method to convert an image from one pixel grid to another. It is essential when we need to increase or decrease the total number of pixels (change the size of an image). Often referred to as interpolation, it works by using known data to estimate values at unknown points.

Bicubic interpolation
This work uses the bicubic interpolation, where it considers a cube of 4x4 pixels. After converting an image to a bigger (or smaller) grid of pixels, the missing pixels around the original ones are calculated using the neighbouring pixels in the 4x4 cube, where apart from the value of the pixels, we also take the distance between pixels in the cube into consideration.

2.2 Text Detection

Text detection is a Computer Vision task that attempts to mark the regions of text on a given image by surrounding the regions with rectangular bounding boxes. The text in images vary in font, color, shape, position or even orientation. Additionally, when it comes to images captured by a camera, we also need to take quality, size and shadows into consideration, which make Text Detection a challenging problem.

To improve Text Detection, the International Conference on Document Analysis and Recognition (ICDAR[8]) holds a competition once every two years and release data sets used in those competitions, which newer works use to benchmark against older models. Although not the latest, the most relevant data sets for Text Detection released by ICDAR are ICDAR2013, ICDAR2015 and ICDAR2017.

Although most of the OCR tools mentioned in the next section already have a Text Detection method, the methods are quite outdated and do not

focus on images captured by a camera (which becomes quite apparent in the Experimentation chapter). For the stated reasons, we introduce EAST[9] and CRAFT[10], which are newer Text Detection models that focus on photos.

2.2.1 Tools and models for Text Detection

Python
Python programming language is one of the most popular programming languages. It is mainly being used in Data-Science, ML and AI research, therefore it is the perfect match for this work. It also has a rich community with many third-party libraries and frameworks that this work utilizes, such as FastAPI for the back-end of the labeling tool, packaged Text Detection models, OCR tools and ML models.

CRAFT
Character Region Awareness for Text Detection or CRAFT[10] is a neural network based model for Scene Text Detection published by Youngmin Baek, Bado Lee, Dongyoon Han, Sangdoo Yun and Hwalsuk Lee in 2019. Although the proper use-case for this model are scene images and not scanned documents, the addition of this model to this work provided great improvements, as described later in the Experimentation chapter.

The CRAFT model is ranked 16th on ICDAR2013, 15th on ICDAR2015 and 8th on ICDAR2017 as seen on paperswithcode.com[1]. Although the model does not rank best on the ICDAR data sets, it is used in this work due to the fact that this model also exists in the form of a Python package called craft-text-detector[11].

OpenCV
OpenCV is a huge open-source library for computer vision tasks, published under the Apache 2 Licence. As a very popular library, it contains all the necessary and up-to-date tools for image-processing that this work needs. OpenCV is supporting Python as a pre-build package, which is called opencv-python[12].

EAST
Efficient and Accurate Scene Text Detector or EAST[13] is a neural network based model for Scene Text Detection published by Youngmin Baek, Bado Lee, Dongyoon Han, Sangdoo Yun, Hwalsuk Lee in 2017. EAST is therefore a predecessor of the CRAFT model and was considered in this work due to the fact that it has been integrated into the OpenCV library for Python. One of EAST model's advantages is also the ability to detect text in real-time through camera.

As the CRAFT model's predecessor, it is obvious it ranked worse on the ICDAR data sets. Nevertheless, just like CRAFT, this model was introduced due to being integrated into a tool and therefore easy to access.

2.3 OCR / Text Recognition

Optical Character Recognition (OCR[14]) is, as the name implies, a task to recognize characters or text in a given image. Some of the most popular OCR engines[15] are OCRopus, Kraken, Calamari and Tesseract.

2.3.1 OCR output formats

Standard output formats of OCR engines are usually hOCR[16] and ALTO[17].

ALTO
Analyzed Layout and Text Object (ALTO) is an open XML Schema standard developed by METAe project with the focus on describing the layout and content of physical text resources, such as pages of a book or a newspaper. The standard is hosted by the Library of Congress since 2010. The structure of the ALTO file is as follows:

```
<alto>
    <Description>
        <MeasurementUnit/>
        <sourceImageInformation/>
        <Processing/>
    </Description>
    <Styles>
        <TextStyle/>
        <ParagraphStyle/>
    </Styles>
    <Layout>
        <Page>
            <TopMargin/>
            <LeftMargin/>
            <RightMargin/>
            <BottomMargin/>
            <PrintSpace/>
        </Page>
    </Layout>
</alto>
```

hOCR
hOCR is an HTML formatting standard of data representation for data obtained from OCR. The hOCR format is much more popular than ALTO and is supported across many OCR engines. Bellow is an example output of an OCR engine in hOCR format:

```
...
<p class='ocr_par' lang='cz' title="bbox930">
    <span class='ocr_line' title="bbox 348 797 1482 838;
    ↪  baseline -0.009 -6">
        <span class='ocrx_word' title='bbox 348 805 402 832;
        ↪  x_wconf 93'>Faktura</span>
        <span class='ocrx_word' title='bbox 421 804 697 832;
        ↪  x_wconf 90'>1234</span>
        <span class='ocrx_word' title='bbox 717 803 755 831;
        ↪  x_wconf 96'>Praha</span>
        <span class='ocrx_word' title='bbox 773 803 802 831;
        ↪  x_wconf 96'>4</span>
        ...
    </span>
    ...
```

2.3.2 Technologies used in OCR engines

PyTorch
PyTorch[18] is an open source machine learning library used for mainly in the fields of Computer Vision and NLP. PyTorch is mainly developed by Facebook's AI Research lab as a free and open-source software, released under the Modified BSD license and supports Python, CUDA, and C++ programming languages.

TensorFlow
TensorFlow[19] is an open-source software library for machine learning. It can be used across a range of tasks but has a particular focus on deep learning. TensorFlow is being developed by the Google Brain Team and also supports Python, C++, CUDA and JavaScript.

RNN
Recurrent neural networks (RNN[20]) are a class of neural networks[21] that process sequential or time series data. The word "recurrent" is used due to it's dynamic behavior, since the architecture (2.3) allows previous outputs to be used as inputs between the layers of the network.

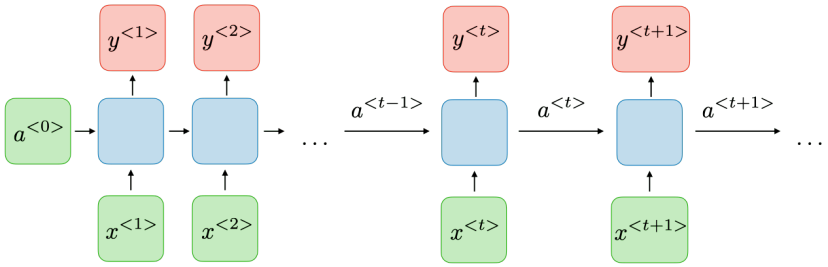

Figure 2.3: Architecture of a traditional RNN

Formally, for each timestep t, the activation $a^{<t>}$ and the output $y^{<t>}$ are expressed as follows:

$$a^{<t>} = g_1(W_{aa}a^{<t-1>} + W_{ax}x^{<t>} + b_a)$$

and

$$y^{<t>} = g_2(W_{ya}a^{<t>} + b_y)$$

where $W_{ax}, W_{aa}, W_{ya}, b_a, b_y$ are coefficients that are shared temporally and g_1, g_2 are activation functions.

LSTM
Long Short-Term Memory (LSTM[22]) networks fall under RNN, but have a slightly different architecture. The idea behind the LSTM architecture is, as the name could suggest, to utilize long-term memory, which is created from processed short-term memory, where short-term memory refers to how RNN works. Simply put, LSTM networks can remember context longer, making them suitable for NLP, Speech Recognition and time series classification and prediction tasks.

2.3.3 OCR Engines

OCRopus
OCRopus is a collection of document analysis programs, not a complete solution for OCR. The big disadvantage is that it requires Python 2.7 and older versions of other Python packages, which is mainly due to the fact that one of the main developers of OCRopus was hired by NVidia to rebuild the tool with deep learning capabilities. The new version of OCRopus was released by NVidia under the name OCRopus3, but is now being replaced by OCRopus4, which is a rewrite of OCRopus3 using PyTorch 1.7.

Kraken
Kraken is a complete OCR system just like Tesseract, which was created by forking OCRopus and finalizing it into a turnkey system. Performance-wise, it is on par with OCRopus, but apart from hOCR ouput, it also supports ALTO output format.

Calamari
Calamari is a relatively new OCR engine based on OCRopus and Kraken using Python3 with TensorFlow. It would seem that Calamari is another iteration of Kraken, with the addition of deep learning capabilities, designed with ease of use and customizability in mind. Unfortunately, Calamari does not have many pre-trained models that could be used for this work.

Tesseract
Tesseract (Tesseract OCR[23]) is an open-sourced OCR engine with a long and rich history. Development for Tesseract started already in the 1980s by HP. The engine was then open-sourced in 2005 and Google started backing this project starting 2006.

Today, Tesseract is arguably the most popular open-source OCR engine, with continuous updates, such as the addition of LSTM to the engine, new comprehensive and up to date documentation and an active community, directly supported by Google. Thought the biggest advantage is the amount of available pre-trained models for various languages, especially Czech, which is the requirement for this thesis. Tesseract is written in C/C++ and supports plain text, hOCR and PDF output formats.

As a universal purpose OCR engine, Tesseract has many modes in which it can operate. The following list are all the page segmentation modes offered by Tesseract, which was taken from it's manpage.

```
 0    Orientation and script detection (OSD) only.
 1    Automatic page segmentation with OSD.
 2    Automatic page segmentation, but no OSD, or OCR.
 3    Fully automatic page segmentation, but no OSD. (Default)
 4    Assume a single column of text of variable sizes.
 5    Assume a single uniform block of vertically aligned
 ↪  text.
 6    Assume a single uniform block of text.
 7    Treat the image as a single text line.
 8    Treat the image as a single word.
 9    Treat the image as a single word in a circle.
10    Treat the image as a single character.
11    Sparse text. Find as much text as possible in no
 ↪  particular order.
12    Sparse text with OSD.
```

```
13    Raw line. Treat the image as a single text line,
      bypassing hacks that are Tesseract-specific.
```

This work uses Tesseract along with a Python package[24] that wraps Tesseract. The mainly used page segmentation modes are mode 12, which conducts both Text Detection and Recognition, and mode 7, which only attempts Text Recognition.

2.3.4 Commercial OCR engines

Commercial solutions for OCR, such as Google Cloud Vision, Microsoft Azure Computer Vision and Amazon's Rekognition API come in the form of an API service, not a downloadable application. Although the performance of these engines are slightly better, relying on commercially available resources would defeat this work's purpose.

2.4 Structured Data Extraction

In this work, Structured Data Extraction is a task that ingests the unstructured output from the previous two tasks and attempts to transform them into a structured format. Specifically, the inputs for this tasks are the texts from an OCR engine and their locations or rather, bounding-boxes, from the Text Detection model. The following snippet is an example input (note that the input is in the JSON[25] format as it was converted from the hOCR format):

```
[
    {
        "bounding_box": {"x": 416.5892974853, "y":
        ↪  44.422203327, "width": 41.9310344828, "height":
        ↪  9.9310344828},
        "text": "Faktura",
    },
    {
        "bounding_box": {"x": 461.8306767957, "y":
        ↪  49.9394447064, "width": 4.4137931034, "height":
        ↪  1.1034482759},
        "text": "-",
    },
    {
        "bounding_box": {"x": 470.6582630026, "y":
        ↪  44.422203327, "width": 43.0344827586, "height":
        ↪  13.2413793103},
        "text": "daňový",
```

```
    },
    ...
]
```

The output of this task are identified and extracted values, which are the most important for invoice registration, namely: Invoice ID (ID), Due Date (Datum k uhrade), Amount (Celkem k uhrade) and Person Identification Number (ICO). An example output we would like to get looks as follows:

```
[
    {
        "type": "Datum splatnosti",
        "bounding_box": {"height": 18.2731628418, "width":
        ↪  62.0893554688, "x": 517.4844970704, "y":
        ↪  200.1935195923},
        "text": "05.04.2018",
    },
    {
        "type": "Invoice ID",
        "bounding_box": {"height": 22, "width": 60, "x":
        ↪  456.3251953125, "y": 64.1928710938},
        "text": "2018134",
    },
    ...
]
```

As we can see, the task of Structured Data Extraction is in this case to correctly classify the bounding boxes and their texts, which we obtained from the Text Detection and Recognition tasks.

Traditionally, solving this problem would require a template as seen in the project invoice2data[26], where the template comes in the form of a YAML document and contains regular expressions[27] to successfully match the important data. One example of a template for the invoice2data project looks as follows:

```
issuer: Amazon Web Services, Inc.
keywords:
- Amazon Web Services
exclude_keywords:
- San Jose
fields:
  amount: TOTAL AMOUNT DUE ON.*\$(\d+\.\d+)
  amount_untaxed: TOTAL AMOUNT DUE ON.*\$(\d+\.\d+)
```

```
  date: Invoice Date:\s+([a-zA-Z]+ \d+ , \d+)
  invoice_number: Invoice Number:\s+(\d+)
  partner_name: (Amazon Web Services, Inc\.)
options:
  remove_whitespace: false
  currency: HKD
  date_formats:
    - '%d/%m/%Y'
lines:
    start: Detail
    end: \* May include estimated US sales tax
    first_line: ^(?P<description>\w+.*)
    line: (.*)\$(\d+\.\d+)
    last_line: VAT \*\*
```

Template based solutions are not practical and become quite expensive in the long run as mentioned before (1.1). This work aims to implement a template-less solution with the utilization of ML models similar to the commercial applications introduced in the section 2.1. From our example input and output, we determined that we essentially deal with a classification problem and since we plan to train our model on labeled data, the more formal definition for this task is supervised machine learning classification[28].

2.4.1 Supervised machine learning classification models

There is a big selection of supervised learning classification models to choose from. In this work, we focus on the most popular ones: Logistic Regression, K-Nearest Neighbours, Support Vector Machine, Naive Bayes, Random Forest and Gradient Boosting classifiers. Most of these models can perform both classification and regression, however in this work, we focus on the classification form of these models. All of the mentioned ML classification models are implemented in Scikit-learn[29], which is a Python ML library.

Since we are dealing with supervised learning, let us assume we have a training data set of predictor (or independent) variables $\boldsymbol{X} \in \mathbb{R}^{N,p}$ (information we use to predict) with the known predicted (dependent) variables $\boldsymbol{Y} \in \mathbb{R}^{N}$ (depended variables are the labeled data).

Logistic Regression

Logistic regression[30] is a statistical model which is similar to linear regression. It is used when the dependent variable is not a number, but a category (e.g., a ”yes/no” in case of binary classification). It could be intercepted as a linear regression for classification (which is the reason why it is called ”regression” although it performs classification). Although Logistic regression can

be adjusted to predict more than two categories, for simplicity reasons, we describe the Binary Logistic regression:

Firstly, the model performs linear regression, which is essentially a method that attempts to find a linear combination of predictor variables $\boldsymbol{X}$ that can properly estimate the predicted variable $\boldsymbol{Y}$

$$\boldsymbol{Y} \approx \boldsymbol{w}_0 + \boldsymbol{w}_1\boldsymbol{x}_1 + ... + \boldsymbol{w}_p\boldsymbol{x}_p,$$

where $\boldsymbol{x}_i$ are exact values of predictor $\boldsymbol{X}_i$ and $\boldsymbol{w}_i$ are unknown coefficients.

Next, a logistic function is applied to obtain the predicted classifications. The function needs to transform the value of $\boldsymbol{w}_0 + \boldsymbol{w}_1\boldsymbol{x}_1 + ... + \boldsymbol{w}_p\boldsymbol{x}_p$ to stay in the range of $[0, 1]$. The usual choice for the logistic function is the sigmoid function:

$$f(x) = \frac{e^x}{1 + e^x} = \frac{1}{1 + e^{-x}}$$

where the value of $f(x)$ is the probability of $Y = 1$ and $1 - f(x)$ for $Y = 0$.

K-Nearest Neighbours

The k-nearest neighbors algorithm (K-NN[31]) is a non-parametric classification method first developed by Evelyn Fix and Joseph Hodges in 1951. The simple idea behind the K-NN classifier is for a predictor not in the training data set $\boldsymbol{x_0} \notin \boldsymbol{X}$, find k closest neighbours to $\boldsymbol{x_0}$ where $\boldsymbol{x}_1, ..., \boldsymbol{x}_k \in \boldsymbol{X}$. Next, combine the y values of the k neighbours and output the y value that is most common among them. To find the nearest neighbours, a distance metric has to be defined for the predictors $\boldsymbol{x} \in \boldsymbol{X}$. The most common metric is the Euclidean distance, also called L_2 distance:

$$||\boldsymbol{x} - \boldsymbol{y}||_2 = d_2(\boldsymbol{x}, \boldsymbol{y}) = \sqrt{\sum_{i=0}^{p-1}(x_i - y_i)^2},$$

for two points $\boldsymbol{x} = (x_0, x_1, ..., x_{p-1})$ and $\boldsymbol{y} = (y_0, y_1, ..., y_{p-1})$ where $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^p$.

Support Vector Machine

A support-vector machine (SVM) attempts the classification problem by separating data points using hyperplanes in a high- or infinite-dimensional space. The training of the SVM model is to find the hyperplane with the largest distance from the training data of any class.

When the data is not linearly separable (cannot separate using a hyperplane), the SVM model performs something called as a "Kernel Trick". It is essentially a method that maps the training data on a higher dimension, where a separation is possible.

Naive Bayes
The naive Bayes classifier[32] is based on Baye's theorem and in it's core, relies on the assumption that all features are independent (unrelated to each other). The assumption is often wrong, hence the name "naive", but despite that, it is one of the best classification methods.

The prediction problem can be written in a probabilistic form:

$$\hat{Y} = \arg\max_{y \in \boldsymbol{Y}} P(Y = y | \boldsymbol{X} = \boldsymbol{x}),$$

Naive Bayes model however, approaches the problem differently, by assuming the possible estimation of $P(\boldsymbol{X} = \boldsymbol{x}|Y = y)$. Using the Baye's Theorem, we get

$$P(Y = y|X = x) = \frac{P(X = x|Y = y)P(Y = y)}{P(X = x)}.$$

After removing $P(\boldsymbol{X} = \boldsymbol{x})$ since we are interested only in the maximum value of the predicted value $\hat{Y}$, we get the prediction formula

$$\hat{Y} = \arg\max_{y \in \boldsymbol{Y}} P(\boldsymbol{X} = \boldsymbol{x}|Y = y)P(Y = y).$$

Finally, to estimate $P(\boldsymbol{X} = \boldsymbol{x}|Y = y)$, we utilize the "naive" assumption, which can be described as

$$P(\boldsymbol{X} = \boldsymbol{x}|Y = y) = P(X_1 = x_1|Y = y) \cdot ... \cdot P(X_n = x_n|Y = y)$$

which gives us the final form of the Naive Bayes classifier

$$\hat{Y} = \arg\max_{y \in \boldsymbol{Y}} \prod_{i=1}^{p} P(X_i = x_i|Y = y)P(Y = y).$$

2.4.1.1 Ensemble models

In statistics and machine learning, ensemble methods use numerous learning algorithms together to obtain better predictive model than could be obtained from any of the previous learning algorithms alone.

Random Forest
Random forests or random decision forests classifiers[33] are an ensemble method, which typically utilizes decision trees[34], thus the name "forest". Decision trees is a simple learning algorithm that works on a basis of creating a tree of decisions based on the training data. The decisions are constructed using entropy, which is the measure of impurity

$$Entropy(D) = -\sum_{i=0}^{k-1} p_i \log p_i$$

where D is usually the depended variable. For each new decision in the Decision tree, we want to choose such a feature $X_i \in \boldsymbol{X}$ that maximally reduces the entropy, by calculating the information gain

$$IG(D, X_i) = Entropy(D) - t_0 Entropy(D_0) - t_1 Entropy(D_1)$$

where D_0 is a subsample of D for which $X_i = 0$, D_1 is a subsample of D for which $X_i = 1$ and t_i is the proportion of D's elements in D_i, ie. $t_i = \frac{\#D_i}{\#D}$.

The random forest classifier is then constructed by creating n random subsets of the training data and training n decision trees on those subsets. The classification is then decided by the majority vote of all decision trees.

Gradient Boosting
Gradient boosting classifiers work on the same principle as Random Forest, utilizing weaker prediction algorithms together to construct a better predictive model. Unlike Random Forest however, the Decision Trees (or other predictive algorithm) are not constructed in parallel, but in sequence, meaning every Decision Tree is influenced by the previous one (except for the first one). The influence is the previous models come in the form of "hidden" instance weights, where it stores the information of which instances were classified incorrectly so the next model in line can try harder to do a better job.

2.4.2 Exploratory analysis

To successfully classify the tracked values, we examine the characteristics of each value on invoice images in our data set to find features (predictors) that could help in the identification of said values, which are: ID, ICO, Due Date and Amount.

Format
ID is usually a combination of numbers and letters in upper case. ICOs standard format is a set of numbers with exact width. Due Date format is a date, here we have to be careful about being able to detect various date formats (czech date format, english date format, full year vs last two digits of year, etc). Amount format is usually a floating point number, but sometimes just an integer as well. All of the important values are also often in bold, or bigger font sizes.

Position
Other semantic information comes from the position of the data points. While ID is on the top, ICO and Due Date are usually in the middle part of the document and Amount is often at the bottom.

Text
Additional information that could help identify key values in the document are the texts around them. For example, ICO almost always has "ICO" or "IC" in front of the number. Amount usually has "Celkem" or fully "Celkem k uhrade" on the left of the number, with currency (Kc) on the right. Due Date also almost always has "Datum" or "Datum splatnosti" on the left of the value. More information can also be obtained by applying natural language processing on text data.

2.4.3 Natural Language Processing

Natural Language Processing (NLP) combines modeling of human language with statistical, machine learning, and deep learning models. Together, these technologies enable computers to process human language in the form of text or voice data in order to "understand" the meaning.

This work deals with text on invoice images, therefore the NLP methods used are Lemmatisation and Named Entity Recognition.

Lemmatisation
In many languages, words appear in several inflected forms. For example, in Czech, the verb 'bežet' may appear as 'bež', 'běhal', 'běží', 'běhali' and many more. The base form, 'běžet', that can be looked up in a dictionary, is called the lemma for the word.

Lemmatisation[35] is similar to stemming. The difference is that a stemmer works on a single word without knowledge of the context, and therefore cannot discern between words which have different meanings depending on the context. Stemmers are typically easier to implement and run faster, making them appropriate for applications that do not mind the reduction in accuracy.

For example: The word "běžet" is the base form for the word "běží", and hence this is matched in both stemming and lemmatisation, however the word "topit" can be either "to burn" or "to drown" depending on the context; e.g., "topit v kamnech uhlím" vs. "topit se ve vodě". Unlike stemming, lemmatisation attempts to select the correct lemma depending on the context.

Named Entity Recognition (NER)
Named entity recognition (NER) — sometimes referred to as entity chunking, extraction, or identification is a task that attempts to recognize named entities from text. The task can be split into two sub-tasks:

1. Detect a named entity
2. Categorize the entity

For example, for sentence "My friend was in Prague in 2019" should recognize "Prague" and "2019" as named entities. Next, the named entities are

categorized. Most common categories for NER are: Person, Organization, Time, Location and Work of art. In our example, "Prague" should be categorized under Location and "2019" should fall under the Time category.

2.4.3.1 Tools for NER and Lemmatisation

Since this work focuses on Czech invoices, the tools for Lemmatisation and NER need to have pre-trained models for the Czech language. For that reason, NameTag 2[36] for NER and MorphoDiTa[37] for Lemmatisation are used to generate text specific features for our model.

Both NameTag 2 and MorphoDiTa are publitized by Institute of Formal and Applied Linguistics, Charles University under Mozilla Public License 2.0 and authors of both works are Jana Strakova and Jan Straka.

2.4.4 Metrics for evaluation and comparison

The following metrics have been defined (apart from the Levenshtein distance, which is referenced) to properly evaluate tools and parameters of tools in the Experimentation chapter. These metrics also help to understand the relationship between the labeled data generated by the labeling tool and the raw data generated by Text Detection and Recognition models.

Invoice and Coordinates
Let invoice image I have a width $w(I)$ and height $h(I)$. A pair (x, y) is then a coordinate on the invoice image I, where $x \in [0, w(I)]$ and $y \in [0, h(I)]$. Then,

$$C(I) = [0, h(I)] \times [0, w(I)]$$

is the set of all coordinates for the invoice I.

Box Intersection
Let a bounding box of invoice image I be defined as a pair of 2 coordinates $(c_1, c_2) \in C(I) \times C(I)$ where for $x_1, y_1 \in c_1 \wedge x_2, y_2 \in c_2 : x_1 \leq x_2 \wedge y_1 \leq y_2$. Let $a, b \in C(I) \times C(I)$ be two different bounding boxes on invoice I. An intersection $a \cap b$ exists when for $x_{a1}, y_{a1} \in c_{a1} \wedge x_{a2}, y_{a2} \in c_{a2} \wedge x_{b1}, y_{b1} \in c_{b1} \wedge x_{b2}, y_{b2} \in c_{b2}$:

$$min(x_{a2}, x_{b2}) - max(x_{a1}, x_{b1}) \geq 0$$

and

$$min(y_{a2}, y_{b2}) - max(y_{a1}, y_{b1}) \geq 0.$$

Box Match metric
For invoice image I, let $\mathbb{A} \in C(I) \times C(I)$ be a set of bounding boxes generated by a Text Detection method. Let $\mathbb{B} \in C(I) \times C(I)$ be a set of bounding boxes that have been generated by a human through the labeling tool. Each $b \in \mathbb{B}$ also belongs to a specific type $T(b) = c$, where $c \in \mathbb{C}$ and $\mathbb{C} = \{ID, ICO, DueDate, Amount\}$. For $\forall b \in \mathbb{B}$ and $\forall a \in \mathbb{A}$, if $\exists(b \cap a)$, it means an intersecting box for that the type of b was found, therefore for $c = T(b)$ we set the value $intersected(c) = 1$, otherwise $intersected(c) = 0$ is set. The Box Match metric $BoxMatch(I)$ is then calculated as:

$$BoxMatch(I) = \frac{\sum_c^{\mathbb{C}} intersected(c)}{|\mathbb{C}|}$$

where $|\mathbb{C}|$ is the number of types tracked.

Levenshtein distance
Levenshtein distance[38] is a string metric for measuring the difference between two sequences. Informally, the Levenshtein distance between two words is the minimum number of single-character edits (insertions, deletions or substitutions) required to change one word into the other. It is named after the mathematician Vladimir Levenshtein.

The recursive definition of Levenshtein distance between two strings a, b is given by $\text{lev}(a, b)$ where

$$\text{lev}(a,b) = \begin{cases} |a| & \text{if } |b| = 0, \\ |b| & \text{if } |a| = 0, \\ \text{lev}(\text{tail}(a), \text{tail}(b)) & \text{if } a[0] = b[0] \\ 1 + \min \begin{cases} \text{lev}(\text{tail}(a), b) \\ \text{lev}(a, \text{tail}(b)) \\ \text{lev}(\text{tail}(a), \text{tail}(b)) \end{cases} & \text{otherwise.} \end{cases}$$

where $|x|$ is the length of string x, the tail of some string x is a string of all but the first character of x, and $x[n]$ is the nth character of the string x, starting with character 0.

Accuracy of Value metric
Consider the type set $\mathbb{C} = \{ID, ICO, DueDate, Amount\}$ and the bounding box sets $\mathbb{A} \in C(I) \times C(I)$ and $\mathbb{B} \in C(I) \times C(I)$ from Box Match metric definition. Let $text(a)$ be the reference string generated by a human through the labeling tool contained in bounding box $a \in \mathbb{A}$. Let $text(b)$ be the string of the bounding box $b \in \mathbb{B}$ generated by the application (bounding box generated by Text Detection model and text generated by OCR engine) and $\exists(b \cap a)$. The Accuracy of value for a specific type $c = T(b)$, where $c \in \mathbb{C}$ is then calculated

as:

$$Accuracy(c) = \frac{|text(a)| - \text{lev}(text(a), text(b))}{|text(a)|}$$

where $|x|$ is the length of string x and $\text{lev}(x, y)$ is the Levenshtein distance between strings x and y.

2.5 Labeling tool technologies

The following tools and technologies have been chosen for the labeling tool.

JavaScript
JavaScript is one of few core technologies for the World Wide Web, used in 97% of web applications[39], making it the most popular programming language in the world. Due to it's popularity, there is a wide range of third-party libraries to choose from, which makes creating web applications much easier.

React.js
React.js is a open-sourced front-end JavaScript library backed by Facebook, which focuses on building user interfaces or components. Just like JavaScript, React.js has a strong community and there is no shortage of third-party libraries to use, making it an obvious pick for developing the front-end of the labeling tool.

FastAPI
FastAPI is a relatively new Python web framework mainly used to build APIs, distinguishing itself as a very high performance and easy to use framework (supposedly just as fast as NodeJS or GO). It was a though pick, as there are more popular predecessors such as Django and Flask, but the performance and ease of use were the deciding factors.

AWS RDS
AWS RDS is a distributed relational database service offered by Amazon Web Services. In this work we use AWS RDS to host a PostgreSQL databse for our data. Using a commercial service for our database is only due to convenience reasons and can easily be replaced by a open-sourced solution running on a local machine. Furthermore, this work does not focus on the use of database, therefore it does not defeat the purpose of this work.

AWS S3
AWS S3 is a storage service offered by Amazon Web Services. This work uses the AWS S3 bucket to store invoice images. Just like AWS RDS, it is mainly used due to convenience purposes and can easily be replaced by storing invoice images on a local machine where the labeling tool would run on.

Chapter 3

Design

In order to reach the final model for Structured Data Extraction, the first thing we need are training data. To obtain training data, which in this work are identified and localized important values for invoice images, we first need to create a tool that can help us generate a larger number of this data. As such, we start with the labeling tool.

3.1 Functional Requirements of the Labeling tool

The following requirements are the fundamental ones for the labeling tool in order to produce training data.

- FR1: The user should be able to add or remove invoice images to the labeling tool.
- FR2: The user should be able to input correct data he can read from an invoice.
- FR3: The user should be able to mark the area around an important value on an invoice image.
- FR4: The user should be able to edit the data.

More functional requirements are then added when finishing the solution for Invoice Data Capture.

- FR 5: The application should be able to extract key data from uploaded invoice images (Invoice Data Capture task).
- FR 6: The user should be able to re-train the model after labeling more invoices.

3.2 Non-functional Requirements of the Labeling tool

- NFR1: Inputting correct data for an invoice should be easy, fast and intuitive, eg. by drawing on an invoice image, color distinctions and auto-reading capabilities.
- NFR2: The data created by the user should be saved persistently so other users can join in on the labeling work.
- NFR3: The Invoice Data Capture model should have a reasonable accuracy to make it usable.

3.3 Implementation steps

Bellow we can see the planned implementation steps for this work. There is also the diagram 3.1 of the designed application for reference.

1. Create labeling tool and obtain labeled data
2. Evaluate and compare Text Detection and Text Recognition models and methods, pick the best method
3. Using the best Text Detection and Recognition methods, extract data from invoice images
4. Combine extracted data with the labeled data to obtain training data
5. Evaluate and compare different ML classification models on training data, pick the best performing model
6. Train a ML based model for Structured Data Extraction
7. The combination of Text Detection, Text Recognition and Structured Data Extraction is the solution for the Invoice Data Capture task
8. Integrate the solution into the labeling tool

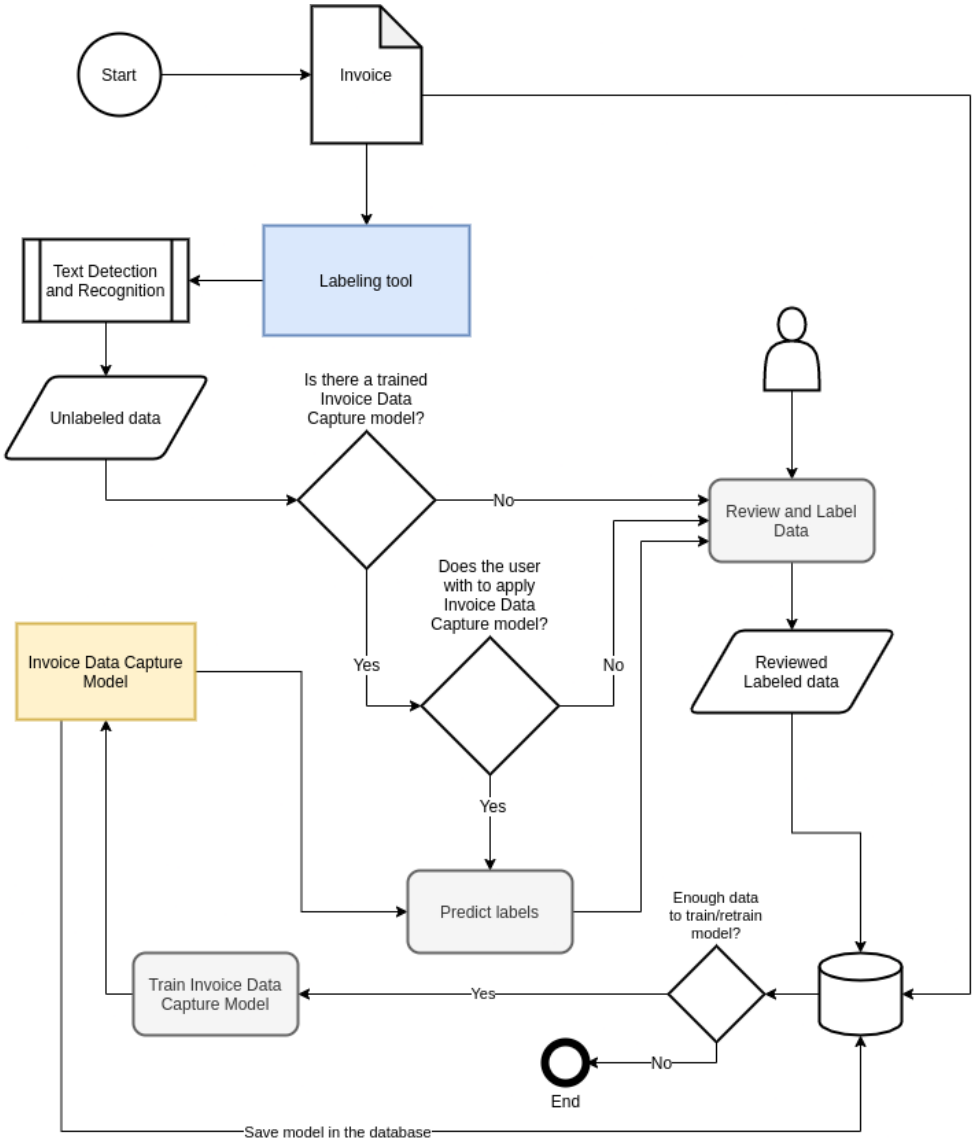

Figure 3.1: Design diagram of the application

Chapter 4

Labeling tool implementation

The labeling tool is critical for this work moving forward, as we require a good sample of labeled data for evaluating and comparing tools that could be used for Text Extraction and Recognition as well as evaluating and comparing machine-learning based models for Structured Data Extraction, since supervised machine-learning requires training data by default.

The labeling tool's main functionality is to help the user locate and label key values on an invoice. Apart from that, the tool also handles storing and uploading of the invoices for convenience purposes. Finally, the machine-learning based solution for extraction of structured data from invoices, which is the main product of this work, will also be integrated into the labeling tool to assist with further tagging.

4.1 Front-end

4.1.1 Home page/Invoices page

The homepage of the labeling tool shows all invoice images that have been uploaded to the tool (4.1). Each invoice on the homepage is viewed as a small clickable card with a preview image of a invoice, name of that invoice and special tags, each representing a state for an invoice:

- No tag: Invoice cards without a tag represent invoices that have been freshly uploaded.
- Generated: Generated tag represents invoices that were scanned by the application and are waiting for review.
- Reviewed: Reviewed tag represents invoices that have been reviewed by a human.

Each invoice cards also contain two basic operation buttons: Edit Invoice and Delete. Edit Invoice has the same functionality just like simply clicking

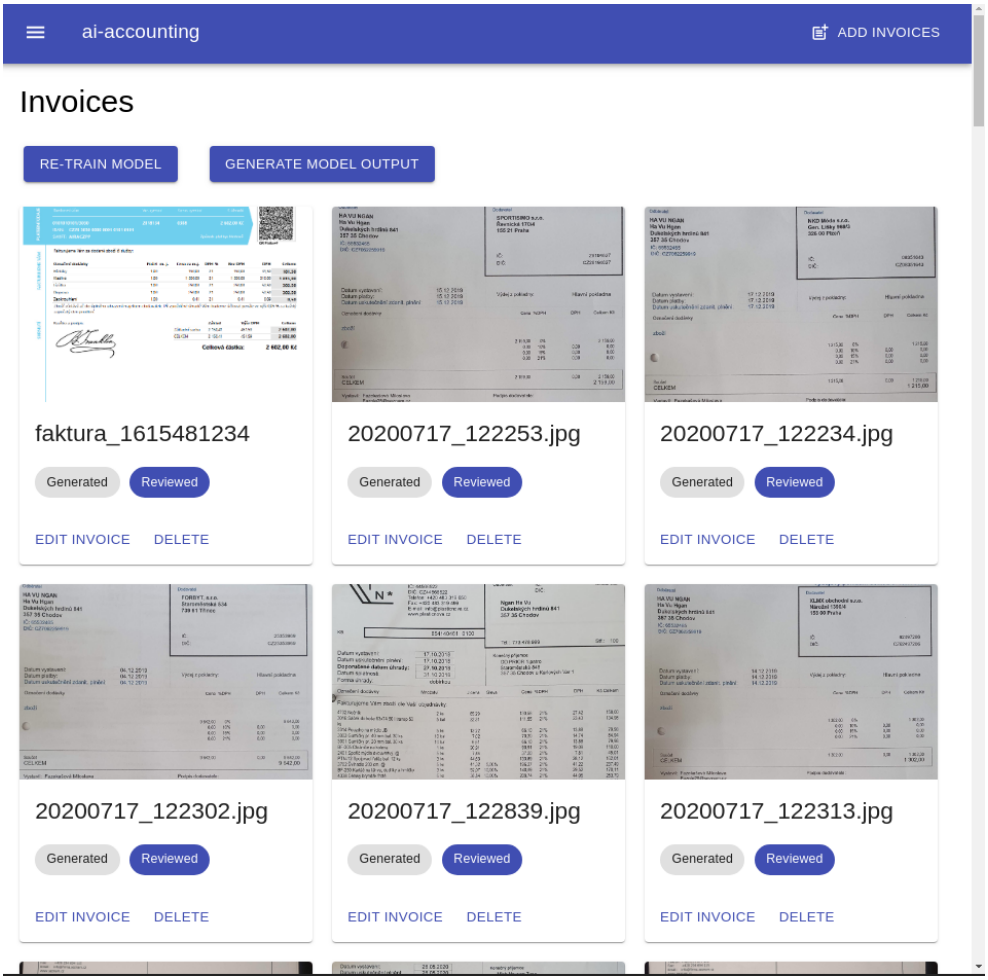

Figure 4.1: Front-end homepage

on the invoice card, which brings us to an editing environment described later on. Delete removes that specific invoice from the application.

Apart from the invoice cards, the labeling tool also contains two buttons on the upper part of the home page. Re-train model button takes all the data from reviewed invoices and uses them to train a new model. Button Generate Model Output generates predicted values for all invoices without a tag (new invoices). Once done, the invoices with newly generated predicted values are tagged as Generated. Both of these operations take quite some time, which is why loading animations were implemented for them.

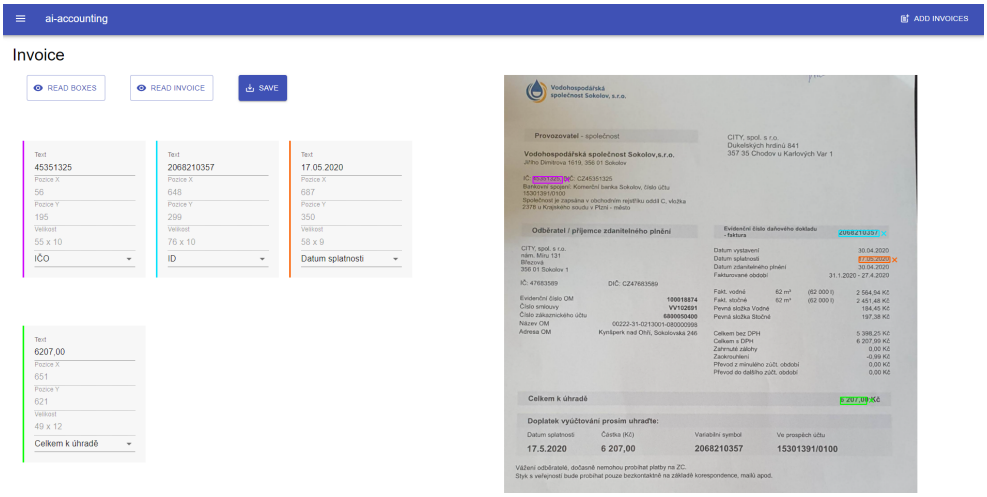

Figure 4.2: Front-end editing page

4.1.2 Upload page

The navigation part on the home page contains only one button called Add Invoices, which is located on the right. Clicking that button gets us to the Upload page, where the user can add Invoice images to the application by simply dragging the images into the rectangle, or click on the rectangle to search for the invoice images on his/hers local machine.

4.1.3 Editing page

Upon clicking an invoice card and moving into the editing environment (4.2). In order to keep the task of labeling invoices as easy and quick as possible, the tool views the selected invoice on a canvas, which is located on the right side of the page. The user can draw bounding boxes on the invoice image by clicking and dragging, preferably around a location of a key value. Creating a bounding box automatically creates a corresponding card on the left of the page, which shows the data of the bounding box.

The data cards corresponding to the bounding boxes on the invoice image contain coordinates and size of the bounding box, which are not editable by the user other than by dragging or resizing the bounding box. Upon creation of a new bounding box, the text and select input fields are set to default values, while the color of the bounding box and data card are set to black. Clicking the select input field shows a drop-down menu, where the user can choose which of the 4 types (ID, ICO, Due Date and Amount) the located value represents. Choosing a type changes the color of the data card and corresponding bounding box accordingly to help distinguish between each

connected pair:

- ID: Blue
- ICO: Purple
- Due Date: Orange
- Amount: Green

The text input field is empty on creation and is supposed to contain the text value, which is marked on the invoice image by the corresponding bounding box. The user can either write the correct value into the input field, or leverage the Read Boxes button on the top left corner, which attempts to read the text inside all the defined bounding boxes.

The user also has the ability to skip the previous steps altogether, by using the button Read Invoice, which lets the application attempt to locate and read all the key values itself. However, the output is not always correct, therefore it is expected for the user to review the data and in some cases fix them, before proceeding.

After making sure that everything is correct for the specific invoice, the user can save the data by clicking on the Save button. The application then saves all the data inputted by the user and tags the invoice as Reviewed.

4.2 Back-end

The labeling tool's back-end is written in Python utilizing the FastAPI framework and communicates with the front-end through REST requests. For communication with the PostgreSQL database on AWS RDS, psycopg2 was used, which is the most popular PostgreSQL database adpater for Python. For communication with AWS S3, this work uses boto3, which is an AWS SDK for Python.

4.2.1 Endpoints

All the defined tables and their attributes in the PostgreSQL database can be seen in figure 4.3 and should serve as a reference while reading the endpoint definitions. The following endpoints are split into Simple Endpoints and Complex Endpoints. Simple Endpoints were created first and were the base of the labeling tool. Complex Endpoints were added at a later date, after finalizing the Invoice Data Capture model for which the training data from the base labeling tool were used.

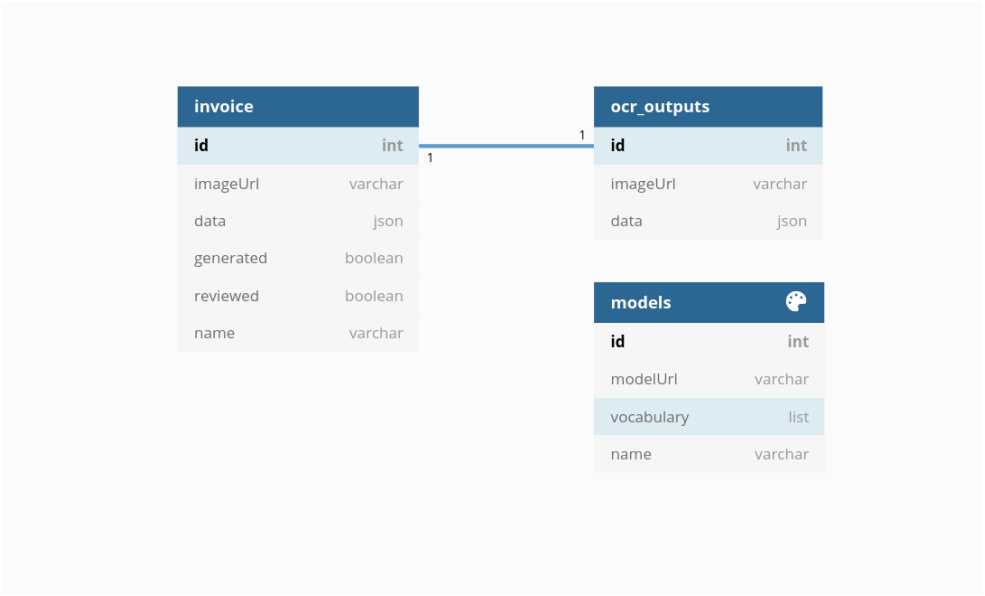

Figure 4.3: Database schema

4.2.1.1 Simple Endpoints

POST /upload-invoice
Endpoint for uploading invoices. Saves the image of the uploaded invoice in the S3 Bucket and inserts a new row into the invoices table. The newly added row contains id, imageUrl (URL of the image in S3 bucket) and name. Attributes data, generated and reviewed are empty for newly added invoices.

POST /read-boxes
This endpoint is called after the user finishes drawing bounding boxes on an invoice image and clicking Read Boxes button. The endpoint receives the data of bounding boxes and uses it to crop the invoice image. Each cropped image is then attempted to be read by Tesseract OCR. The text outputs of the OCR engine are then returned.

POST /update-data
This endpoint is supposed to be called after the user finishes reviewing the text of each key value of a specific invoice and then clicking on Save button. When called, it converts the correct texts and bounding boxes into json format and stores the data in the invoice table under attribute data.

DELETE /delete-invoice/[id]
This endpoint attempts to delete an invoice when given an invoice id. When

invoice id is found in the database, it deletes the invoice image in S3 bucket and drops the row of that specific invoice from the invoice table.

4.2.1.2 Complex Endpoints

GET /train-classification-model
This endpoint is called when the button Re-train Model on the home-page is clicked. It takes the raw OCR output stored in the ocr_outputs table as well as the labeled data from the invoices table for all invoices that have been reviewed and transforms them into training data. A new Data Extraction model is then trained on this data and then uploaded to the S3 bucket. The models table also gets a new row, consisting of the id, modelUrl, vocabulary and name of the model. A more detailed explanation of model training can be found later in chapter Extraction of Structured Data.

POST /generate-predictions
This endpoint is called when the Generate Model Output button on the home-page is clicked. Firstly, it generates the raw OCR output of Tesseract OCR with CRAFT Text Detection model for invoices that did not have the raw OCR output generated before. Next, it downloads the latest model by looking at the bottom row of the models table. Lastly, it applies the model on all invoices that do not have the reviewed tag, saves the prediction into the data attribute of the invoice in the invoice table, and sets the generated tag to True.

POST /classify-invoice-data
This endpoint works similarly to the /generate-predictions endpoint. It is called by clicking the Read Invoice button in the Editing page of a specific invoice. Apart from the fact that this endpoint works for a single invoice, the other difference is that this endpoint does not use the CRAFT model for Text Detection like the /generate-predictions endpoint. Instead, it uses only the Tesseract OCR for both Text Detection and Recognition before applying the Data Extraction model, which greatly improves the speed at the expense of a slight accuracy drop, making this method much more appropriate for real-time data extraction. Note that this method is only relevant to newly added invoices without the Generated tag.

Chapter 5

Invoice Data Capture model implementation

Once we finish the labeling tool and generate labeled data, we can then use the data to evaluate all the models and tools for Text Detection and Recognition mentioned in sections 2.2 and 2.3. The best technologies and configurations of Text Detection, Recognition and Image processing techniques (which we will examine in the Evaluation and Comparison chapter 6 later on) should then be used to extract raw data from invoices. An example of how this data looks was seen in 2.4 section.

The missing piece to achieve the solution for the task Invoice Data Capture model is solving the Structured Data Extraction task. As mentioned in 2.4 section, this task can be interpreted as a classification problem, for which we have picked the most popular machine learning models (listed in subsection 2.4.1). To train the models however, we first need to obtain training data from the labeled data and extracted raw data.

5.1 Feature engineering

Feature engineering is a process of using expert knowledge to extract features (characteristics) from raw data. In our case, raw data is the output of Text Detection and Recognition tasks. From our findings and observations in the Exploratory analysis section (2.4.2), we transform the information hidden in the raw data into a form that our model can understand.

Firstly, as our key values that we want to identify are mostly consisting of numbers, we split all text from the invoice into two groups: values that are mostly numbers and values that are mostly text.

Number dominant values
For each number dominant value, we determine whether they match defined

regex expressions for each key value (ID, ICO, Due Date and Amount). Position of value is described by splitting the invoice into segments, which is achieved by grouping the coordinates of the value. Font size of value is described by the value's height of the bounding box, which is split into groups. For the groups of heights in an invoice document, we also calculate their number of occurrence.

Character dominant values
Character dominant values are grouped together by distance to number dominant values, with priority for the same line. This way, we capture relevant text that help in identifying a key value (eg. "IC/ICO" for ICO, "Celkem k uhrade" for Amount, "Datum splatnosti" for Due Date). Further feature engineering for character dominant values are in the next section.

5.1.1 Applying Natural Language Processing

To properly convert the character dominant values into features, we need to apply Natural Language Processing methods. As such, we use the Named Entity Recognizer NameTag2 to identify named entities in the groups of character dominant values. The categories of identified named entities are then features for the number dominant value. Next, we use the Lemmatizer MorphoDita to convert all words of character dominant values into their Lemmatized form. This way, we ensure that words such as "Celkem" and "Celkově" do not create two separate features after converting every word in all character dominant values into features.

5.2 Building data extraction model

All the steps in previous sections led to building the model for extraction of structured data. In detail, the model's task is to pick the best candidate for each value type (ID, ICO, Due Date and Amount) for a given invoice, which is a classification problem. The model is to learn solve this task using the data labeled by a human through the labeling tool, as well as the extracted information from the Feature engineering section. The core of this model is the best performing ML classifier, which is described in the next chapter Evaluation and Comparison (6).

For usability reasons, the model is integrated back into the labeling tool, where it can help with the labeling process, which in turn, would create more training data for the model and eventually result in a better performing model after re-training. These functions are implemented as the Complex endpoints (4.2.1.2) of the labeling tool's back-end.

Chapter 6

Evaluation and Comparison

The following experiments were conducted on a local machine with AMD Ryzen 5 2600X Six-Core Processor and Radeon R7 260X Graphics card, using 60 labeled invoices.

6.1 Text Detection and Image Pre-processing

Tesseract OCR is the used solution for the Text Recognition task, which was already mentioned in the Analysis chapter (2). In this section, we focus on the effects of different Text Detection methods as well as various image pre-processing techniques[40] on the reading accuracy for Tesseract. The tested Text Detection models are CRAFT and EAST along with Tesseract's own Text Detection method.

For evaluation and comparison between all the methods, we utilize the Box Match and Accuracy of Value metrics defined in the Metrics for evaluation and comparison section (2.4.4).

6.1.1 Tesseract Text Detection method

The table 6.1 shows the effects of mentioned image pre-processing techniques on the accuracy of Tesseract OCR engine output for our invoice data. Each value represents the mean Accuracy of Value for ID, ICO, Due Date and Amount for all invoice images. Tesseract OCR engine was used for both Text Detection and Recognition, which is done by utilizing Tesseract's page segmentation mode 12.

Simply re-scaling and applying gray-scale on the images of invoices in our dataset yielded the best results. Although combining all of the image pre-processing effects resulted in a better "Due Date" accuracy, it came at the cost of worse "Amount" accuracy. The average time for Text Extraction with Tesseract on an invoice documents is 8 seconds.

	Box match	ID	ICO	Due Date	Amount
No Pre-processing	95%	98.1%	87.8%	81.8%	79.5%
Re-scaling and Grayscale	97.5%	98.5%	89.7%	86.8%	87%
Binarization	94.5%	97.9%	88%	84.5%	76.7%
Noise reduction	91.5%	94.8%	89%	82.4%	76.7%
Dilation and Erosion	94.5%	98.36%	88.9%	83.4%	84.4%
All of the above combined	95%	98.45%	87.7%	87.8%	83.2%

Table 6.1: Tesseract OCR accuracy by image pre-processing techniques with Tesseract's native Text Detection

6.1.2 CRAFT

The table 6.2 shows the effects of applying CRAFT for Text Detection. The output of the CRAFT model is used to crop the invoice image, which is then read by Tesseract OCR engine in page segmentation mode 7. (Page segmentation 13 was used as well, but resulted in worse results, which is not shown in this table).

	Box match	ID	ICO	Due Date	Amount
No Pre-processing	100%	86%	84%	94.5%	94.4%
Re-scaling and Grayscale	100%	86%	84.2%	92.8%	94.4%
Binarization	100%	86%	83.5%	96%	95%
Noise reduction	98%	83.5%	80%	93%	93.3%
Dilation and Erosion	100%	86%	84.2%	93.6%	94.4%
All of the above combined	98%	83.9%	79.7%	94.4%	94.5%

Table 6.2: Tesseract OCR accuracy by image pre-processing techniques with CRAFT Text Detection

The most notable improvement by applying the CRAFT text detection is in Box match, achieving a perfect score of 100%. There is also a considerable improvement in accuracies for Amount and Due Date, both around 95% . However, the accuracy for ID has been reduced by more than 10% across the board, while accuracy for ICO has also sadly detoriated. As for the effects of Image pre-processing techniques, no image pre-processing seems to already have good results, where by adding image pre-processing only results in marginal improvements.

The results clearly indicate that CRAFT performs it's job of Text Detec-

tion perfectly, however, following up with OCR on the cropped images yields mixed results. There is also a drawback in computational speed. Text extraction with CRAFT took 40 seconds for an invoice image on average, compared to 8 seconds in the previous section.

6.1.3 EAST

	Box match	ID	ICO	Due Date	Amount
No Pre-processing	93.8%	48%	46.4%	45.6%	50%

Table 6.3: Tesseract OCR accuracy by image pre-processing techniques with EAST Text Detection

Compared to CRAFT, EAST Text Detection is a little faster (32 seconds on average), but at the expense of accuracy as seen in table 6.3. Although the Box match metric is pretty high, it is not higher than Tesseract's native Text Detection method. Furthermore, the position of the bounding boxes are not perfect, resulting in a significant drop of Accuracy of Value metric across the board. Due to the fact that EAST performed worse on our dataset than Tesseract's native Text Detection and was much slower, we skip applying Image pre-processing for this Text Detection method.

6.1.4 Best Text Detection

The findings clearly show the dominating performance of the CRAFT model. Although the Accuracy of Value dropped for ID and ICO, the improvement of Accuracy of Value for Due Date and Amount make up for it. A major drawback for CRAFT is computational speed, where its 5 times slower than using Tesseract by itself. In conclusion, we can split these models into two different use-cases: Tesseract Text Detection and Recognition can be used for real-time extraction, useful for example in mobile applications, whereas CRAFT Text Detection in combination with Tesseract Text Recognition could be run overnight on a collection of images, and reviewed the next day, for accuracy reliant use-cases.

6.2 Comparing Machine Learning Models

To compare machine learning models, we split the labeled data set randomly into training data set, consisting of 77% and testing data set, consisting of 33% of the labeled data set. The data requires prediction for 5 classifications: ID, ICO, Due Date, Amount and None, where the None classification represents all the other number dominant values on invoices. The sample size of the None classification is therefore much larger than the sample size of previous

4 classifications combined (more than 90% in labeled data). Due to this fact, we define our own accuracy score, which takes only the first 4 classifications into consideration. The resulting scores are the Training Accuracy, which is performed on training data, and Testing Accuracy, performed on testing data. For comparison, we retain the None classification in the form Default Accuracy.

The table 6.4 shows the achieved Training Accuracy and Testing Accuracy, as well as the Default Accuracy for each classification model. The best performing models seem to be ensemble methods (Random Forest and Gradient Boosting). The 100% accuracy on the training data set might seem weird and is most likely due to overfitting[41]. Despite that, they still have the best predictive power (73% and 70%) on the testing data set of all the models shown. Logistic Regression and SVM did not perform bad either, unlike K-NN or worse, Naive Bayes.

	Training Accuracy	Testing Accuracy	Default Accuracy
Logistic Regression	84%	67%	98%
K-NN	53%	20%	95%
SVM	85%	58%	98%
Naive Bayes	-1%	-1.3%	81%
Random Forest	100%	73%	99%
Gradient Boosting	99%	70%	99%

Table 6.4: Machine-learning classification models comparison by accuracy

Apart from Gradient Boosting, all models finished training before 1 second, while Gradient Boosting took 7 seconds to train.

6.2.1 Hyperparameter tuning

Since we suspect the overfitting on the ensemble methods, let us try different combinations of parameters for the models to find the right configuration for our data set. This process is called hypterparameter tuning[42]. The figure 6.1 shows the accuracy development through different combinations of $n\_estimators$ and max_depth parameters for the Random Forest Classifier. As we can see, low values max_depth causes dips in accuracy while $n\_estimators$ have a positive impact on the accuracy on testing data until the value 400. The resulting best parameter values for the Random Forest Classifier were in this case 151 for $n\_estimators$ and 28 for max_depth.

The Gradient Boosting Classifier is much slower to train, therefore we opted for a smaller number of parameter combinations. Again, the figure 6.2 shows the accuracy development, where smaller max_depth were causing dips in accuracy, while $n\_estimators$ larger than 50 show worsening accuracy on

the test data. The best parameters found are 37 *n_estimators* and *max_depth* 4.

Random Forest Classifier took about 400 seconds to try out 600 combinations of parameters, while Gradient Boosting Classifier took around 300 seconds to try out 300 parameter combinations.

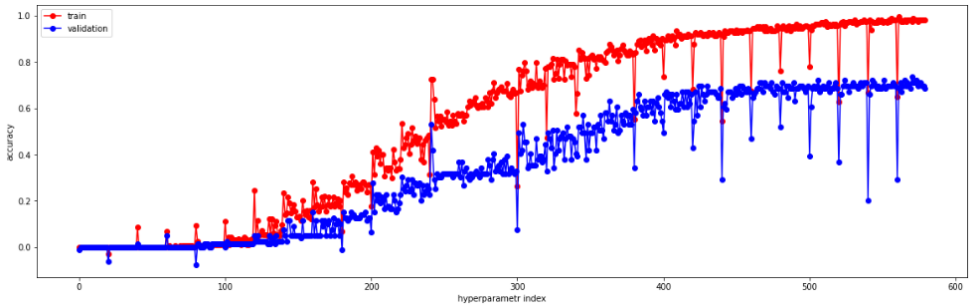

Figure 6.1: Random Forest Classifier Hyperparameter tuning

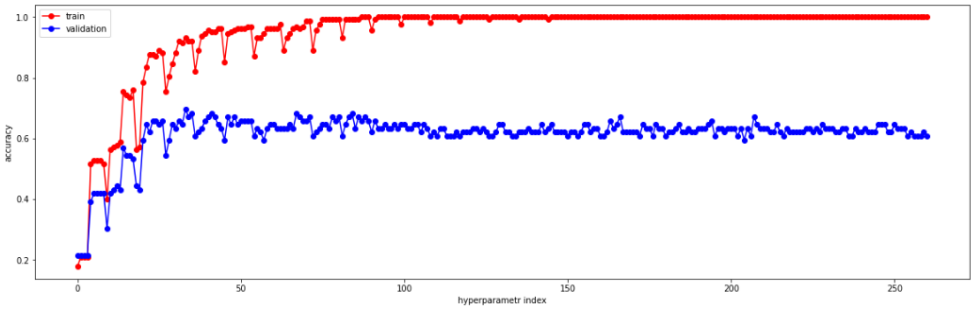

Figure 6.2: Gradient Boosting Classifier Hyperparameter tuning

6.2.2 Most important features

One of Random Forest Classifier's advantages is the ability to output feature importance scores, which is shown in table 6.5. Here we can find which features we created in the Feature Engineering section (5.1) that had the best predictive ability. Most of the features we can find in the table are as expected: "splatnosti", "uhrade", "ic", "celkem", "castka" are the texts that usually occur around our key values. The "top_bins" and "left_bins" are features that convey the location segment on invoices, while "height_bin" and "height_bin_scarcity" should represent the grouped font sizes and scarcity of the groups on each invoice image. "ID candidate" and "ICO candidate" rep-

rank	score	feature
1	0.116962	splatnosti
2	0.060957	uhrade
3	0.051209	text_length
4	0.049187	top_bins
5	0.048335	height_bin_scarcity
6	0.040368	doklad
7	0.038229	castka
8	0.037184	left_bins
9	0.032298	vyuctovani
10	0.028949	ICO candidate
11	0.025694	ic
12	0.024192	danovy
13	0.022357	faktura
14	0.022340	height_bins
15	0.018727	dic
16	0.014688	martin
17	0.013226	ID candidate
18	0.012690	celkem
19	0.012122	placeno
20	0.011703	platbu

Table 6.5: Feature importances from Random Forest Classifier

resent whether the value matches the regular expression formats. Here, it is quite weird that "Due Date candidate" and "Amount candidate" did not make it to the list of top 20 features.

Chapter 7

Conclusion

This work introduced a solution for the task of Template-less Extraction of Structured Data from Czech Invoices (also called Invoice Data Capture), which is currently only available commercially. In this work, the Invoice Data Capture task was split into three sub-tasks: Text Detection, which locates the bounding boxes of texts on an invoice image, Text Recognition, which reads the text contained in the bounding boxes and Structured Data Extraction, which identifies important information from the outputs of previous two tasks.

Firstly, we analyzed the current open-source and commercial Invoice Data Capture applications. We then studied the existing approaches and tools of Text Detection, Text Recognition (OCR), Supervised Machine Learning Classification, Natural Language Processing and Image Pre-processing.

Next, we implemented the labeling tool, which was used to label the essential parts of invoice images, creating labeled data for evaluation and comparison of all considered tools, models and methods, as well as for training of the machine-learning models.

The labeled data was then used to test and evaluate the accuracy of Text Detection and Recognition models. The results indicated the dominant performance in accuracy, achieving 100% in Text Detection and around 90% in Text Recognition when applying the CRAFT model against the EAST model and Tesseract's native text detection method. However, the major drawback of the CRAFT model is the computational speed, therefore the Tesseract's native text detection method, which is the second best accurate, is retained for the use-case of real-time extraction of structured data from an invoice.

Using the best Text Detection and Recognition method, we extracted raw data from invoices and combined them with the labeled data, obtaining training data to test and compare ML classifiers for the Structured Data Extraction task. Our findings showed that among 6 most popular ML classification models, Random Forest Classifier had the best final accuracy at 100% for training data and 73% for out of sample data, making the resulting solution usable, thus the goal of this thesis was achieved, while also fulfilling all instructions

and requirements of the assignment.

There is certainly room for improvements in this work. The labeling tool could be enhanced by adding authentication for support of multiple users, implementing more functions and improving the user interface, while the model for Invoice Data Capture could be improved by introducing deep learning neural networks.

Bibliography

[1] Papers With Code. https://paperswithcode.com/, 2021, [Online; accessed 22-June-2021].

[2] Petr Baudis, rossum.ai. THE REAL COST OF INVOICE AUTOMATION: THE TCO OF INVOICE DATA CAPTURE (PART 3). https://rossum.ai/blog/the-tco-of-invoice-data-capture-cognitive-cloud-based-solution-3, 2019, [Online; accessed 22-June-2021].

[3] Compare Data Extraction Tools. https://aimultiple.com/data-extraction-tools, [Online; accessed 6-June-2021].

[4] Holt, X.; Chisholm, A. Extracting structured data from invoices. In *Proceedings of the Australasian Language Technology Association Workshop 2018*, Dunedin, New Zealand, Dec. 2018, pp. 53–59. Available from: https://www.aclweb.org/anthology/U18-1006

[5] Ohki, M.; Zervakis, M. E.; et al. 3-D Digital Filters. In *Multidimensional Systems: Signal Processing and Modeling Techniques*, *Control and Dynamic Systems*, volume 69, edited by C. Leondes, Academic Press, 1995, pp. 49–88, doi:https://doi.org/10.1016/S0090-5267(05)80038-6. Available from: https://www.sciencedirect.com/science/article/pii/S0090526705800386

[6] Unknown. Dilation and Erosion — The MathWorks. http://matlab.izmiran.ru/help/toolbox/images/morph2.html, 2005, [Online; accessed 22-June-2021].

[7] Murzova, A.; Seth, S. Otsu's Thresholding with OpenCV. 2020, [Online; accessed 22-June-2021]. Available from: https://learnopencv.com/otsu-thresholding-with-opencv/

[8] International Conference on Document Analysis and Recognition (ICDAR). https://icdar2021.org/, 2021, [Online; accessed 22-June-2021].

[9] Zhou, X.; Yao, C.; et al. EAST: An Efficient and Accurate Scene Text Detector. *CoRR*, volume abs/1704.03155, 2017, 1704.03155. Available from: http://arxiv.org/abs/1704.03155

[10] Baek, Y.; Lee, B.; et al. Character Region Awareness for Text Detection. 2019, [Online; accessed 16-May-2021], 1904.01941.

[11] Akyon, F. C. Fast and accurate text detection library built on CRAFT implementation. 2021, [Online; accessed 22-June-2021].

[12] Heinisuo, O.-P. Wrapper package for OpenCV python bindings. 2021, [Online; accessed 22-June-2021].

[13] Zhou, X.; Yao, C.; et al. EAST: An Efficient and Accurate Scene Text Detector. 2017, 1704.03155.

[14] Rahman, A. What is Optical Character Recognition(OCR)? 2019, [Online; accessed 6-December-2020]. Available from: https://medium.com/analytics-vidhya/what-is-ocr-f67c9ab218bf

[15] Han, T. Our Search for the Best OCR Tool, and What We Found. 2019, [Online; accessed 6-December-2020]. Available from: https://source.opennews.org/articles/so-many-ocr-options/

[16] Wikipedia contributors. HOCR — Wikipedia, The Free Encyclopedia. 2021, [Online; accessed 18-June-2021]. Available from: https://en.wikipedia.org/w/index.php?title=HOCR&oldid=1028360158

[17] Wikipedia contributors. ALTO (XML) — Wikipedia, The Free Encyclopedia. 2021, [Online; accessed 18-June-2021]. Available from: https://en.wikipedia.org/w/index.php?title=ALTO_(XML)&oldid=1026197877

[18] Paszke, A.; Gross, S.; et al. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems 32*, edited by H. Wallach; H. Larochelle; A. Beygelzimer; F. d'Alché-Buc; E. Fox; R. Garnett, Curran Associates, Inc., 2019, pp. 8024–8035. Available from: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

[19] Abadi, M.; Agarwal, A.; et al. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. 2015, software available from tensorflow.org. Available from: https://www.tensorflow.org/

[20] Amidi, A.; Amidi, S. Recurrent Neural Networks cheatsheet — Stanford University. https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-recurrent-neural-networks, 2021, [Online; accessed 22-June-2021].

[21] Chen, J. Neural Network. 2020, [Online; accessed 22-June-2021]. Available from: https://www.investopedia.com/terms/n/neuralnetwork.asp

[22] Brownlee, J. A Gentle Introduction to Long Short-Term Memory Networks by the Experts. 2017, [Online; accessed 6-December-2020]. Available from: https://machinelearningmastery.com/gentle-introduction-long-short-term-memory-networks-experts/

[23] Smith, R. Tesseract OCR. https://github.com/tesseract-ocr/tesseract, 2005, [Online; accessed 6-December-2020].

[24] Lee, M. pytesseract, python wrapper for Google's Tesseract-OCR. 2020, [Online; accessed 22-June-2021].

[25] json.org. Introducing JSON. 2020, [Online; accessed 22-June-2021]. Available from: https://www.json.org/json-en.html

[26] Joshi, H. Data extractor for PDF invoices - invoice2data. https://github.com/invoice-x/invoice2data, 2005, [Online; accessed 6-December-2020].

[27] Hope, C. Regex. 2020, [Online; accessed 22-June-2021]. Available from: https://www.computerhope.com/jargon/r/regex.htm

[28] Shetty, B. AN IN-DEPTH GUIDE TO SUPERVISED MACHINE LEARNING CLASSIFICATION. 2020, [Online; accessed 22-June-2021]. Available from: https://builtin.com/data-science/supervised-machine-learning-classification

[29] Pedregosa, F.; Varoquaux, G.; et al. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, volume 12, 2011: pp. 2825–2830.

[30] Čepek, M.; Klouda, K. BIE-VZD lecture 9, Czech Technical University in Prague, Faculty of Information Technologies. https://courses.fit.cvut.cz/BIE-VZD/lectures/files/2020/09/BI-VZD-09-en-slides.pdf, 2020, [Online; accessed 22-June-2021].

[31] Čepek, M.; Dedecius, K.; et al. BIE-VZD lecture 5, Czech Technical University in Prague, Faculty of Information Technologies. https://courses.fit.cvut.cz/BIE-VZD/lectures/files/2020/05/BI-VZD-05-en-slides.pdf, 2020, [Online; accessed 22-June-2021].

[32] Čepek, M.; Dedecius, K.; et al. BIE-VZD lecture 6, Czech Technical University in Prague, Faculty of Information Technologies. https://courses.fit.cvut.cz/BIE-VZD/lectures/files/2020/06/BI-VZD-06-en-slides.pdf, 2020, [Online; accessed 22-June-2021].

[33] Čepek, M.; Dedecius, K.; et al. BIE-VZD lecture 3, Czech Technical University in Prague, Faculty of Information Technologies. https://courses.fit.cvut.cz/BIE-VZD/lectures/files/2020/03/BI-VZD-03-en-slides.pdf, 2020, [Online; accessed 22-June-2021].

[34] Čepek, M.; Dedecius, K.; et al. BIE-VZD lecture 2, Czech Technical University in Prague, Faculty of Information Technologies. https://courses.fit.cvut.cz/BIE-VZD/lectures/files/2020/02/BI-VZD-02-en-slides.pdf, 2020, [Online; accessed 22-June-2021].

[35] Wikipedia contributors. Lemmatisation — Wikipedia, The Free Encyclopedia. `https://en.wikipedia.org/w/index.php?title=Lemmatisation&oldid=1022854248`, 2021, [Online; accessed 22-June-2021].

[36] Straková, J.; Straka, M.; et al. Neural Architectures for Nested NER through Linearization. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, Stroudsburg, PA, USA: Association for Computational Linguistics, 2019, ISBN 978-1-950737-48-2, pp. 5326–5331.

[37] Straková, J.; Straka, M.; et al. Open-Source Tools for Morphology, Lemmatization, POS Tagging and Named Entity Recognition. In *Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, Baltimore, Maryland: Association for Computational Linguistics, June 2014, pp. 13–18. Available from: `http://www.aclweb.org/anthology/P/P14/P14-5003.pdf`

[38] Wikipedia contributors. Levenshtein distance — Wikipedia, The Free Encyclopedia. `https://en.wikipedia.org/w/index.php?title=Levenshtein_distance&oldid=1011098657`, 2021, [Online; accessed 22-May-2021].

[39] w3techs.com. Usage statistics of JavaScript as client-side programming language on websites. `https://w3techs.com/technologies/details/cp-javascript/`, 2021, [Online; accessed 22-May-2021].

[40] Kuguoglu, K. How to use image preprocessing to improve the accuracy of Tesseract. 2018, [Online; accessed 22-May-2021]. Available from: `https://medium.com/free-code-camp/getting-started-with-tesseract-part-ii-f7f9a0899b3f`

[41] Tripathi, M. Underfitting and Overfitting in Machine Learning. 2020, [Online; accessed 22-June-2021]. Available from: https://datascience.foundation/sciencewhitepaper/underfitting-and-overfitting-in-machine-learning

[42] Jordan, J. Hyperparameter tuning for machine learning models. 2017, [Online; accessed 22-June-2021]. Available from: https://www.jeremyjordan.me/hyperparameter-tuning/

Appendix A

List of used abbreviations

AI Artificial Intelligence

API Application Programming Interface

AWS Amazon Web Services

ML Machine Learning

OCR Optical Character Recognition

RDS Relational Database Services

RNN Recurent Neural Networks

LSTM Long Short-Term Memory

NER Named Entity Recognition

KNN K-Nearest Neighbours

SVM Support Vector Machine

CRAFT Character Region Awareness for Text Detection

EAST Efficient and Accurate Scene Text Detector

ALTO Analyzed Layout and Text Object

NLP Natural Language Processing

JSON JavaScript Object Notation

REST Representational State Transfer

ID Identity

ICO Identifikační číslo osoby (Personal Identification Number)

VAT Value Added Tax

XML Extensible Markup Language

ICDAR International Conference on Document Analysis and Recognition

CUDA Compute Unified Device Architecture

PDF Portable Document Format

SDK Software Development Kit

SQL Structured Query Language

URL Uniform Resource Locator

Appendix B

Contents of the enclosed media

- `README.md` ..... brief description of the content
- `src`
  - `impl` ..... source code of the implementation
    - `client` ..... labeling tool front-end source code
    - `server` ..... labeling tool back-end source code
  - `thesis` ..... thesis text folder for LaTeX source code format
- `text` ..... thesis text folder
  - `thesis.pdf` ..... thesis text in PDF format