KTH ROYAL INSTITUTE
OF TECHNOLOGY

Degree Project in Technology

First cycle, 15 credits

# Exploring Machine Learning Solutions in the Context of OCR Post-Processing of Invoices

**JACOB DWYER**
**SARA BERTSE**

Stockholm, Sweden, 2022

# Exploring Machine Learning Solutions in the Context of OCR Post-Processing of Invoices

JACOB DWYER

SARA BERTSE

Degree Programme in Computer Engineering
Date: August 10, 2022

Supervisors: Johan Montelius, Helene Wahlqvist
Examiner: Gerald Quentin Maguire Jr
School of Electrical Engineering and Computer Science
Host company: Rillion AB
Swedish title: Utforskning av Maskininlärningslösningar för Optisk Teckenläsningsefterbehandling av Fakturor

© 2022 Jacob Dwyer and Sara Bertse

# Abstract

Large corporations receive and send large volumes of invoices containing various fields detailing a transaction. Such fields include VAT, due date, total amount, etc. One common way to automatize invoice processing is optical character recognition (OCR). This technology entails automatic reading of characters from scanned images. One problem with invoices is that there is no universal layout standard. This creates difficulties when processing data from invoices with different layouts. This thesis aims to examine common errors in the output from Azure's Form Recognizer general document model and the ways in which machine learning (ML) can be used to solve the aforementioned problem, by providing error detection as a first step when classifying OCR output as correct or incorrect. To examine this, an analysis of common errors was made based on OCR output from 70 real invoices, and a Bidirectional Encoder Representations from Transformers (BERT) model was fine-tuned for invoice classification. The results show that the two most common OCR errors are: $(i)$ extra words showing up in a field and $(ii)$ words missing from a field. Together these two types of errors account for 51% of OCR errors. For correctness classification, a BERT type Transformer model yielded an F-score of 0.982 on fabricated data. On real invoice data, the initial model yielded an F-score of 0.596. After additional fine-tuning, the F-score was raised to 0.832. The results of this thesis show that ML, while not entirely reliable, may be a viable first step in assessment and correction of OCR errors for invoices.

## Keywords

Machine learning, Optical character recognition, BERT, Error detection, Invoice

# Sammanfattning

Stora företag tar emot och skickar ut stora volymer fakturor innehållande olika fält med transaktionsdetaljer. Dessa fält inkluderar skattesats, förfallodatum, totalbelopp, osv. Ett vanligt sätt att automatisera fakturahantering är optisk teckenläsning. Denna teknologi innebär automatisk läsning av tecken från inskannade bilder. Ett problem med fakturor är att det saknas standardmall. Detta försvårar hanteringen av inläst data från fakturor med olika gränssnitt. Denna uppsats söker utforska vanliga fel i utmatningen från Azure's Form Recognizer general document model och sätten på vilka maskininlärning kan användas för att lösa nämnda problem, genom att förse feldetektering som ett första steg genom att klassificera optisk teckenläsningsutmatning som korrekt eller inkorrekt. För att undersöka detta gjordes en analys av vanligt förkommande fel i teckenläsningsutdata från 70 verkliga fakturor, och en BERT-modell finjusterades för klassificering av fakturor. Resultaten visar att de två vanligast förekommande optiska teckenläsningsfelen är:$(i)$ att ovidkommande ord upptäcks i ett inläst värdefält och $(ii)$ avsaknaden av ord i ett värdefält, vilka svarar för 51% av de optiska teckenläsningsfelen. För korrekthetsklassificeringen användes Transformermodellen BERT vilket gav ett F-värde på 0.98 för fabrikerad data. För data från verkliga fakturor var F-värdet 0.596 för den ursprungliga modellen. Efter ytterligare finjustering hamnade F-värdet på 0.832. Resultaten i denna uppsats visar att maskininlärning, om än inte fullt tillförlitligt, är ett gångbart första steg vid bedömning och korrigering av optiska teckenläsningsfel.

## Nyckelord

Maskininläsning, Optisk teckenläsning, BERT, Feldetektering, Faktura

# Acknowledgments

We would like to thank Gerald Q. Maguire Jr at KTH and Helene Wahlqvist at Rillion for providing guidance during this thesis.

Stockholm, August 2022
Jacob Dwyer Sara Bertse

# Contents

**1 Introduction** **1**
1.1 Background . . . 1
1.2 Problem . . . 2
1.2.1 Original problem and definition . . . 2
1.2.2 Scientific and engineering issues . . . 2
1.3 Purpose . . . 3
1.4 Goals . . . 5
1.5 Research Methodology . . . 5
1.6 Delimitations . . . 6
1.7 Structure of the thesis . . . 6

**2 Background** **7**
2.1 Optical Character Recognition . . . 7
2.1.1 Post-Processing . . . 8
2.1.2 Azure's Form Recognizer: General Documents Model 9
2.2 Machine Learning . . . 9
2.2.1 Deep Learning . . . 10
2.2.2 Gradients and backpropagation . . . 12
2.2.3 Transformer models . . . 13
2.3 Related work . . . 13
2.3.1 ML for invoice error detection . . . 14
2.3.2 BERT . . . 14
2.3.3 BERT for text classification . . . 15
2.4 Summary . . . 16

**3 Method** **17**
3.1 Research Process . . . 17
3.2 Research Paradigm . . . 18
3.3 Data Collection . . . 19

3.3.1 Sample Size . . . . . 20
3.4 Experimental design . . . . . 20
3.4.1 Hardware and Software used . . . . . 21
3.5 Assessing reliability and validity of the data collected . . . . . 22
3.5.1 Validity of method . . . . . 22
3.5.2 Reliability of method . . . . . 23
3.5.3 Data validity . . . . . 23
3.5.4 Reliability of data . . . . . 24
3.5.5 Data Analysis Technique . . . . . 24
3.5.6 Software Tools . . . . . 24
3.6 Evaluation framework . . . . . 25

**4 Building a neural model** **27**
4.1 Data Generation Scripts . . . . . 27
4.1.1 Correctly Scanned Data . . . . . 28
4.1.2 Error Data . . . . . 28
4.2 BERT Model . . . . . 29
4.2.1 Data Tokenization . . . . . 29
4.2.2 Training and Validation . . . . . 30
4.2.3 Evaluation on Test Set . . . . . 31
4.2.4 Evaluation on Real Data and additional fine-tuning . . 31

**5 Results and Analysis** **33**
5.1 Major results . . . . . 33
5.1.1 OCR errors . . . . . 33
5.1.2 BERT . . . . . 36
5.1.2.1 Fabricated data . . . . . 36
5.1.2.2 Real invoice data . . . . . 38
5.1.2.3 Summary of scores . . . . . 39
5.2 Reliability Analysis . . . . . 41
5.3 Validity Analysis . . . . . 41

**6 Discussion** **43**

**7 Conclusions and Future work** **47**
7.1 Conclusions . . . . . 47
7.2 Limitations . . . . . 48
7.3 Future work . . . . . 48
7.3.1 Functionality yet to be implemented . . . . . 48
7.3.1.1 International invoices . . . . . 48

7.3.1.2 Additional key-value fields . . . . . 49
7.3.1.3 Post-processing . . . . . 49
7.3.1.4 Integration pipeline . . . . . 49
7.4 Reflections . . . . . 49

**References** **51**

# List of Figures

1.1 Two alternative models for combining error detection and error correction . . . . . 4

2.1 A neural network which transforms three numerical inputs into five numerical outputs . . . . . 11

3.1 The process of key-value extraction from an invoice . . . . . 19

5.1 The distribution of the frequency of errors by key-value pairs, regardless of error type . . . . . 34
5.2 The distribution of the frequency of errors by type across all key-value pairs . . . . . 35
5.3 The progression of training versus validation loss throughout the epochs . . . . . 37
5.4 The MCC for each of the batches . . . . . 38
5.5 The training loss, validation loss and validation accuracy for each of the two epochs . . . . . 39
5.6 The Matthews Correlation Coefficient (MCC) scores before and after additional fine-tuning when testing with real invoice data . . . . . 40

# List of Tables

3.1 The software and corresponding versions used . . . . . 21

5.1 The distribution of errors by key type . . . . . 34
5.2 The distribution of error types, independent of in which key-value pair the error occurs . . . . . 35
5.3 The training loss, validation loss and validation accuracy for each of the four epochs . . . . . 37
5.4 The training loss, validation loss and validation accuracy for each of the two epochs . . . . . 38
5.5 A summary of the F-scores and MCC for the models and type of testing data . . . . . 39

# List of acronyms and abbreviations

| | |
|---|---|
| BERT | Bidirectional Encoder Representations from Transformers |
| DSM | design science methodology |
| GDM | general document model |
| GLUE | General Language Understanding Evaluation |
| GPU | graphical process unit |
| JSON | JavaScript Object Notation |
| MCC | Matthews Correlation Coefficient |
| ML | machine learning |
| MLM | masked language model |
| NLP | natural language processing |
| NSP | next sentence prediction |
| OCR | Optical Character Recognition |
| PDF | portable document format |
| RNN | recurrent neural network |
| SDG | Sustainable Development Goal |
| SDK | software development kit |
| UN | United Nations |
| VAT | Value Added Tax |

# Chapter 1

# Introduction

This thesis aims to examine whether the use of machine learning (ML) is a viable alternative for error detection as a first step in solving the challenge of interpreting Optical Character Recognition (OCR) invoice contents *independent* of layout variation.

## 1.1 Background

Large corporations receive and send large volumes of invoices containing various fields detailing a transaction. Such fields include Value Added Tax (VAT), due date, total amount, *etc*. One common way to automatize invoice processing is OCR. This technology entails reading handwritten or electronic texts and converting them into data. An example of this is processing forms in portable document format (PDF) format, read through OCR, and often passed to some application for further processing.

Various recognition errors can occur when extracting data from documents using OCR [1]. Alphanumeric values can be misinterpreted, for example the letter 'O' can falsely be identified as the number '0' and *vice versa*. To mitigate this problem, post-processing techniques are often implemented. According to a survey of OCR post-processing approaches [1], these techniques can be divided into error detection and error correction. Which techniques to use depends on the type of text being read. The survey outlines two groups: one main group includes isolated word approaches, where a misspelled word is changed into the likeliest candidate of its correct form without taking context into account, while another group is context-based approaches which, unlike isolated word approaches, also takes context into account.

A challenge when scanning invoices with OCR is that there is no universal standard layout format for an invoice. This creates difficulties when processing data from invoices with different layouts. A possible approach to solving this entails ML. In ML, a predictive model that analyzes patterns in a large set of data is trained. The model then makes generalizations based on these patterns and can generalize to similar but unseen material [2]. An example of ML can be found in Azure's Form Recognizer, where ML can be used to train models to extract key-value pairs and to recognize different invoice layouts [3]. Another example of ML is a model classifying invoices into correctly OCR-processed versus incorrectly OCR-processed, which can be used for error detection in invoices [4].

## 1.2 Problem

Despite the aim for automatic processing of invoices, the use of manual processing remains a necessity, largely due to OCR errors. Particularly with invoices, where a single alphanumeric error could lead to the invoice being processed incorrectly, which might mean an economic loss for the company involved. The higher the error rate from OCR is, the more extensive and rigorous the manual correction needs to be. For that reason, even a small improvement in the accuracy of OCR-reading provides a large benefit. A first step in error detection may be the classification of invoice OCR output as either correct or incorrect. When reframed into a binary classification problem, can ML facilitate the detection of errors for invoices of varying layouts?

### 1.2.1 Original problem and definition

> Is it possible for a deep neural model to classify OCR-read invoices as correctly or incorrectly read with an accuracy of 90%?

### 1.2.2 Scientific and engineering issues

OCR is a broad field which can be used in many different areas and disciplines, including digitization of historical texts [1], receipts [5], and handwritten documents [6]. Because of this, many different post-processing techniques have been developed, making it difficult to find the most appropriate one to use in any given context. The same can be said for ML, which can be used for a multitude of purposes. The models can also be trained using different training

algorithms and with different hyperparameters. Additionally, the data set used for training the model affects the model's ability to generalize correctly [2].

Within the context of invoice processing, it is conceivable that ML can be used as a first step in OCR post-processing in the form of error detection, and finding the right ML model could improve the overall process of automatic data extraction from invoices.

## 1.3 Purpose

The purpose of this thesis is to provide further research into automatic invoice data extraction by use of OCR and ML. Companies handling invoices as well as other documents in need of digitization could benefit from it. The academic benefit from this is providing additional research in the fields of ML for detecting OCR errors.

Error detection goes hand in hand with error correction. Although this thesis focuses on **error detection** and not on error correction, the authors of this thesis propose two alternative theoretical models as abstractions for combining error detection with error correction. These models are illustrated in Figure 1.1.

The first model proposes that error correction be done after classification. OCR output is fed into a trained neural model, which classifies the output as correct or incorrect. If the output is classified as incorrect, post-processing in the form of error correction can be carried out. Optionally, the corrected output can be fed back into the neural model once again to see whether the error correction was sufficient or not. If there is no further error correction to be done and the post-processed output is still incorrect, at least the user of the system can be notified that the output may be incorrect, in which case it can be manually corrected.

The second model proposes that error correction be done before classification. After the output is corrected, a trained neural model classifies the output as correct or incorrect. If the output is still incorrect, the user can be notified and the output can be manually corrected.

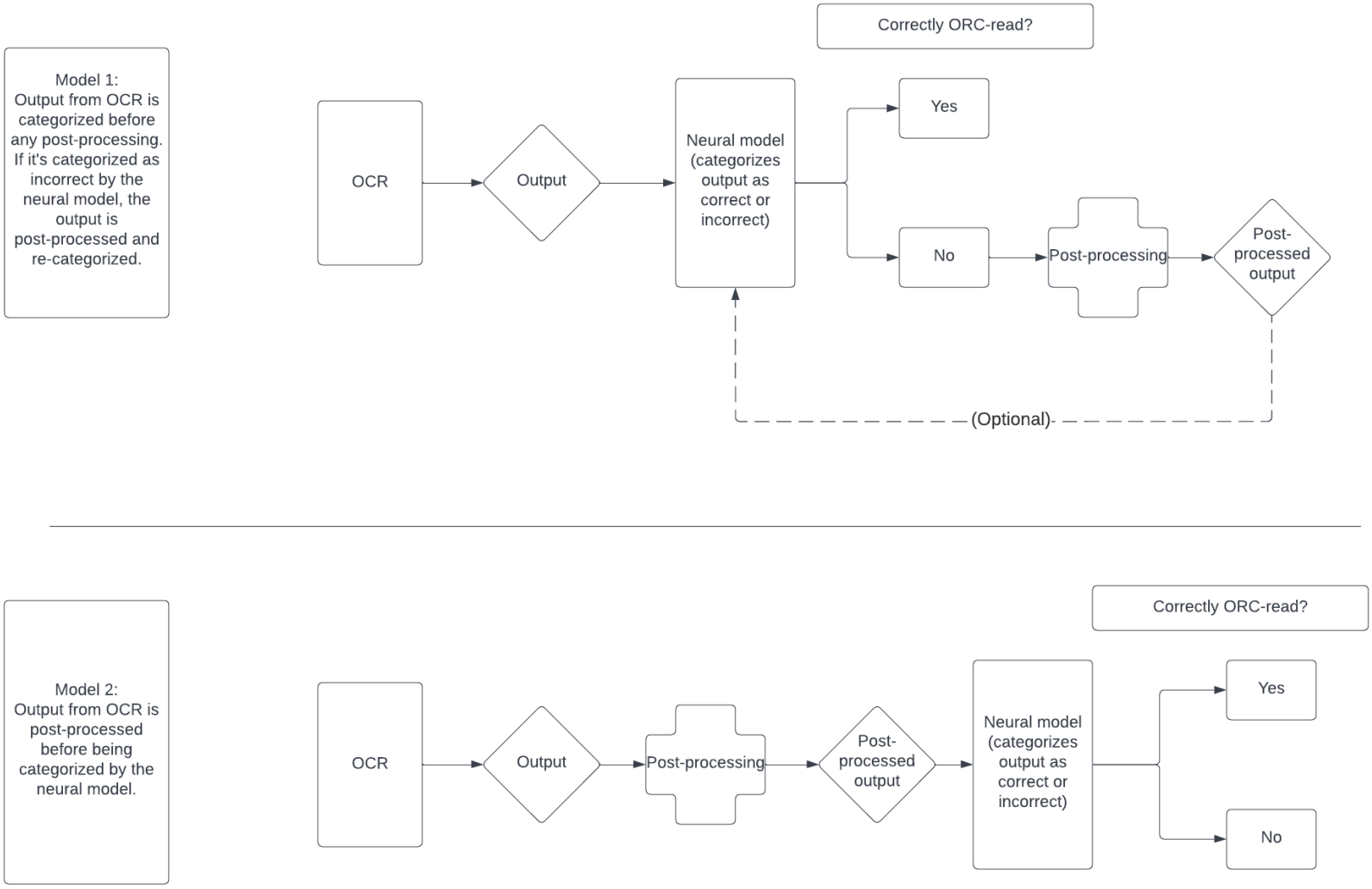


Figure 1.1: Two alternative models for combining error detection and error correction

By examining the accuracy of classification, the authors hope to find an indication of which theoretical model best fits this context (*i.e.,* OCR'd invoices).

The thesis also aims to contribute to the United Nations (UN) Sustainable Development Goals (SDGs) [7]. The specific goals related to this thesis are

**goal 3** good health and well-being,

**goal 8** decent work and economic growth,

**goal 9** industry, innovation and infrastructure,

**goal 13** climate action, and

**goal 15** life on land.

Any work pertaining to digitalization may have an impact on both the environment and socio-economic growth. A discussion of how this thesis may impact these goals can be found in Section 7.4.

## 1.4 Goals

The goal of this project is to create an ML model which can accurately differentiate between correctly and incorrectly scanned invoices of different layouts. Within this, there are two subgoals:

**SG 1** Determine which types of OCR problems are the most common with Azure's Form Recognizer general document model (GDM) when it comes to invoices of different layouts, and

**SG 2** Construct a model which reaches an overall accuracy of 90% for classification.

## 1.5 Research Methodology

The design science methodology (DSM) will be used for this project [8]. This methodology follows three phases: the problem identification phase, in which a relevant research problem is identified; the solution design phase, where further examination of the problem may lead to a redefinition of the original problem [8]. The outcome of the solution design phase is an artifact (*i.e.,* a tool, model, or technique) [8]. Lastly, the evaluation phase, in which the

research hypothesis is refined if needed and methods such as case studies and laboratory experiments may be applied [8]. Other research methodologies that were considered but ultimately rejected were action research and technology transfer model.

## 1.6 Delimitations

The delimitations for this thesis are as follows:

- The OCR itself will not be implemented from scratch but will rather be provided by Azure, including extraction of key-value pairs and entity recognition,
- Although Azure provides an OCR extraction model for invoices, the GDM was chosen instead for host company reasons,
- The invoices analyzed will be of American and Canadian origin,
- One deep learning model will be looked at, specifically Bidirectional Encoder Representations from Transformers (BERT), and
- Invoice OCR output data analyzed will be limited to key-value pairs, within which the most common or significant keys will be chosen.

## 1.7 Structure of the thesis

Following this introduction, Chapter 2 gives the reader background information during which relevant technologies are introduced and related works are summarized. Chapter 3 discusses the method, including the research process and paradigm, data collection, and planned data analysis. Chapter 4 contains information about the practical implementation details. The results of the project are then presented in Chapter 5. In Chapter 6, the results are discussed in detail. The final chapter (Chapter 7) is dedicated to drawing conclusions about and reflecting on the thesis, suggestions for future work, and reflections on the UN's SDGs.

# Chapter 2

# Background

This chapter presents the technologies used during the thesis project as well as related works. Section 2.1 describes OCR and some post-processing techniques used with it, as well as Azure's Form Recognizer tool and its GDM, while Section 2.2 provides an introduction to ML. Section 2.3 provides descriptions of works related to the topic of this thesis, including results of previous research. Section 2.4 serves as a summary of this chapter.

## 2.1 Optical Character Recognition

Optical character recognition is a method for digitizing machine printed or handwritten text from scanned documents and interpreting its patterns as characters [6]. The two core components in OCR algorithms are feature extraction and classification [6]. In the feature extraction stage, an attempt to recover pattern attributes (character protrusions, number of concavities in its outer contour, corners, *etc*.) characteristic of each character class is made and compared to members of a sample set to determine if they are a possible match [6].

Different approaches can be taken during the classification stage [6]. In statistical classification, the patterns of the scanned character are represented as points in a multidimensional feature space, which consists of measurements and feature values [6]. The feature space is then partitioned into regions associated with each class [6]. Template matching is a method in which individual pixels are used as features, and the placement of black pixels and white pixels in the scanned character are compared to those in a template class [6]. Then, the template with the maximum number of agreements (black pixels and white pixels in the template matching those of the scanned

character) or minimum number of disagreements (black pixels and white pixels in the template *not* matching those of the read character) can be chosen as the most probable character candidate [6]. Another option is the structural approach, in which character pattern images are represented as primitives and relations between these primitives [6]. An example is the letter 'H': when tasked with describing its structure, one could say that it consists of two vertical lines, with one horizontal line connecting them at their midpoint.

### 2.1.1 Post-Processing

When a document has been interpreted by OCR, there are usually mistakes in the interpreted text [1]. The field of OCR post-processing deals with correcting any such mistakes, and there is a plethora of post-processing correction methods [1].

OCR post-processing generally consists of error detection as well as error correction [1]. Within error correction, there is candidate generation and then candidate ranking [1]. Error detection is an important part of post-processing, as errors cannot be corrected if they are not first detected [1]. For example, one type of OCR error that can be targeted in error correction is segmentation errors [1]; "sub way" instead of "subway". Another type of error to target can be recognition errors [1]; "hme" instead of "home".

Candidate generation and candidate ranking are about finding appropriate candidates to replace erroneous words with [9]. For example, the word "hme" could be "home", but it could also be "hem". The candidates must then be ranked so that the best match can be found. The type of ranking and what the results of that ranking are depends on the algorithm used.

Post-processing can also be performed through the use of a lexical approach. This entails finding or creating a lexicon of a finite number of entries, with the OCR'd text then being compared to those entries and corrected according to the nearest match, as was done in [10] with post-processing of Swedish addresses. This technique can be used when it is possible to compile a limited and finite lexicon set.

Another type of post-processing is what Kamaleson *et al.,* [11] refer to as key information extraction. Many types of documents, including invoices, greatly benefit from key information extraction in the form of logically paired extracted text fields, for example the text "shipping address" and the actual address, the text "total price" and the numerical value of the price, and so on. However, while this key information extraction is of great benefit, there are new challenges that come with it, such as mislabeled key-value pairs or

incorrectly extracted values.

### 2.1.2 Azure's Form Recognizer: General Documents Model

Azure's Form Recognizer is a tool by Microsoft for OCR extraction. It provides multiple models targeting different domains, such as personal ID, receipts, and invoices, as well as the option of training a custom model. Additionally, there is also the general document model (GDM), which is the the model used in this thesis.

The GDM extracts tables, entities, text, and structure from the document by attempting to find key-value pairs [12]. The **key** in a key-value pair is a span of text within the document, used as an identifier, whereas the **value** constitutes the value associated with each key. An example of a key in a form document could be the text span "Date:", and the value associated with that key would then be the filled in date. However, this presumes that the date has been filled in and that the date is visible to the OCR; if neither of these conditions are met, then the key has no value [12].

Running in tandem with the extraction of key-value pairs is the named entity recognition model which identifies entities such as locations, dates, and organizations [12]. This way, keys or values which may not be found by the key-value pair extraction model may still be found and identified among the entities.

The GDM is pre-trained, meaning it does not require labelling or training; unlike, for example, the custom model. The GDM supports structured, unstructured, and semi-structured documents [12].

## 2.2 Machine Learning

Acccording to Zhi-Hua Zhou, machine learning (ML) is a technique for improving system performance by learning through experience using computational methods, which is done through building models from data using learning algorithms [2]. These models are used for predicting the outcome of new observations. ML makes use of collections of data known as data sets to conduct training, whereas individual data in the set are known as samples, which describe the attributes, or features, of an object or event.

Additionally, he says that training of a model is done using a training set containing data known as training data. Learning problems can be

roughly divided into two different categories: supervised learning for labelled training data, and unsupervised learning for unlabelled training data. Making predictions using a learned model is called testing, with the samples to be predicted being known as testing samples.

### 2.2.1 Deep Learning

A deep learning network contains what are called neurons, a term derived from neuroscience [13]. In deep learning, a neuron takes a numerical input and produces a numerical output [13]. This numerical output represents whether the neuron is activated or not [2]. The numerical value of the output is obtained through an activation function, for example the Sigmoid function, which converts output values to numbers between 0 (zero) and 1 (one) [13]. A zero represents a non-activated state while one represents an activated state, and values in between can be said to represent a partial activation [2]. This can be thought of as the strength of a neuron's activation. Used together with a loss function, these values can be interpreted as confidence scores indicating the confidence with which the model classifies the input as belonging to a certain category [13].

A classic example for illustration is a neural network which reads a handwritten digit and interprets the value. At the end of such a network, there are neurons to represent the numbers 0-9, one neuron for each number. Imagine a sloppily written '7', which can be read as a '1': the network might be "unsure" of whether the number indeed is a '7' or a '1'. The neuron representing '1' might have the activation value of 0.65 (an arbitrary value used for this example) whereas the neuron representing '7' might have value 0.80. These can be interpreted as probabilities,*i.e.,* there is a 65% probability that the number is a '1', and an 80% probability that the number is a '7'. However, for numbers not similar to a '7', for example an '8', the neuron representing '8' might output the value '0', meaning there is 0% probability of the number being an '8'.

Neurons are grouped together according to layers (see Figure 2.1), with the first layer known as the input layer, followed by hidden layers, which are followed by the output layer at the end [13]. Neural networks with more than two hidden layers are referred to as deep neural networks, hence the term deep learning [13]. Depending on the layer, the neurons in a layer are connected to neurons in either the following layer, the previous layer, or both [13]. The output of a neuron in one layer may be fed as input to one or more neurons in the following layer [13]. For fully connected layers, the neurons in the input

and hidden layers have as many outgoing values as the number of neurons in the upcoming layer, while the neurons in the output layer have none [13]. Excluding the input layer, each layer can be viewed as a vector resulting from a linear transformation of the values from the previous layer [13]. These transformations are influenced by the weights and biases associated with the layer in question [13].

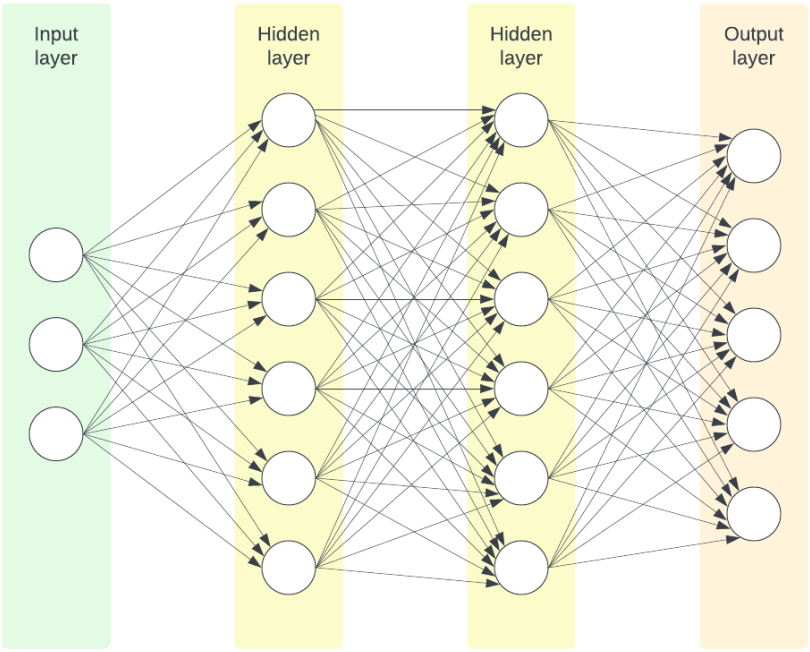


Figure 2.1: A neural network which transforms three numerical inputs into five numerical outputs. Note that the neurons in each layer are connected to each neuron in the next layer; the layers are *fully connected*.

Since a deep neural network consists of multiple layers, previous layers generate an output and feed it into the next layer [13]. But not every output or input fed into the next layer holds the same importance [2]. Consider the many factors that go into any prediction. Some of these factors may matter more for prediction, such as how sunny it is on a summer day when predicting the number of visitors at a beach, while others may matter much less, for example whether there are parasols to use at the beach or not. The importance of each factor is then the weight for how strongly that specific factor influences the prediction. In the same way, every connection in a neural network has a

weight associated with it, signifying the strength of the connection between two neurons [13].

When an ML model trains, the task in the learning is to find the correct weight values [13]. Because each neuron is connected to multiple other neurons, there is a huge number of weights and setting them manually would be a monumental task [13]. This is overcome by supervised training, as it leads to the model adjusting and re-adjusting the weights based on feedback until the correct values are found [13].

### 2.2.2 Gradients and backpropagation

To find correct values for the weights, an algorithm for gradient descent is used [14]. Though there are multiple algorithms pertaining to gradient descent, the concept is the same for all of them [14]. The goal is to minimize loss, which is a measure of how far away the prediction is from the actual value [14]. In order to minimize loss, a local or global minimum is searched for in a given function (the function itself depends on the data, the model, the goal, *etc*.) [14]. By following the gradient, a value will get closer and closer to the minimum of the function [14].

In practice, this is how the adjustment of the weights happens [14]. Some of the weights may only need to be adjusted a little, indicating that a point close to the minimum has been found, while others may need to be adjusted a lot, making the gradient steeper [14]. Going back to the example with the model interpreting numbers, if the number 8 is interpreted as the number 1, the gradient is very steep, because the weights need to be adjusted a lot since the prediction is very far off. If, however, an 8 is interpreted as a 9, it is partially correct; there is a loop on the top for both of them. If, in a very simplified example, one neuron is responsible for the top part of the number and another neuron is responsible for the bottom part of the number, it can be seen that the top of the number is interpreted correctly, and the bottom of the number is interpreted incorrectly. Therefore, the top interpretation does not need to be adjusted, while the bottom one does. The gradient for the top neuron is thus very small, while the gradient for the bottom neuron is quite large.

Backpropagation is when the gradient values are propagated backwards throughout the model, essentially acting as feedback to previous layers [15]. This is the key to how a model learns. Because each connection to each neuron has its own gradient, the weights are adjusted as a sum of the gradients of the connected neurons, based on the backpropagated feedback. Without backpropagation, ML would not be possible in the way that it works today.

### 2.2.3 Transformer models

Recurrent neural networks (RNNs) are neural networks which are able to consider the structures of the properties in their inputs and as such are able to recognize patterns in sentences of arbitrary lengths [13]. In terms of natural language processing (NLP) use cases, RNNs allow for prediction of the next word in a sentence based on all the previous words in the sentence [13].

While RNNs have been considered to lead to "breakthroughs in language modeling" [13], RNNs are not able to parallelize their operations as they are sequential, putting limits on their performances based on memory constraints [16]. The Transformer model architecture was created to avoid the drawbacks inherent in the RNN architecture by abandoning the recurrence and convolution mechanisms inherent to RNN in favor of an attention mechanism, called self-attention or intra-attention [16].

The attention mechanism in a Transformer model is known as multi-head attention [16]. This mechanism allows the attention function, which enables modelling of dependencies regardless of the distances between them in the input and output sequences, to be performed in parallel across different attention layers [16]. This attention is used in three different ways: in the encoder-decoder attention layers, the encoder, and the decoder [16].

The Transformer model follows an encoder-decoder structure; the encoder maps an input sequence to a sequence of continuous representations which are then handed to the decoder, which generates an output sequence of symbols [16]. The model is auto-regressive at each step, meaning that previously generated symbols are used as additional input when generating the next symbol [16].

One example of a Transformer model is Bidirectional Encoder Representations from Transformers (BERT). While earlier language model fine-tuning approaches were unidirectional [17], BERT makes use of bidirectional pre-training, combining left and right contexts [17].

## 2.3 Related work

This section presents scientific works which relate to the topic of this thesis. Section 2.3.1 summarizes a thesis about invoice error detection using ML while Section 2.3.2 discusses two works on BERT, including its performance compared to other systems and parameters for fine-tuning BERT. Lastly, Section 2.3.3 provides a short summary on three works about BERT used in classification tasks.

### 2.3.1 ML for invoice error detection

One suggested way to increase the efficiency of invoice processing is to try to detect errors in OCR-read invoices, which is the topic of Larsson and Segerås' [4] Bachelor's thesis in computer engineering. Although OCR post-processing articles often focus on word errors and segmentation errors in words, error detection in general need not only pertain to misspelled words. As part of their thesis, Larsson and Segerås trained a neural model to recognize overall errors in invoices, using the Naive Bayes Algorithm. In their work, an invoice was classified as correct when the model was 99% certain of its correctness. Using two training sets with 75 lines of labeled invoice data in each, with three iterations of 25 invoice data in each, the prediction of the model increased with each iteration for correctly classified invoices and automatically processed invoices and decreased incorrectly classified invoices. Their training set size of 75 invoices in each of the two sets is relatively small, but there was a clear trend in the improvement of prediction with each training and data set.

### 2.3.2 BERT

BERT was introduced in [17]. In this paper the benefits of using bidirectional pre-training for natural language processing was presented. The pre-training of BERT is done with unlabeled data (*i.e.,* unsupervised learning), while fine-tuning BERT is done with labeled data (*i.e.,* supervised learning). During the unsupervised pre-training, BERT uses a masked language model (MLM) to mask 15% of words at random. Simply put, the model learns by predicting the masked words. In combination with this, BERT also learns by next sentence prediction (NSP). These two techniques, combined with bidirectional learning, enable the BERT model to take context into account. A recent study from 2022 [18] shows that BERT also takes word-order into account, even when the words in a sentence are identical but are presented in a different order.

Once a BERT model is pre-trained, it can be fine-tuned for a specific domain or problem set [17]. The original paper [17] examines and compares BERT with other context-based systems, specifically ELMo and OpenAI GPT. The BERT model outperformed both ELMo and GPT on the General Language Understanding Evaluation (GLUE) benchmarks. Two BERT models were tested: $\text{BERT}_{\text{base}}$ and $\text{BERT}_{\text{large}}$. The BERT models outperformed the previous state-of-the-art system OpenAI GPT by $4.5\,\%$ (for $\text{BERT}_{\text{base}}$) and $7.0\,\%$ (for $\text{BERT}_{\text{large}}$).

Chi Sun *et al.,* [19] examined different ways of both pre-training and fine-tuning a BERT model for language classification. Both the $\text{BERT}_{\text{base}}$ and $\text{BERT}_{\text{large}}$ models are pre-trained, and fine-tuning adapts them for a specific task. One of the recommended hyperparameters in [19] is a base learning rate of $2 \times 10^{-5}$. The learning rate for a model is how quickly the weights mentioned in Section 2.2.1 are updated [2]. Another recommendation in [19] is a training length of four epochs. During an epoch, the model performs a forward and a backward pass through a training set [14].

The fine-tuned BERT models in [19] were tested on five different datasets. The error rates ranged from $0.71\%$ to $30.06\%$ for the $\text{BERT}_{\text{base}}$ model, depending on which dataset was used. Compared with a non-fine-tuned BERT model, the improvement was, on average, $9.22\,\%$.

### 2.3.3 BERT for text classification

When it comes to text classification, BERT has been used in various fields. One such example is in [20], where BERT was used to classify tweets into the categories "disaster" and "not disaster". The BERT-model was fine-tuned using $10\,873$ data samples in the form of tweets (snippets of text) about disasters, with the labels "target" for disaster and "non-target" for non-disaster. The BERT model used a dropout rate of $0.5$ and the activation function of the classification layer was rectified linear activation function or ReLU. The validation accuracy overall was $79\,\%$, with an F-score of $0.84$ for "non-target" (non-disasters) and $0.75$ for "target" (disasters).

Another study on text classification using BERT is [21], where a BERT model was used for automatic detection of fake news. The news were classified into four classes. With the help of an imbalanced dataset, three BERT models were compared: one fine-tuned $\text{BERT}_{\text{base}}$ model, one fine-tuned BERT model with the classifying layer implementing weighted cross entropy, and one model with additional pre-training as well as implementing weighted cross entropy. The F-scores for the models were $0.656$, $0.734$, $0.746$, respectively.

Yet another study on text classification using BERT is [22], where a BERT model was used to classify tweets related to eating disorders. The learning rate of the model was $2 \times 10^{-5}$, the batch size was $32$ and the number of epochs was $15$. The F-scores across the different categories were $0.833$, $0.719$, $0.776$, and $0.86$.

## 2.4 Summary

This chapter outlined OCR, including post-processing, as well as ML, including neural networks, deep learning, and backpropagation. A related work [4] presented the idea of having an ML model classify an invoice as either correct or incorrect when looking across the entirety of the invoice, while other related works compared BERT to other systems, recommended fine-tuning parameters, and presented results related to classification problems using BERT. As stated in this chapter, error detection is an important part of error correction, thus separating correct invoices from incorrect invoices through classification is a first step toward correcting erroneous invoices.

# Chapter 3

# Method

This chapter outlines the methodology used in this thesis. Section 3.1 describes the research process. Section 3.2 describes the research paradigm. Section 3.3 describes the data collection process. Section 3.4 describes the design of the experiments. Section 3.5 entails an assessment of the reliability and validity of both input and output data, concerning both the method used to collect the data and the collected data itself, as well as descriptions of the data analysis technique and software tools used. Section 3.6 introduces the evaluation framework used to evaluate the results of this thesis.

## 3.1 Research Process

The research was divided into a research process and a development process. The research process was undertaken by conducting research on the topics related to the project, including ML and OCR. This was achieved by finding relevant papers and books on these subjects and by searching for research papers with premises similar to this thesis. Once the first stage of the research process finished, the development process started.

The development process consisted of writing Python code to execute scripts for extracting key-value pairs and entities from invoices, generating string representations of invoices being captured successfully, and generating string representations of invoices being captured unsuccessfully. The next part of the development process consisted of fine-tuning and testing neural network models that could accurately classify a given string representation as belonging to either a correctly or an incorrectly scanned invoice. The code for this step was also written in Python, albeit using the PyTorch library in Google's Colab environment [23].

After getting results from the development process, the research process continued by analysing these results and discussing the analysis in detail.

The process for the thesis project can be summarized in the following steps:

- Plan the project,
- Conduct necessary research,
- Collect data,
- Conduct experiment,
- Analyze results of the experiment, and
- Discuss the results of the analysis.

## 3.2 Research Paradigm

This thesis followed the design science paradigm (and methodology). Within the design science paradigm, the aim is to produce a practical design for a practical problem [24]. According to [24], the other research paradigms are formal sciences and explanatory sciences. An example of a formal science can be mathematics, which is not dependent on empirical observation or validation. An example of an explanatory science is natural science, which describes or explains different aspects of the natural world. In contrast to both of these, design sciences pertain to man-made inventions and the question of how to improve these.

According to Emelie Engström *et al.,* [24], design science research is carried out in a specific research context, wherein general problems are instantiated by looking at specific problems. The validation of the solution (artifacts, methods, *etc.*) should be carried out in a real-world context. Furthermore, they claim that understanding an instance of a real-world problem leads to understanding of the general problem and the potential solutions. Thus, as mentioned previously in Section 1.5, the problem identification phase in design science is an important part of understanding the general problem. But just conceptualizing the problem is not enough - the conceptualization needs to be framed by a solution, as conceptualizing the problem but not venturing further and exploring solutions fails to lead into the second and third phases of research design: the solution design phase and the evaluation phase.

## 3.3 Data Collection

A total of 70 real invoices were analyzed by extracting key-value pairs and entities, using Azure's AI Form Recognizer package from the Azure software development kit (SDK) For Python API [25]. The model used for OCR was the GDM described in Section 2.1.2. Some of the invoices were inspected with Microsoft's Form Recognizer Studio [26] on Azure's platform. The reason for using Azure's Form Recognizer Studio in combination with the script was to get a visual confirmation along with a visual idea of which fields the OCR read, which fields were labeled as key-value pairs, and which fields were labeled as entities. Figure 3.1 provides an overview of the process.

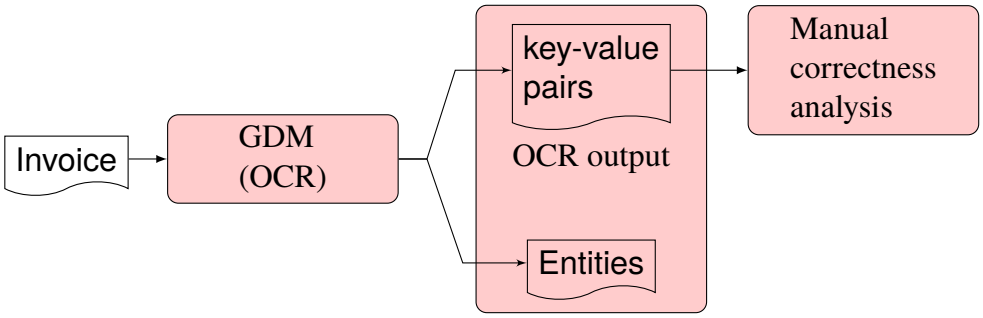


Figure 3.1: The process of key-value extraction from an invoice

An error analysis was performed to determine the most frequently occurring errors using the data analysis technique described in Section 3.5.5. Out of the 70 processed invoices, 67 were of unique invoice formats, with the remaining three being of duplicate invoice format.

Depending on the invoice format and the contents of the invoice, different key-value pairs were extracted. For example, some invoices had keys for unit price, number of units ordered, phone numbers, sub-totals, *etc*. Out of the key-value pairs, five keys were picked in order of frequency. As these keys can be regarded as sensitive information by the company hosting this project, these keys will be known as "Key A", "Key B", "Key C", "Key D", and "Key E".

As the analysis of invoices proved time-consuming (two weeks of full-time work), in order to fabricate larger volumes of data for model fine-tuning, addresses and company names were collected from two datasets, Fortune 1000 [27] and a JavaScript Object Notation (JSON) file based on data from a dataset by OpenAddresses [28, 29]. The former contains structured data concerning the 1000 American corporations with the highest revenues in 2021,

as published by American Business Magazine [27]. The OpenAddresses-based JSON file consists of 3220 real, public domain addresses in the United States that can be geocoded successfully [28, 29].

Using these datasets, Python scripts were written to generate strings representing key-value pairs read from scanned invoices. One script generated strings representing correct OCR output while the other script generated strings representing incorrect OCR output. From the previously mentioned analysis, the most frequent or serious errors formed the basis for a fabricated dataset.

### 3.3.1 Sample Size

As previous mentioned in Section 3.3, the sample size for invoice analysis was 70 invoices.

The sample size for model fine-tuning was 20 000 generated strings. This number was chosen because it provided a good balance between training time and needed accuracy, with a training time of 13 minutes and 38 seconds.

The sample size used for the final model assessment was 4000 generated strings. After testing with a set consisting of 200 samples, and then with 500 samples, and then finally with 4000 samples, the latter set provided the most accurate metric results. The chosen sample size was influenced by the string generation speed. A set of this size (4000 samples) could be generated quickly, requiring 0.012 seconds for each sample, while still providing the desired accuracy.

For testing with real invoice data, 42 invoices were used.

For the additional fine-tuning, an augmented dataset containing 4000 new samples was generated. As the model had already been fine-tuned once, 4000 was a small enough set to be effective without leading to noticeable over-fitting, as well as not demanding a large amount of time to generate (approximately 48.65 seconds).

## 3.4 Experimental design

To see whether a transformer model could differentiate between correct OCR output and incorrect OCR output, a $\text{BERT}_{\text{base}}$ model was fine-tuned. The performance of the model was evaluated in terms of the evaluation framework described in Section 3.6. All of the training sets mentioned in Section 3.3.1 were balanced, meaning there was an equal number of correct and incorrect samples. For the incorrect samples, each error corresponded to 10% of the

total number of incorrect samples, meaning there were 10 types of errors represented evenly in the incorrect samples.

The testing set consisting of OCR output from 42 real invoices was used for validity testing. As was expected, the model could improve further with additional fine-tuning; therefore, an augmented dataset was created and was also balanced. The same testing set with the 42 invoices was used to test the additionally fine-tuned model.

### 3.4.1 Hardware and Software used

The code for model construction and training was written in Python with extensive use of the PyTorch software library [30]. The Pandas data analysis library was used for creating dataframes in which to store data [31]. The `argmax` function from the NumPy library was also used [32]. Because of the extensive computing power needed to train the model, the code was executed using Google's Colab cloud computing interface [23]. The versions of the software used are shown in Table 3.1. Google provides computation resources for Colab in the form of graphical process units (GPUs)[33]. The GPUs used during the computation of the code were models belonging to NVIDIA's T4 and K80 series.

Table 3.1: The software and corresponding versions used

| **Software** | **Version** |
|---|---|
| Colab | Unspecified |
| NumPy | 1.21.6 |
| Pandas | 1.3.5 |
| Python | 3.7.13 (Colab),<br>3.10.4 (desktop) |
| PyTorch | 1.11.0+cu113 |
| Sci-kit Learn | 1.0.2 |
| Transformers from Hugging Face | 4.20.0 |

## 3.5 Assessing reliability and validity of the data collected

This section discusses the validity of the method in Section 3.5.1, the reliability of method in Section 3.5.2, as well data validity and reliability of data in Section 3.5.3 respectively Section 3.5.4. The data analysis technique is explained in Section 3.5.5. Finally, the software tools used for reliability and validity assessment are outlined in Section 3.5.6.

### 3.5.1 Validity of method

Design science research is a paradigm where evaluation is important. The authors in [34] describe four types of evaluations. The first type involves evaluation of the problem itself. Is the problem truly an important one to solve or is it a novel problem? Is it possible to solve the problem? The second type involves evaluation of the solution's design. Is the solution's design simple and consistent? The third type involves evaluating the solution instance for metrics such as ease of use and robustness. Fourth is the evaluation of the use of the solution: Is it effective and efficient?

A proposed framework for validity in design science research [35] is design antecedence validity, development and use context validity (and within this, artifact validity), and design outcome validity. Design antecedence validity pertains to whether the proposed solutions are congruent with current theory as well as the usefulness of the solution. In the case of this thesis, based on previous research, neural networks can be used in natural language processing and in classification problems. Furthermore, classifying OCR output as "correctly read" and "incorrectly read" pertains to error detection, an important step in error correction.

Development and use context validity pertains to data input and the reasonableness of data used in development of a solution [35]. Within this, artifact validity deals with consistency as well as asking whether the solution represents the reality in a relevant manner [35]. In this thesis, although the artifact is more a proof-of-concept, it is shown that the solution artifact (the model) can consistently classify invoices as correct or incorrect. Although this thesis looks at a limited number of key-value pairs from the OCR output, care has been taken to analyze and represent the actual errors that occur in the OCR output.

Design outcome validity pertains to the performance of the artifact [35]. Whether the artifact outperforms current systems or is on-par and whether the outcome measures up to (realistic) current expectations. In the case of the models in this thesis, the framework described in Section 3.6 is used to assess their performance, which is then compared to previous related work in the field.

### 3.5.2 Reliability of method

Reliability is, according to [36], about dependability and replication.

Because of how time-consuming analyzing invoices is, the limit was set at 70 invoices. The errors and distribution of errors found in those might change with a larger sample size.

In terms of training a model, there can be issues with reproducibility. When training a model, initial weights are randomized. To prevent bias during training, a random seed can be used.

Once a neural model is trained and fine-tuned, it will perform similarly and reliably when it comes to the same types of predictions. However, depending on which key-value pairs are considered, the reliability might change. In the case of this model, reliability has been measured by having the model give predictions for different (but similar) test data. Additionally, the amount of data for fine-tuning was large, which considerably raises the reliability. Care has also been taken to provide balanced test data.

### 3.5.3 Data validity

As data has been provided by the host company and with the analyzed invoices being real invoices, the validity is high. A vast majority of invoices were of unique formats which is in line with the thesis problem of classifying invoices of different formats.

The limitation in validity for training data is that the limited number of key-value fields narrows the scope of the research. In reality, invoice output contains more than the five fields considered in this thesis.

The validity of the model's predictions have the same limitations, as does the validity of the training data. The predictions made by the models are congruent with the training data, but in reality, there may be formats of the keys that have not been accounted for in the training set. Thus the models may classify correctly OCR'd output as incorrect, leading to false negatives.

### 3.5.4 Reliability of data

Although the invoices used have been carefully analyzed, and although the invoices had many different formats, there are countless other invoice formats that have not been tested. Different invoice formats, or even different invoices of the same format but with different content, may give different results in terms of the most common errors.

For the data provided for training, reliability is high as the data was generated by a script which always has the same format and the same generated types of errors.

Hyperparameters provided for training may affect the accuracy of predictions. The specific models developed in this thesis will give identical predictions with identical confidence scores each time the same prediction is run, but re-training the same model with other hyperparameters would give different values.

### 3.5.5 Data Analysis Technique

For the invoice output error analysis, invoice key-value pairs were extracted using Azure's Form Recognizer GDM. Key-value pairs were separated from entities.

All other key-value pairs than those chosen were removed. A scanned invoice was then compared to the OCR output and it was noted where in the selected key-value pairs errors occurred. Finally, the invoice output was manually labeled as "correct" or "incorrect" depending on whether errors were found or not.

The errors found were then manually classified into the error types and keys presented in Section 5.1.1. A sample could have more than one error, in which case each error was counted as a single error type.

### 3.5.6 Software Tools

The Scikit-learn software library was used for reliability and validity assessment[37].

## 3.6 Evaluation framework

When assessing how well a model can generalize after training, there are a number of evaluation metrics to be looked at. These metrics are error, accuracy, precision, recall, and F1 [2]. These concepts, as described in [2], are summarized in this section.

Error is a measure of proportionality between incorrectly classified samples and the total number of samples. Within the concept of error, there is division between: training error and generalization error. Generalization error is also referred to as validation error. In some contexts, depending on the loss function, training error is also called training loss and validation error is called validation loss. When training a model, part of the data set is the training set while another part is the validation set. The training set is for learning while the validation set is for validating said learning. Training loss and validation loss corresponds to these two types of training data. Thus, the generalization loss shows how well a model can generalize.

According to Zhi-Hua Zhou [2], a model that has a high training loss is said to be underfit, which means the model has not learned from the training set. In contrast, a model that has a low training loss but a high validation loss is said to be overfit, meaning the model has not learned to generalize from the training set. Overfitting is a common problem.

Other than incorrectly classified samples and correctly classified samples, there are also two other cases: false positives and false negatives. Precision and recall are metrics for those, and can be defined as:

$$\text{Precision} = \frac{\text{true positives}}{\text{true positives} + \text{false positives}}$$

$$\text{Recall} = \frac{\text{true positives}}{\text{true positives} + \text{false negatives}}$$

Precision and recall are usually at odds with each other. When a larger amount of samples are picked as positives, false positives increase. In contrast, when only samples with a high certainty are picked as positives, the amount of false negatives increases as there may be samples which are positive, even if the model isn't as certain about them.

Because of this, a metric called F1 is often used. F1 can be said to be the average (harmonic mean) of precision and recall and can be calculated with the following formula:

$$\text{F1} = \frac{2 \times \text{true positives}}{\text{total number of samples} + \text{true positives} - \text{true negatives}}$$

Besides the aforementioned metrics, another metric used specifically for binary classification models is the Matthews Correlation Coefficient (MCC), which measures the correlation between the true classes $c$ and the predicted labels $l$. The value of the MCC ranges from -1 (worst) to +1 (best) [38].

# Chapter 4

# Building a neural model

This chapter outlines how data was generated and how the neural model was fine-tuned using this data. Section 4.1 details the scripts for generating the data, while Section 4.2 explains how the BERT model was fine-tuned.

## 4.1 Data Generation Scripts

Reframing the problem as a text classification problem, similar to the related works mentioned in Section 2.3.3, extracted keys and values for the different invoice fields were merged into one cohesive string, as BERT takes both context and word-order into account.

Two full-time workdays were spent manually processing, cleaning up, and labelling output to use as training data until it was realized that the number of samples needed for fine-tuning would not be possible to acquire manually, as only 40 samples were produced during these two days. For this reason, the aforementioned data generation script from Section 3.3 was written based on the analysis of OCR errors.

One script was made to fabricate correctly processed data. This script is described in Section 4.1.1. While another script fabricates incorrectly processed data, as described in Section 4.1.2. Each script allowed to user to specify how many data samples, in form of strings and corresponding labels, would be generated.

### 4.1.1 Correctly Scanned Data

The script output was in the form of a JSON file containing the pre-determined number of samples. Each sample consisted of the attributes “text” and “label” containing, respectively, the parsed invoice data and a number indicating a correctly or incorrectly scanned invoice. A “label” attribute with a value of ’0’ specified that the invoice was incorrectly scanned, whereas a value of ’1’ specified the opposite. One complete example of a data entry labeled as correct, with keys and values replaced in order to not divulge sensitive information, is: `{"text": "Key A Value A Key B Value B Key C Value C Key D Value D Key E Value E", "label": 1}`. Each key could have various representations, as the essence of a key could be captured in different words. The values themselves could also be of different formats.

Randomization in key-value generation was attained through Python’s `random` library[39].

### 4.1.2 Error Data

A second script was written in order to fabricate incorrectly processed data. This script first creates or overwrites a JSON file with the name given in the script file, then generates as many samples as stated in the file, with a tenth of this number for each of the ten different chosen errors. These specific errors were:

- Missing values,
- Wrong value appearing in the wrong field,
- Spaces appearing where they should not appear,
- Extra words or characters appearing when they should not, and
- Strings of number mistakenly replacing correct values.

Each instance of the fabricated incorrectly processed data is represented as a string with only one of the possible errors appearing in each string representation. These strings are then treated as JSON elements in the same format as that shown in the earlier example for correctly processed data, with the only difference being that the “label” attribute has the value ’0’.

## 4.2 BERT Model

The type of BERT model used was `bert-base-uncased` [40] of the BertForSequenceClassification interface from Hugging Face, which constitutes a pre-trained BERT model with a sequence classification head on top [41]. The fact that the model is ”uncased” signifies that it does not differentiate between two words that are the same but are capitalized differently [40]. ”bert-base-uncased” was pre-trained with MLM and NSP objectives in mind, using four cloud-based tensor processing units with a batch size of $256$ over $1\,000\,000$ steps [40]. For $90$ % of the steps, the sequence length was limited to $128$ tokens while for the remaining number of steps, it was limited to $512$ tokens [40].

The Adam optimizer was also used during pre-training, with a learning rate of $10^{-4}$ and a weight decay of $0.01$ [40]. For the first $10\,000$ steps the pre-training had a learning warmup, while the steps that followed had linear decay of the learning rate [40]. The model was pre-trained on the BookCorpus dataset and English Wikipedia [40].

As mentioned in Section 2.3.2, a pre-trained BERT model can then be fine-tuned for a specific task. The BertForSequenceClassification model was fine-tuned using GPUs provided by Google through Colab, as mentioned in Section 3.4.1. The code for fine-tuning of the model was written in Python.

Much of the functionality used in the fine-tuning of the model was imported from the `transformers` package from Hugging Face. This made use of the PyTorch interface.

### 4.2.1 Data Tokenization

The JSON dataset was first loaded into the Colab notebook using the `load` method from the JSON library [42]. Then the dataset was split into a sentence collection and a label collection. The sentences were then split up into tokens and stored in a tensor using the method `encode_plus` from a pre-made BERT tokenizer for `bert-base-uncased` models [41]. Each sentence was split into a maximum of $64$ tokens; if less than $64$ tokens were needed to represent a sentence, then the remainder of the tensor was padded. An attention mask was also constructed for each sentence in order for BERT to pay attention to the non-padded tokens and to ignore the padded tokens. Then the special tokens ’[CLS]’ and ’[SEP]’ were added to the beginning and end of each tensor, respectively. The tokenized sentence and attention mask was then added to their respective lists before the loop started a new iteration. After the

loop had finished, the lists were converted to tensors.

The training dataset, as described in Section 3.4, was then divided into a training set containing 90 % of the data and a validation set containing the remaining 10 %. The testing set, described in the same section, was used to evaluate the model. The training and validation sets for additional fine-tuning are described in Section 3.4, with the same split of training and validation percentages being used. All sets were made iterable by converting them to the `DataLoader` class from the PyTorch library [30]. While the samples for the the training sets' dataloaders were chosen at random using the `RandomSampler` class [30], the samples for the validation set's dataloader were chosen in sequence using `SequentialSampler` [30]. Both dataloaders had a batch size of 32.

The optimizer used for fine-tuning of the model was `AdamW` from the Hugging Face library [43]. It had a learning rate of $2 \times 10^{-5}$, as recommended in previous research in Section 2.3.2, and an $\epsilon$ value of $1 \times 10^{-8}$ was used for stability [43]. A learning rate scheduler was then created using this optimizer. The number of training steps for it equalled the number of batches multiplied by the number of epochs; the number of epochs chosen was four, based on the recommendation mentioned in Section 2.3.2.

### 4.2.2 Training and Validation

The training and validation of the model was performed in a loop which iterated over the number of batches in the training dataloader for the selected number of epochs. For each iteration, gradients from previous calculations were cleared and a forward pass was performed, followed by a backward pass. Then the gradient norm was clipped in order to limit the range of the gradient, followed by an update of the learning rate as dictated by the AdamW optimizer [43], which was followed by a step taken using the calculated gradient. Following the termination of the loop, the average loss for the training set was calculated by dividing the total loss by the length of the training dataloader.

Once the training steps of the epoch were completed, a loop iterated over each element in the validation dataloader. During each iteration, the input IDs, attention masks, and labels of the current batch were loaded onto the GPU for computation, and a forward pass was made, but no backward pass; backpropagation was only used during training. As such, no gradient calculations were needed during the validation step. Lastly, the average accuracy and loss for the validation run were calculated.

### 4.2.3 Evaluation on Test Set

The test data was prepared for tokenization in the same way as the training data, with a test dataloader created using a `SequentialSampler` [30] and a `TensorDataset` [30]. For each batch in the test dataloader, the model made a forward pass, calculating the prediction on the current test sentence and appending the result to a list, while the true labels were appended to another list.

The MCC of the test data was evaluated using the `matthews_corrcoef` method. The F1-score was calculated using the `f1_score` method. Both of these methods were imported from the SciKit-learn library [37].

For the MCC, all predictions calculated in the training stage were iterated over. The index containing the highest value was chosen using the NumPy method `argmax` [32] and compared with the true label of the corresponding prediction using `matthews_corrcoef`. The return value was then appended to the set of MCC calculations. The total MCC was calculated by comparing the true labels with the prediction results across all batches, also using `matthews_corrcoef`.

### 4.2.4 Evaluation on Real Data and additional fine-tuning

From the real invoice data, 21 samples of correct output and 21 samples of incorrect output were used to analyze the validity and accuracy of the model. F-scores and MCC were calculated for these 42 samples using the initial fine-tuned model.

Incorrectly classified invoice data was further analyzed for the purpose of additional fine-tuning of the model. Three new key representations were added to the augmented training set, which consisted of 4000 data samples as described in Section 3.4. The additional fine-tuning was done with the same hyperparameters, with the only difference being the length of the training, which was 2 epochs, so as to prevent overfitting. F-scores and MCC were then calculated for the additionally fine-tuned model.

# Chapter 5

# Results and Analysis

This chapter presents the results of the thesis. Section 5.1 presents the major results related to OCR errors as well as results related to the fine-tuned BERT models. Section 5.2 analyzes the validity of the results while Section 5.3 analyzes the reliability of the results.

## 5.1 Major results

The results of the thesis are presented in this section. Section 5.1.1 presents results, charts, and tables related to the types of errors which occur during OCR extraction through the GDM. Section 5.1.2 presents results related to the fine-tuned BERT models.

### 5.1.1 OCR errors

Out of the 70 scanned and analyzed invoices, 28 had errors (40 %). Some of the invoices had more than one error and the total number of errors was 49. Of those with an error, 14 of the invoices had one error, eight had two errors, five had three errors, and one had four errors (an average of 1.75 errors per incorrectly scanned invoice, looking at only the five selected key-value pairs).

The distribution of errors by key type, *i.e.,* the key-value pairs most frequently containing errors, independent of the type of error, is presented in Table 5.1, along with a visual representation in Figure 5.1.

Table 5.1: The distribution of errors by key type

| **Key** | **Prevalence** % |
|---|---|
| Key C | 34.7 |
| Key A | 24.5 |
| Key D | 20.4 |
| Key B | 14.3 |
| Key E | 6.1 |

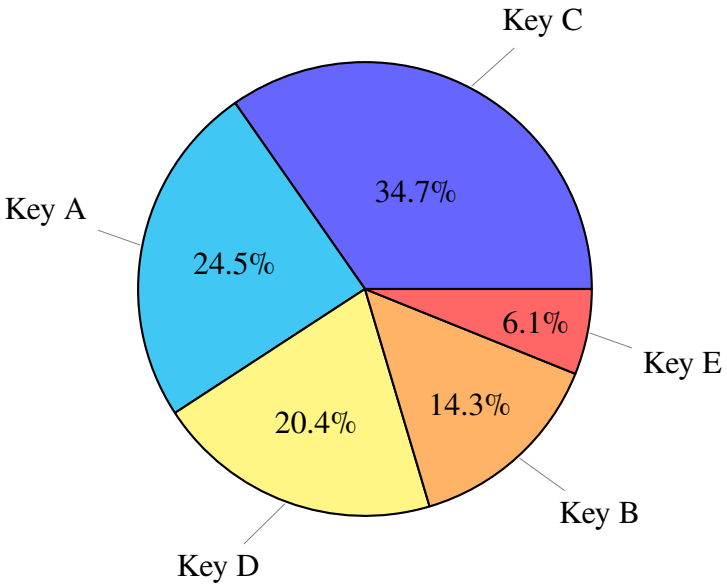


Figure 5.1: The distribution of the frequency of errors by key-value pairs, regardless of error type

As can be seen, the key-value pair in which errors most frequently occur was the pair with “Key C”, containing more than a third of all errors (34.7 %). This is followed by the key-value pair belonging to “Key A”, where almost a quarter of all errors occur (25.5 %). The key-value pair least prone to errors was “Key E”, with 6.1 %.

Next, the distribution of errors by error type (for example “key is missing value”), independent of in which key-value pair this error occurred, is presented in Table 5.2, along with a visual representation in Figure 5.2.

Table 5.2: The distribution of error types, independent of in which key-value pair the error occurs

| **Type of error** | **Prevalence** % |
|---|---|
| extra words/symbols/spaces | 26.5 |
| parts of value missing | 24.5 |
| key-value pair missing | 14.3 |
| key is missing value | 10.2 |
| value listed under wrong key | 8.2 |
| key-value pair divided into two | 8.2 |
| key-value pair mashed up | 4.1 |
| part of value registered as key | 2.0 |
| part of key missing | 2.0 |

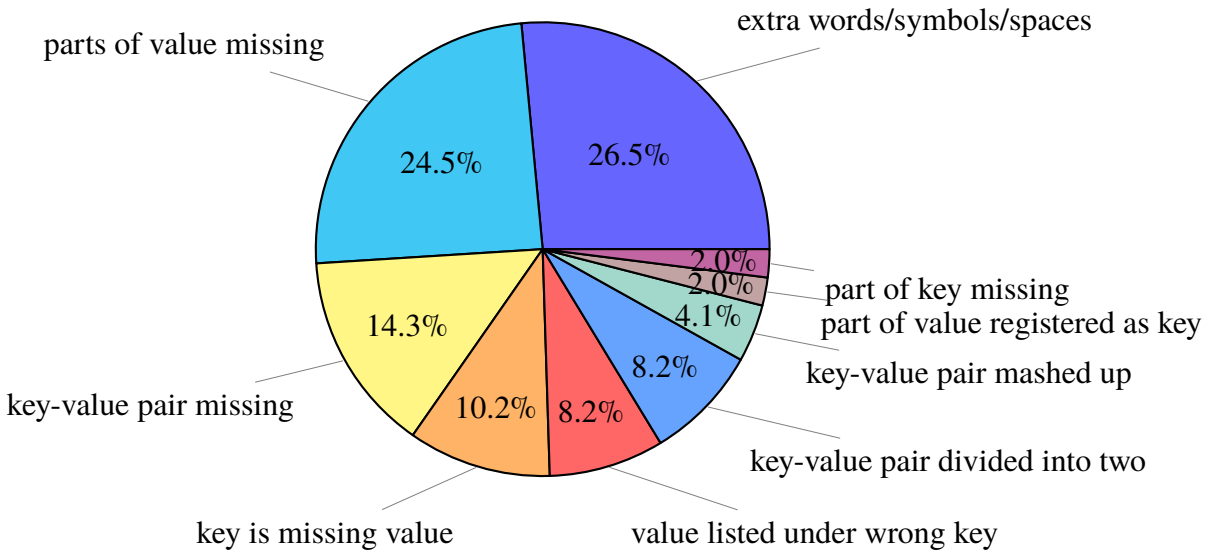


Figure 5.2: The distribution of the frequency of errors by type across all key-value pairs

The most frequent errors were “extra characters/symbols/spaces” (26.5 %) and “parts of value missing” (24.5 %), making up more than half of all the error types (51 %). 14.3% of the errors pertained not to the actual values, but rather to the keys themselves, with the three types being “key-value pair divided into two” (8.2 %), “key-value pair mashed up” (4.1 %), and “part of value registered as key” (2.0 %).

Below are some examples of the types of errors found. Entirely missing values and entirely missing key-value strings are excluded for obvious reasons. The error type is stated next to each example.

- Part of the value for one key ends up in the value for another key (“value listed under the wrong key”),
- Values incorrectly consist of a string of numbers (“parts of value missing”),
- Values have extra words, for example additional notes or numbers belonging to other fields of the invoice (“extra words/symbols/spaces”), and
- Keys which should contain numeric values contain non-numeric values (“value listed under the wrong key”).

### 5.1.2 BERT

This subsection presents results related to the BERT models. In Section 5.1.2.1, the model is tested using fabricated data to assess validation accuracy, loss, F-scores, and MCC. In Section 5.1.2.2, the initial fine-tuned model as well as the additionally fine-tuned model are tested using real invoice data. Finally, a summary of the F-scores and MCC are presented in Section 5.1.2.3 for the two models and the two different testing sets.

#### 5.1.2.1 Fabricated data

For each of the epochs, the training loss, validation loss, and validation accuracy is shown in Table 5.3. Training and validation loss throughout the epochs is also displayed in Figure 5.3.

Figure 5.3: The progression of training versus validation loss throughout the epochs

Table 5.3: The training loss, validation loss and validation accuracy for each of the four epochs

| **Epoch** | **Training loss** | **Validation loss** | **Validation accuracy** |
|---|---|---|---|
| 1 | 0.239 | 0.057 | 0.987 |
| 2 | 0.064 | 0.047 | 0.988 |
| 3 | 0.055 | 0.043 | 0.989 |
| 4 | 0.050 | 0.047 | 0.988 |

The MCC on $4000$ random samples of fabricated data was $0.971$. This is displayed in Figure 5.4 below, which shows the MCC score of each batch.

The F-score for both classifications (correct and incorrect) was $0.982$, meaning the overall F-score was also $0.982$.

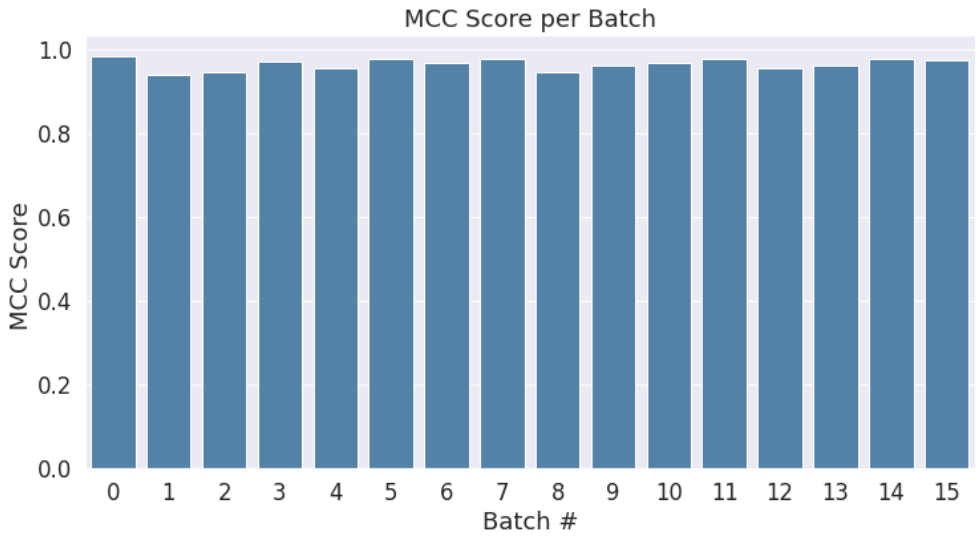


Figure 5.4: The MCC for each of the batches

#### 5.1.2.2 Real invoice data

Classification of data from 42 real invoices, made by the BERT model fine-tuned on 20 000 samples, resulted in F-scores of 0.692 for invoices classified as "incorrect", and 0.5 for invoices classified as "correct", with an overall F-score of 0.596. The total MCC score was 0.271.

The model was then further fine-tuned using an additional 4000 augmented samples. Table 5.4 shows the training loss, validation loss, and validation accuracy for each epoch in the additional fine-tuning. Figure 5.5 shows the visual representation of loss through the two epochs.

Table 5.4: The training loss, validation loss and validation accuracy for each of the two epochs

| Epoch | Training loss | Validation loss | Validation accuracy |
|---|---|---|---|
| 1 | 0.097 | 0.110 | 0.964 |
| 2 | 0.056 | 0.105 | 0.966 |

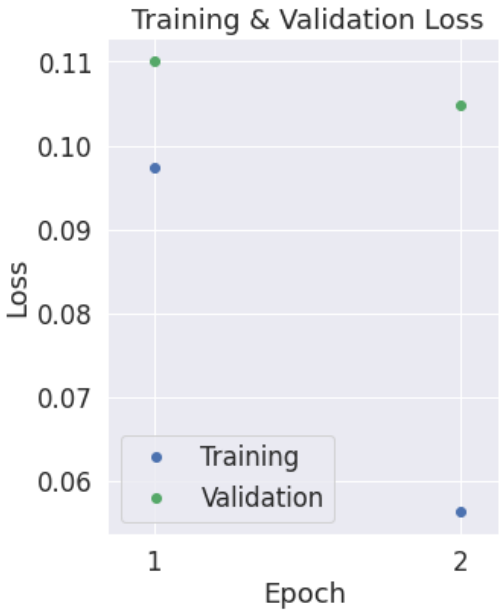


Figure 5.5: The training loss, validation loss and validation accuracy for each of the two epochs

After the additional fine-tuning, the F-score for invoices classified as "incorrect" was 0.821, and 0.844 for invoices classified as "correct". The overall F-score was 0.832. The MCC was 0.674.

#### 5.1.2.3 Summary of scores

Table 5.5 summarizes the results across the two BERT models and the two different testing sets.

Table 5.5: A summary of the F-scores and MCC for the models and type of testing data

| | **Initial model** | | **Additionally fine-tuned model** |
|---|---|---|---|
| **Score** | **fabricated data** | **real invoice data** | |
| F-score for "incorrect" | 0.982 | 0.692 | 0.821 |
| F-score for "correct" | 0.982 | 0.5 | 0.844 |
| Overall F-score | 0.982 | 0.596 | 0.832 |
| MCC | 0.971 | 0.271 | 0.674 |

As can be seen from Table 5.5, the initial model tested on fabricated data scores the highest. However, the fabricated data contained the exact same errors as the training set and is not necessarily representative of all possible errors that occur in OCR output from real invoices. Instead, the results when using real invoice data for testing provide better validity when it comes to correctness classification of real invoices. Figure 5.6 displays MCC scores for the two models tested with real invoice data.

Between the initial model and the additionally fine-tuned model, the additionally fine-tuned model performed better. The difference between the initial fine-tuned model and the model with additional fine-tuning, in terms of F-scores, was $0.129$ for the "incorrect" category and $0.344$ for the "correct" category. For the overall F-score, the difference was $0.236$. The difference in MCC was $0.399$.

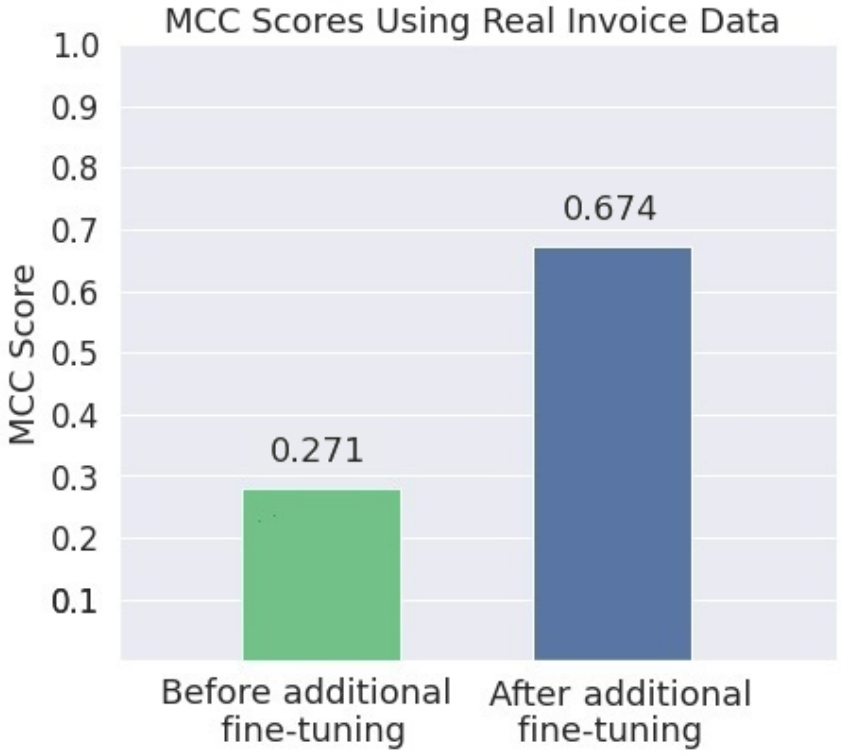


Figure 5.6: The MCC scores before and after additional fine-tuning when testing with real invoice data

## 5.2 Reliability Analysis

The OCR error results are based on the 70 invoices of which 67 were of unique formats and three were duplicate formats. It is possible that new error types may appear which are not represented here. Especially vulnerable to this is the "Key E" key-value pair, which only showed up three times in the errors by key type analysis. Another limitation is that the analyzed invoices were all invoices from the United States and Canada. Invoices in other languages would most likely provide different results.

When it comes to the model predictions, the consistency depends not only on the architecture of the model but on the training data used as well. Another factor are the initial weights of the model during training.

## 5.3 Validity Analysis

The distribution of errors analysis is limited to errors that occur in the GDM. However, it is conceivable that similar errors with similar distributions may occur with other models or software that provide OCR and key-value pair extraction. As mentioned in Section 5.2, the invoices analyzed were from the United States and Canada and thus the results most likely cannot be generalized to invoices of other languages. Additionally, the model has limited validity because only five fields were examined.

As a proof-of-concept, the results show that a fine-tuned BERT can differentiate between correct and incorrect OCR output.

# Chapter 6

# Discussion

The two goals of this thesis were to determine the most common OCR errors with Azure's GDM, and to construct a model which reaches an accuracy of $90\,\%$ for classification.

The results showed that errors frequently occur and that multiple errors in one OCR output is not uncommon. Errors occurred in all five chosen key-value pairs and pertained to not only values extracted, but to *keys* found during OCR extraction as well.

These results apply to the GDM, which was used for extracting key-value pairs from digital invoice documents. Azure's Form Recognizer provides other models, to which these results may not be generalizable.

For fine-tuning the BERT model, representative OCR errors were selected based on how often the key appeared in invoices as well as on the importance and relevancy to invoice processing software. This meant that less frequently occurring keys were not accounted for, which could be non-representative for invoices as a whole. When it comes to the importance of fields, the "Key E" field was correctly read in the vast majority of cases; only in three of $49$ error cases was it read incorrectly. Even so, given how frequently "Key E" occurred, inclusion of the field in the training set for the model was warranted. One complicating factor was that different invoices used terms which could have ambiguous meaning, in some cases referring to the same entity whereas in other cases these fields refer to two different entities.

For the BERT model, the F-score for the fabricated data was 0.982, with an MCC of $0.971$. However, when tested on real invoice data, the F-score was initially $0.596$, with an MCC of $0.271$. One of the main explanations for this was that the real invoices contained keys which the model had not been fine-tuned for.

After additional fine-tuning, the new F-score was $0.832$, with an MCC of $0.674$. This is in line with previous research, where F-scores in text classification problems ranged from $0.65$ to $0.86$, with an average F-score of $0.77$ and a median F-score of $0.75$.

The difference between the initial model and the additionally fine-tuned model on real invoice data shows that a BERT model can be fine-tuned further in order to improve accuracy. An implication and use for this could be semi-automated customer feedback, where any invoice processing software can gather data pertaining to fields which tend to be correct and fields which tend to be incorrect. This data can then be used in further fine-tuning.

One major challenge in classifying invoice output is that sometimes even manual detection of errors can be difficult, unless the output is compared to the original invoice. In one of the false positive cases, an extra line on the scanned invoice caused the OCR to misinterpret it as the character 'I'. It was then incorrectly appended, together with a blank space, to a value field. Unless the original invoice was manually reviewed, catching this as an error may be hard or even impossible, even for a human.

In other cases, the incorrect output came from mashed-up keys and values. These are still sequentially correct; for example, a fictitious example could be the key "Contact person [First Name]" with the [Last Name] as the value, when the correct version should be "Contact person" as the key and "[First Name] + [Last Name]" as the value. The order of the words is identical even though the key is technically incorrect, but this is irrelevant to the current implementation of the model, making key-value errors very hard to detect. This may not necessarily be a drawback but it does further complicate the problem of error detection in key-value pairs.

In yet other cases, the errors were existent but not severe enough for the model to detect them. These are errors in the form of extra characters, for example an extra period or dash before or in the value, although in some of the cases, the model did detect those errors as well.

In one of the false positive cases, a value was registered twice. This was not classified as incorrect by the model when it should have been classified as incorrect.

Finally, in another of the false positive cases, the model failed to detect that a value was missing. In another case, an entire key-value pair was missing, which the model failed to detect.

This means that when it comes to the two theoretical models of error correction in combination with error detection as presented in Section 1.3, the second alternative (post-processing and correcting errors before classification) seems to be the most viable option – as the model is not entirely reliable. Especially in the case of false positives, where incorrect output would be classified as correct, the post-processing and error correction would need to take place before classification.

# Chapter 7

# Conclusions and Future work

This final chapter presents the conclusions of this thesis in Section 7.1. Limitations of this thesis are discussed in Section 7.2. Ideas for future work are presented in Section 7.3. Finally, reflections on this thesis as well as the relationship between this thesis and the UN's SDGs are presented in Section 7.4.

## 7.1 Conclusions

As can be seen from the results of this thesis, error occurrence in OCR is a common problem. In the GDM, multiple invoice outputs had several combinations of errors. Therefore, error detection should be a vital part of OCR post-processing.

While this thesis shows that it is possible for a model to achieve an accuracy of $90\,\%$ or above, as was done with the fabricated data in this thesis, this kind of result may not be possible for every problem. In the end, the additionally fine-tuned BERT model achieved an accuracy of $83\,\%$ when evaluated on real invoice data. Whether or not this specific model could achieve an accuracy of at least 90% with a more targeted fine-tuning, or through use of different invoice data, is uncertain.

Using this project as a case study of ML, it is obvious that ML is a useful tool for assessing and classifying documents. However, when using ML for classification, it is important to be mindful of the fact that ML models rarely are completely accurate. For this reason, there may be benefits to post-processing OCR output even when the output is classified as correct by an ML model.

## 7.2 Limitations

The research in this thesis was limited by the following:

- There exist many types of transformer models, but only the BERT variant was used in this thesis. Training models of different types and comparing their resulting assessment metrics may be an interesting topic for further examination,
- The analyzed invoices were North American, resulting in only English-language examples of key-value pairs and common errors,
- Key-value pairs were extracted from $70$ invoices, a relatively small number which might not provide a representative view of invoices,
- Only $42$ invoices were used as samples for validity testing and as groundwork for further fine-tuning of the model, and
- Only five possible key-value pairs were chosen for analysis. Although they were chosen based on how representative they were for the examined invoices, many common invoice fields were left out.

## 7.3 Future work

Future research on using ML models to classify invoices could be conducted by comparing different types of transformer models, such as GPT-2 or GPT-3. The scope could also be expanded by having the model handle more than North American invoices and more key-value pairs.

### 7.3.1 Functionality yet to be implemented

This subsection briefly introduces various functionalities that are not yet implemented but that might make useful additions.

#### 7.3.1.1 International invoices

The current ML model classifies North American invoices successfully, but classification of invoices from other regions remains to be implemented.

#### 7.3.1.2 Additional key-value fields

The current ML model looks at five fields. The next step in expanding the model would be to add additional fields.

#### 7.3.1.3 Post-processing

For post-processing, four potential solutions are suggested:

- When it comes to post-processing OCR output, a suggestion for using the GDM could be to complement key-value fields with entity fields in order to extract additional information,
- Output can in some cases be cleaned up when it comes to additional characters or spaces. An example of this might be a value getting assigned to the incorrect key, in which case it may be possible to re-assign it to the correct key,
- Since the values in fields such as addresses and company names may assume a limited number of correct options, it may be possible to use a lexical approach in these cases, and
- For numerical values, checksums can in some cases be used to verify amounts. Whether BERT is capable of such tasks is best left for another study, but as BERT is mainly used for NLP, there may be other types of models better suited for this specific task.

#### 7.3.1.4 Integration pipeline

Building a pipeline to integrate this solution into other software would be of great benefit. In the later stages, user feedback about correct or incorrect invoices, and particularly the different fields in the invoices, could be used as additional data for this or other future neural models to provide greater accuracy.

## 7.4 Reflections

Although the authors of this thesis were previously familiar with the concept of ML, there was a lack of previous experience with it. When constructing the fabricated datasets, it was not clear from the start to the authors that the casing of letters did not matter when using `bert-base-uncased`. However, this

was not of concern; although it led to certain keys appearing more often than others, but it did not affect the overall accuracy of the model's predictions. In some cases, certain types of values appearing more often in the dataset was simulated on purpose as it was representative of values found in real invoices. Nonetheless, the model learned to recognize these values, as intended, despite the purposely simulated imbalance.

One of the challenges of the thesis was how time-consuming it was to analyze and process data. It was something the authors had underestimated and something which the authors will keep in mind in future projects.

This thesis aimed to contribute to the UN's SDGs[7]. The five goals to which this thesis was most relevant were goal 3, goal 8, goal 9, goal 13, and goal 15.

For goal 3, digitalization can lead to a better working environment as workers working within invoice processing can be freed up for other tasks. On the other hand, there is no guarantee that other tasks are less monotonous or more rewarding.

For goal 8, the automation of invoice processing may contribute to overall economic growth, as companies can handle larger volumes of invoices in less time and with more convenience, increasing the efficiency of businesses. On the other hand, for individuals working within manual invoice processing, advancement in the field of automatic invoice processing may lead to unemployment. However, as the model performance is not without flaws in regards to accuracy and scope, the prospect of unemployment might not be immediately threatening. Nonetheless, as technology improves, unemployment in certain fields may increase.

For goal 9, the digitalization of invoice processing may lead to an increase of startups and small-scale businesses as fewer employees are needed for invoice processing. This may create new job opportunities in other fields. The same goal also aims to "upgrade the technological capabilities of industrial sectors in all countries" [44]. As this thesis examined the technological capabilities of ML, NLP, and transformers, it is in line with this goal.

Finally, goal 9 also encompasses "clean and environmentally sound technologies" [44]. Other goals related to environment and climate are goal 13 and goal 15. Improving OCR post-processing may lead to a reduction in the use of envelopes, signifying a reduction of paper waste. A decrease in the transport of documents also lessens the environmental impact as digitalization increases.

# References

[1] T. T. H. Nguyen, A. Jatowt, M. Coustaty, and A. Doucet, "Survey of post-OCR processing approaches," *ACM computing surveys*, vol. 54, no. 6, pp. 1–37, 2021. doi: 10.1145/3453476 [Pages 1, 2, and 8.]

[2] Z.-H. Zhou, *Machine Learning*. Springer Singapore, 2021. ISBN 978-981-15-1967-3. [Online]. Available: https://doi.org/10.1007%2F978-981-15-1967-3 [Pages 2, 3, 9, 10, 11, 15, and 25.]

[3] Microsoft Corporation. (2022) Azure form recognizer documentation. [Online]. Available: https://docs.microsoft.com/en-us/azure/applied-ai-services/form-recognizer/?branch=release-build-cogserv-forms-recognizer [Page 2.]

[4] A. Larsson and T. Segerås, "Automated invoice handling with machine learning and OCR," Bachelor's thesis, KTH, Computer and Electronic Engineering, 2016. [Pages 2, 14, and 16.]

[5] K. M. Yindumathi, S. S. Chaudhari, and R. Aparna, "Structured data extraction using machine learning from image of unstructured bills/invoices," in *Smart Computing Techniques and Applications*, S. C. Satapathy, V. Bhateja, M. N. Favorskaya, and T. Adilakshmi, Eds. Singapore: Springer Singapore, 2021. ISBN 978-981-16-1502-3 pp. 129–140. [Page 2.]

[6] R. Kasturi, L. O'Gorman, and V. Govindaraju, "Document image analysis: A primer," *Sadhana (Bangalore)*, vol. 27, no. 1, pp. 3–22, 2002. doi: 10.1007/BF02703309 [Pages 2, 7, and 8.]

[7] United Nations Development Programme. (2022) Sustainable development goals | united nations development programme. [Online]. Available: https://www.undp.org/sustainable-development-goals [Pages 5 and 50.]

[8] C. Wohlin and P. Runeson, "Guiding the selection of research methodology in industry–academia collaboration in software engineering," *Information and Software Technology*, vol. 140, p. 106678, 2021. doi: 10.1016/j.infsof.2021.106678. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0950584921001361 [Pages 5 and 6.]

[9] K. Hosseini, F. Nanni, and M. Coll Ardanuy, "Deezymatch: A flexible deep learning approach to fuzzy string matching," in *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. Online: Association for Computational Linguistics, oct 2020. doi: 10.18653/v1/2020.emnlp-demos.9 pp. 62–69. [Online]. Available: https://aclanthology.org/2020.emnlp-demos.9 [Page 8.]

[10] M. Andersson, "Post-processing of optical character recognition for swedish addresses," Master's thesis, KTH, School of Electrical Engineering and Computer Science (EECS), 2022, TRITA-EECS-EX-2022:201. [Page 8.]

[11] N. Kamaleson, D. Chu, and F. E. B. Otero, "Automatic information extraction from electronic documents using machine learning," in *Artificial Intelligence XXXVIII*, ser. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2021, pp. 183–194. ISBN 9783030910990 [Page 8.]

[12] Microsoft Corporation. (2022, Jun.) Form recognizer general document model (preview). [Online]. Available: https://docs.microsoft.com/en-us/azure/applied-ai-services/form-recognizer/concept-general-document [Page 9.]

[13] Y. Goldberg, *Neural network methods in natural language processing*, 1st ed., ser. Synthesis lectures on human language technologies. Morgan & Claypool, 2017. ISBN 1-62705-295-X [Pages 10, 11, 12, and 13.]

[14] T. M. Arif, *Introduction to deep learning for engineers using Python and Google Cloud Platform*, ser. Synthesis lectures on mechanical engineering ; Number 28. Morgan & Claypool Publishers, 2020. ISBN 1-68173-914-3 [Pages 12 and 15.]

[15] A. Nandy and M. Biswas, *Neural Networks in Unity C# Programming for Windows 10*, 1st ed. Apress, 2018. ISBN 1-4842-3673-4 [Page 12.]

[16] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds., vol. 30. Curran Associates, Inc., 2017. doi: https://doi.org/10.48550/arXiv.1706.03762. [Online]. Available: https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf [Page 13.]

[17] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: pre-training of deep bidirectional transformers for language understanding," *CoRR*, vol. abs/1810.04805, 2018. doi: 10.48550/arXiv.1810.04805. [Online]. Available: http://arxiv.org/abs/1810.04805 [Pages 13 and 14.]

[18] I. Papadimitriou, R. Futrell, and K. Mahowald, "When classifying grammatical role, bert doesn't care about word order... except when it matters," *arXiv*, 2022. doi: 10.48550/arXiv.2203.06204 [Page 14.]

[19] C. Sun, X. Qiu, Y. Xu, and X. Huang, "How to fine-tune BERT for text classification?" in *Chinese Computational Linguistics*, ser. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2019, pp. 194–206. ISBN 3030323803 [Page 15.]

[20] A. K. Ningsih and A. I. Hadiana, "Disaster tweets classification in disaster response using Bidirectional Encoder Representations from Transformer (BERT)," *IOP conference series. Materials Science and Engineering*, vol. 1115, no. 1, p. 12032, 2021. doi: 10.1088/1757-899X/1115/1/012032 [Page 15.]

[21] H. Jwa, D. Oh, K. Park, J. M. Kang, and H. Lim, "exBAKE: Automatic fake news detection model based on Bidirectional Encoder Representations from Transformers (BERT)," *Applied sciences*, vol. 9, no. 19, p. 4062, 2019. doi: 10.3390/app9194062 [Page 15.]

[22] J. A. Benítez-Andrades, J.-M. Alija-Perez, M.-E. Vidal, R. Pastor-Vargas, and M. T. García-Ordas, "Traditional machine learning models and Bidirectional Encoder Representations From Transformer (BERT)-based automatic classification of tweets about eating disorders: Algorithm development and validation study," *JMIR medical informatics*, vol. 10, no. 2, pp. e34 492–e34 492, 2022. doi: 10.2196/34492 [Page 15.]

[23] Google LLC. (2022) Welcome to colaboratory. [Online]. Available: https://colab.research.google.com [Pages 17 and 21.]

[24] E. Engström, M.-A. Storey, P. Runeson, M. Höst, and M. T. Baldassarre, "How software engineering research aligns with design science: a review," *Empirical software engineering : an international journal*, vol. 25, no. 4, pp. 2630–2660, 2020. doi: 10.1007/s10664-020-09818-7 [Page 18.]

[25] Microsoft Corporation. (2022) Python API Browser | Microsoft Docs. [Online]. Available: https://docs.microsoft.com/en-us/python/api/?view=azure-python [Page 19.]

[26] ——. (2022) FormRecognizerStudio. [Online]. Available: https://formrecognizer.appliedai.azure.com/studio [Page 19.]

[27] Kaggle. (2021) Fortune 1000. [Online]. Available: https://www.kaggle.com/datasets/winston56/fortune-500-data-2021 [Pages 19 and 20.]

[28] Ethan Brown, Jason Long, Dashiell Sublett. (2018) Github - ethanrbrown/rrad: Real, random address data (rrad). [Online]. Available: https://github.com/EthanRBrown/rrad [Pages 19 and 20.]

[29] OpenAddresses. (2022) The free and open global address collection. [Online]. Available: https://openaddresses.io/ [Pages 19 and 20.]

[30] PyTorch Contributors. (2022) Library API. [Online]. Available: https://pytorch.org/cppdocs/api/library_root.html [Pages 21, 30, and 31.]

[31] pandas development team. (2022) Api reference — pandas 1.4.2 documentation. [Online]. Available: https://pandas.pydata.org/docs/reference/index.html#api [Page 21.]

[32] NumPy Developers. (2022) numpy.argmax. [Online]. Available: https://numpy.org/doc/stable/reference/generated/numpy.argmax.html [Pages 21 and 31.]

[33] Google. (2022) Frequently asked questions. [Online]. Available: https://research.google.com/colaboratory/faq.html [Page 21.]

[34] J. vom Brocke, A. Hevner, and A. Maedche, Eds., *Design science research : cases*, 1st ed., ser. Progress in IS. Springer Cham, 2020. ISBN 3-030-46781-3 [Page 22.]

[35] S. Hofmann, O. M. 0001, and M. Rossi, Eds., *Designing for Digital Transformation. Co-Creating Services with Citizens and Industry - 15th International Conference on Design Science Research in Information Systems and Technology, DESRIST 2020, Kristiansand, Norway, December 2-4, 2020, Proceedings*, ser. Lecture Notes in Computer Science, vol. 12388. Springer, 2020. doi: 10.1007/978-3-030-64823-7. [Online]. Available: https://doi.org/10.1007/978-3-030-64823-7 [Pages 22 and 23.]

[36] R. L. Baskerville, M. Kaul, and V. C. Storey, "Genres of inquiry in design-science research: Justification and evaluation of knowledge production," *MIS quarterly*, vol. 39, no. 3, pp. 541–564, 2015. doi: 10.25300/MISQ/2015/39.3.02. [Online]. Available: https://www.jstor.org/stable/26629620 [Page 23.]

[37] scikit-learn developers. (2022) API reference. [Online]. Available: https://scikit-learn.org/stable/modules/classes.html [Pages 24 and 31.]

[38] D. Chicco, N. Tötsch, and G. Jurman, "The matthews correlation coefficient (MCC) is more reliable than balanced accuracy, bookmaker informedness, and markedness in two-class confusion matrix evaluation," *BioData mining*, vol. 14, no. 1, pp. 13–13, 2021. doi: 10.1186/s13040-021-00244-z [Page 26.]

[39] Python Software Foundation. (2022) random — generate pseudo-random numbers. [Online]. Available: https://docs.python.org/3/library/random.html [Page 28.]

[40] Hugging Face. (2018) bert-base-uncased. [Online]. Available: https://huggingface.co/bert-base-uncased [Page 29.]

[41] ——. BertForSequenceClassification. [Online]. Available: https://huggingface.co/docs/transformers/v4.18.0/en/model_doc/bert#transformers.BertForSequenceClassification [Page 29.]

[42] Python Software Foundation. (2022) json — JSON encoder and decoder. [Online]. Available: https://docs.python.org/3/library/json.html [Page 29.]

[43] Hugging Face. (2022) Adamw (pytorch). [Online]. Available: https://huggingface.co/docs/transformers/v4.19.4/en/main_classes/optimizer_schedules#transformers.AdamW [Page 30.]

[44] United Nations Development Programme. (2022) Industry, innovation and infrastructure. [Online]. Available: https://www.undp.org/sustainable-development-goals#industry-innovation-and-infrastructure [Page 50.]

TRITA-EECS-EX-2022:606

# For DIVA


{
"Author1": { "Last name": "Dwyer",
"First name": "Jacob",
"Local User Id": "u1ge88qq",
"E-mail": "dwyer@kth.se",
"organisation": {"L1": "School of Electrical Engineering and Computer Science",
}
},
"Author2": { "Last name": "Bertse",
"First name": "Sara",
"Local User Id": "u17gsc8q",
"E-mail": "bertse@kth@kth.se",
"organisation": {"L1": "School of Electrical Engineering and Computer Science",
}
},
"Cycle": "1",
"Course code": "II142X",
"Credits": "15.0",
"Degree1": {"Educational program": "Degree Programme in Computer Engineering"
,"programcode": "TIDAB"
,"Degree": "Higher Education Diploma"
,"subjectArea": "Technology"
},
"Title": {
"Main title": "Exploring Machine Learning Solutions in the Context of OCR Post-Processing of Invoices",
"Language": "eng" },
"Alternative title": {
"Main title": "Utforskning av Maskininlärningslösningar för Optisk Teckenläsningsefterbehandling av Fakturor",
"Language": "swe"
},
"Supervisor1": { "Last name": "Montelius",
"First name": "Johan",
"Local User Id": "u1bx6gxe",
"E-mail": "johanmon@kth.se",
"organisation": {"L1": "School of Electrical Engineering and Computer Science",
"L2": "Computer Science" }
},
"Supervisor2": { "Last name": "Wahlqvist",
"First name": "Helene",
"E-mail": "helene.wahlqvist@rillion.com",
"Other organisation": "Rillion AB"
},
"Examiner1": { "Last name": "Maguire Jr",
"First name": "Gerald Quentin",
"Local User Id": "u1d13i2c",
"E-mail": "maguire@kth.se",
"organisation": {"L1": "School of Electrical Engineering and Computer Science",
"L2": "Computer Science" }
},
"Cooperation": { "Partner_name": "Rillion AB"},
"National Subject Categories": "10201, 10206",
"Other information": {"Year": "2022", "Number of pages": "xv,57"},
"Series": { "Title of series": "TRITA-EECS-EX" , "No. in series": "2022:00" },
"Opponents": { "Name": "Axel Forshällen"},
"Presentation": { "Date": "2022-06-28 13:00"
,"Language":"eng"
,"Room": "via Zoom https://kth-se.zoom.us/j/64904954871"
,Address": Isafjordsgatan 22 (Kistagången 16)"
,"City": "Stockholm" },
"Number of lang instances": "2",
"Abstract[eng ]": €€€€



Large corporations receive and send large volumes of invoices containing various fields detailing a transaction. Such fields include VAT, due date, total amount, etc. One common way to automatize invoice processing is optical character recognition (OCR). This technology entails automatic reading of characters from scanned images. One problem with invoices is that there is no universal layout standard. This creates difficulties when processing data from invoices with different layouts. This thesis aims to examine common errors in the output from Azure's Form Recognizer general document model and the ways in which machine learning (ML) can be used to solve the aforementioned problem, by providing error detection as a first step when classifying OCR output as correct or incorrect. To examine this, an analysis of common errors was made based on OCR output from 70 real invoices, and a Bidirectional Encoder Representations from Transformers (BERT) model was fine-tuned for invoice classification. The results show that the two most common OCR errors are: \first extra words showing up in a field and \Second words missing from a field. Together these two types of errors account for 51\% of OCR errors. For correctness classification, a BERT type Transformer model yielded an F-score of 0.982 on fabricated data. On real invoice data, the initial model yielded an F-score of 0.596. After additional fine-tuning, the F-score was raised to 0.832. The results of this thesis show that

ML, while not entirely reliable, may be a viable first step in assessment and correction of OCR errors for invoices.

€€€€,
"Keywords[eng ]": €€€€
Machine learning, Optical character recognition, BERT, Error detection, Invoice €€€€,
"Abstract[swe ]": €€€€

Stora företag tar emot och skickar ut stora volymer fakturor innehållande olika fält med transaktionsdetaljer. Dessa fält inkluderar skattesats, förfallodatum, totalbelopp, osv. Ett vanligt sätt att automatisera fakturahantering är optisk teckenläsning. Denna teknologi innebär automatisk läsning av tecken från inskannade bilder. Ett problem med fakturor är att det saknas standardmall. Detta försvårar hanteringen av inläst data från fakturor med olika gränssnitt. Denna uppsats söker utforska vanliga fel i utmatningen från Azure's Form Recognizer general document model och sätten på vilka maskininlärning kan användas för att lösa nämnda problem, genom att förse feldetektering som ett första steg genom att klassificera optisk teckenläsningsutmatning som korrekt eller inkorrekt. För att undersöka detta gjordes en analys av vanligt förkommande fel i teckenläsningsutdata från 70 verkliga fakturor, och en BERT-modell finjusterades för klassificering av fakturor. Resultaten visar att de två vanligast förekommande optiska teckenläsningsfelen är:\first att ovidkommande ord upptäcks i ett inläst värdefält och \Second avsaknaden av ord i ett värdefält, vilka svarar för 51\% av de optiska teckenläsningsfelen. För korrekthetsklassificeringen användes Transformermodellen BERT vilket gav ett F-värde på 0.98 för fabrikerad data. För data från verkliga fakturor var F-värdet 0.596 för den ursprungliga modellen. Efter ytterligare finjustering hamnade F-värdet på 0.832. Resultaten i denna uppsats visar att maskininlärning, om än inte fullt tillförlitligt, är ett gångbart första steg vid bedömning och korrigering av optiska teckenläsningsfel.

€€€€,
"Keywords[swe ]": €€€€
Maskininläsning, Optisk teckenläsning, BERT, Feldetektering, Faktura €€€€,
}