Semi-supervised Invoice Information Extraction

A Regional Object Detection and Rule-Based Approach

Master Thesis Business Analytics
Vrije Universiteit Amsterdam
Faculty of Science

Author:
Tessa Helmer

The Netherlands
July 30, 2021

MASTER THESIS BUSINESS ANALYTICS
VRIJE UNIVERSITEIT AMSTERDAM
FACULTY OF SCIENCE

Semi-supervised Invoice Information Extraction

A Regional Object Detection and Rule-Based Approach

Author:
Tessa Helmer

Internal supervisor:
dr. M. Cochez

Second reader:
prof. dr. A.C.M. Ran

External supervisor:
ir. E.F. Albers RE

De Boelelaan 1111
1081 HV Amsterdam

Burgemeester Tutein Noltheniuslaan 21
7316 BE Apeldoorn

The Netherlands
July 30, 2021

Preface

This thesis has been written to fulfil the final requirement to graduate from Vrije Universiteit Amsterdam with a Master's degree in Business Analytics. This thesis describes the research and findings of a six-month internship at SoliTrust Dimensions BV. The research in this thesis focuses on localising, reading, and classifying information from invoices. The experiments were executed on the external server Lisa by SURFsara.

First, I would like to thank my host company for the opportunity and the utilities to conduct this research. I would especially like to thank my external supervisor Ernst Albers for his continuous help, support, and feedback. Furthermore, I would like to thank Harmen Lenis for his help in understanding the SQL databases and the SoliTrust datamodel.

Next, I would like to thank my internal supervisor Michael Cochez for his guidance, supervision, and feedback, and for the brainstorming sessions that further improved the research. The bi-weekly meetings were very useful. Moreover, I would like to thank André Ran for being the second reader of this research project.

Lastly, I would like to thank my family, friends, and especially my boyfriend, for supporting me throughout this project.

Management summary

Recently, Machine Learning techniques have become very popular in the field of business innovation. One of these innovations is the automation of the analysis of creditor invoices. This analysis entails checking whether the PDF of the invoice and the logged information in the datamodel are one-to-one. Currently, only a random sample of the invoices is analysed by hand, while a Machine Learning model could also perform this task without needing to pick a random sample. However, recent studies that have tried to solve this problem required fully annotated data, which is not always readily available.

One field that is researched with less to no annotated data is Object Detection, which is the problem of localising and classifying objects on inputs like images. Many researchers have tried to solve the invoice information extraction problem by using common Object detection models like (Faster-)(R)CNN, as the analysis of creditor invoices can be reformulated as an Object Detection problem, where the important information has to be localised and classified. Therefore, this research will answer the question:

Is a (Faster-)(R)CNN model still applicable in invoice information extraction when the data is not fully supervised, or is a Rule-Based or a decision tree model more effective?

To answer the research question many (Faster-)(R)CNN architectures, a Rule-Based model, and a decision tree approach are implemented. Additionally, the unlabelled data must be annotated, which was done using the logged information, which, at times, was incomplete or incorrect. The final model is a (Faster-)(R)CNN that tries to locate the important regions on an invoice by using the images of the invoices as its input, which is followed by a Rule-Based model that identifies the specific information.

The results show that the level of annotation does have a great effect on the performance of an Object Detection model. Especially when regions are mislabelled. As a result, the final model was not able to find appropriate regions.

An ablation study was conducted to compare the potential performance of the model with a Rule-Based model and a decision tree model. The results show that the models that are dependent on the level of annotation, i.e., Object Detection and decision tree models, have lower performance than the Rule-Based model, which is not dependent on the level of annotation.

Future research is recommended to investigate whether adding textual features, such as a type map, to the input could lead to better results. A type map is a grid realisation of the types of the words on the invoice, e.g., the text “$10.-” will be entered into the grid as “price” on the position of “$10.-”. Several researchers have concluded that using textual and visual (the image) features leads to much better localisation results than solely using the image as input. Furthermore, as the level of annotation has such a great effect on the performance, it is advised to fully annotate the data by hand if optimal performance is preferred.

Contents

List of Figures

List of Tables

Table 1: Formal notation of frequently used terms

Notation	**Explanation**
C	Number of classes
p_{ij}	Number of samples that belong in class i, but are classified as class j
β	Batch size
f	Kernel size
$h(\cdot)$	Activation function
σ	Sigmoid function
l	Index of a layer of a Neural Network
$\omega_{kj}^{(l)}$	Weight that corresponds to the k^{th}
	hidden node of layer l and the j^{th} hidden node of the previous layer
$z_i^{(l)}$	Hidden node i of layer l
$a_j^{(l)}$	The *activations* of a hidden node
$b_{kj}^{(l)}$	Bias that corresponds to the k^{th}
	hidden node of layer l and the j^{th} hidden node of the previous layer
η	Learning rate
∇	Gradient (derivatives) operation
k	Number of filters
s	Stride
a	Number of anchors
X	Dataset
Y	Target values of the dataset
y^{true}	Actual targets
y^{pred}	Predicted targets

Acronyms

AE autoencoder. 11, 12

AI Artificial Intelligence. 1, 20

ANN Artificial Neural Network. 8, 11, 26

BERT Bidirectional Encoder Representations from Transformers. 6, 10, 11, 31, 32, 72

BiLSTM Bidirectional Long Short Term Memory neural network. 20, 30

CAE Convolutional Auto Encoder. iv, v, vii, viii, 12, 49–52, 54, 59, 60, 62, 64–66, 85

CNN Convolutional Neural Network. vii, 12, 14–17, 19, 26, 30, 31, 45, 50, 60

DAR Document Analysis and Recognition. 25, 26

IE Information Extraction. 1, 3–5, 8, 11, 26, 28–32, 73

KB Knowledge Base. 27

KG Knowledge Graphs. 27

LSTM Long Short Term Memory neural network. 20

MAE Mean Absolute Error. 21, 51, 61, 62

MLM Masked Language Model. 11, 32

MSE Mean Squared Error. 21, 51, 61, 62

NER Named Entity Recognition. 28

NLP Natural Language Processing. 11, 27–29, 32, 45

OCR Optical Character Recognition. vii, 1, 3, 6, 26, 29, 30, 32, 37, 38, 40, 70, 73

RCNN Region Based Convolutional Neural Network. v, vii, 17–19, 30, 40, 45, 49, 50, 52, 53, 56, 57, 59, 60, 66–68, 71–73, 89

RE Relationship Extraction. 28

RNN Recurrent Neural Network. vi, 19, 20

SGD Stochastic Gradient Descent. 22, 51, 61, 62

VRD Visually Rich Documents. 26, 28, 72

YOLO You Only Look Once. 30, 31

1 Introduction

Recently there has been a wide interest in improving business processes by automating repetitive and time-consuming tasks. One of these tasks is extracting important information from invoices. Previously, each document had to be logged in by hand, which had to be followed up by an accountant checking whether the recorded information was correct. Recent studies have tried to tackle this problem by using Artificial Intelligence (AI), where most models classify as Deep Learning models; this problem falls in the field of Information Extraction (IE). It even led to the start of a competition called “Scanned Receipts OCR and Key Information Extraction" (SROIE), by [Huang et al., 2019], where many researchers have tried various modelling techniques to find the best IE model.

Many companies have embraced the use of such models to automate their reading process, where most of them have a subscription to a program that reads and classifies the information from the documents for them. This program usually consists of a trained AI model that is inspired by one or multiple of the entries of the SROIE competition. However, even though these algorithms are quite precise, an accountant still has to manually check whether the acquired data and the document fields match.

Moreover, most of these models are not open source, which entails that the model can be seen as a black box that just produces what it is asked, without explaining how it is done. Consequently, a company either has to get a subscription or it must make its own model. While many of the SROIE models are successful in extracting the information, they do have to be fine-tuned to make them work for a specific task. Fine-tuning such a model requires a lot of fully annotated data. A key problem here is that most companies do not have a fully annotated dataset. One field that does successfully work with unsupervised (not annotated) data is Object Detection [Nguyen et al., 2019]. Object Detection is the task of locating and classifying objects on inputs like images, e.g., finding each zebra on a picture, where one of the most used and best-performing object detection models is the Faster-RCNN model by [Ren et al., 2015]. As a part of IE is to localise and classify the fields on documents, many researchers have solved IE by using object detection models. [Zhao et al., 2019, Shi et al., 2015]. However, they have not done so in an unsupervised or semi-supervised manner. Therefore, it is desirable to know whether partly annotated data could also produce accurate IE results when using an unsupervised object detection approach, or whether other approaches that do not depend on annotation or deep learning, might be more beneficial. Thus, the main research question of this thesis is:

Is a (Faster-)(R)CNN model still applicable in invoice information extraction when the data is not fully supervised, or is a Rule-Based or a decision tree model more effective?

This thesis is structured as follows: First, a more extensive description is given about the problem and the host company in Section 2. Next, Section 3 will describe the background information from the available literature. Thereafter Section 4 portrays the related work. This is followed by Section 5, where the data and its preparation are explained. Next, in Section 6, an extensive description is given about the methodology and experiments of this research. This is followed by the results in Section 7. The next section, Section 9, describes the conclusion and the recommendations of the research. Lastly, Section 8 will discuss the main findings and the limitations of the research.

2 Problem Description

This section gives a more extensive description of the host company, the problem, and the potential value to the company. First, a background about the company and the context of the problem will be described. Next, the problem itself will be further elaborated on. Lastly, the potential impact on the company will be described.

2.1 Company Background and Context

For the past few years, SoliTrust has been one of the fastest-growing companies in the Netherlands. SoliTrust is a company that provides IT auditing and data-analysis services, with its main focus being data related to the financial statement audit. Their clients are mainly audit firms, where they work with the auditors themselves, as well as the CFOs of these companies. SoliTrust helps these audit firms with improving the efficiency and effectiveness of the audits. One of the offerings is the automation of substantive audit procedures by applying data analysis.

Having many clients leads to having a lot of different databases to work with (e.g., SAP, Oracle, Dynamics NAV, etc). Hence, SoliTrust developed a datamodel to be able to uniform data analysis within the audit. Here, the collected data from the clients is transformed from their own ERP system to the SoliTrust datamodel.

2.2 The problem

Every year a company must provide their financial statement, which consists of the balance sheet, the income statement, and their elucidation. Such an audit is intended to inform clients and others who are interested in the financial status of that firm. This information is then used to decide whether other companies, such as banks or suppliers, would want to work with that firm. Furthermore, a company is obligated to make such a statement every year for the government, as a firm needs to pay taxes over its profits. [Boerse et al., 2017]

The more profit a firm makes, the more taxes they have to pay, hence some firms feel inclined to tamper with their books. While some tamper with the books, others unknowingly make mistakes, for example by making a typo. Therefore, a part of the financial statement audit is to validate the reliability of the data.

An inefficiency in the current audit process is the validation of creditor invoices. Creditor invoices are documents that describe what a firm bought and how much it paid for it. The only way to validate these is to get the invoice and check whether all fields match the fields in the database. This validation is currently executed by the auditors themselves. However, due to the many creditor invoices that a firm receives, an auditor cannot check them all. Hence, the auditor validates the invoices based on a sub-sample of the source documentation (PDF documents) with the data in the SoliTrust datamodel. This not only takes a lot of time, but it could potentially be automated. And with automation, the sample size could also be enlarged.

SoliTrust always tries to improve the efficiency and effectiveness of the audits. In order to further improve this problem, they would like to have a machine learning algorithm that could indicate whether an invoice (in PDF format) is one-to-one with the data that they stored in their datamodel. Hence, they asked for an extra control algorithm that can classify whether an invoice is correctly stored in the database and whether there are discrepancies. This can take a lot of load from an auditor's shoulders and could also further improve the service level of SoliTrust itself.

The goal is to make a model that can accurately identify whether the invoice and the SoliTrust datamodel counterpart match. This classification should then be added to the SoliTrust database as an extra column, which indicates whether the invoice has been verified or not. Therefore, it can also be seen as a reliability checker of the datamodel.

Moreover, if they do not match, the model should return the parts in the PDF that do not match, preferably by highlighting those fields in the PDF in red, and if they do match, then the fields should be highlighted in green. Furthermore, the model should be able to learn the structures of the PDFs, but it should also be applicable for invoices it has never seen before.

This problem can be seen as two subproblems: reading the text and classifying and localising the fields. Reading the text from an image is normally solved by an Optical Character Recognition (OCR) engine while localising and classifying specific fields is solved by an Information Extraction (IE) algorithm. Both are part of the Document Analysis and Recognition (DAR) field. OCR is the problem of reading text from Visually Rich Documents (VRD), which are images (of PDFs or websites, etc) that contain text. IE is the problem of returning the important information of such a document. This can be quite difficult, e.g., an invoice could have multiple dates on it, but there is only one invoice date. As a large part of IE is to localise and classify fields on an image, it can be reformulated as an Object Detection problem as well. One of the better performing object detection models is a Faster-RCNN model, which is explained in Section 3.3.3, and it is also used in IE tasks, which is described in Section 4.

Unlike other researchers who studied similar problems, the provided data in this research is not complete. Not complete in the context of this research entails that the invoices are not annotated and the information that should be used to annotate the data could be missing or incorrect. Hence, this research problem is an unsupervised problem, as the real information on an invoice is not known. However, by using the information that is logged into the datamodel, the data could be partly annotated. This led to a semi-supervised problem. Therefore, the research question is:

Is a (Faster-)(R)CNN model still applicable in invoice information extraction when the data is not fully supervised, or is a Rule-Based or a decision tree model more effective?

Many researchers that have made models to solve the IE problem combine textual features and visual image features to find the information, which is further explained in Section 4. These textual entities could be a text type or a text embedding. Hence, a follow-up question to the main research question is:

Does adding textual features, which are produced by heuristics and an OCR engine, to the input of the aforementioned model increase the IE performance?

Furthermore, some researchers have also concluded that using Spatial Pyramid pooling or atrous convolutional layers could increase the performance of IE models as well. [Chen et al., 2018, Zhao et al., 2019]. According to them, atrous layers focus on the context of the input as well as the regions themselves, which leads to better results. Hence, the second follow-up question is:

Does replacing the normal convolutional layers in the (Faster-)(R)CNN architecture to atrous convolutional layers lead to better IE results?

2.3 Potential value

This reliability checker tool could not only enhance the reliability of an audit, but it could also decrease some of the workload of an accountant. As mentioned before, currently an accountant has to check whether an invoice is legitimately put into the database by hand. Previously they would take a small sample and check those, with this system, they could only check those invoices that are marked as less reliable by the system. This is much more efficient and due to the highlighting of the different fields; it also requires less time.

3 Background

This section describes the background of the algorithms that are used to localise and extract information from text documents. First, the types of learning will be explained (Section 3.1), to give a more general understanding of the approaches. After that, the machine learning approaches are explained. First, a non-Deep learning approach is described, which is a decision tree (Section 3.2). Next, the deep learning approaches are described (Section 3.3). This section describes several Artificial Neural Network structures that form the basis of many IE and Object detection tasks. The last three sections are a more in-depth description of Object Detection (Section 3.4), Document Analysis and Recognition (Section 3.5), and Knowledge Acquisition (Section 3.6).

3.1 Types of learning

There are several types of learning in machine learning. The four most commonly known areas are supervised, unsupervised, reinforcement, and transfer learning. As was mentioned before, this project deals with unlabelled image data, which means that this project is unsupervised. This also means that the data, and therefore also the model, does not have a target. Hence, why an unsupervised model cannot predict a target (e.g., the position of the invoice date). However, it can find properties of the structure of the training set. This structure could indicate that there are multiple groups within the data, which is a task called clustering. [Goodfellow et al., 2016, Bishop, 2007]

Although the data is unlabelled, the model does have to find the correct fields. As is shown in the literature, most labelling tasks are trained with supervised datasets. This is because they know what they are looking for and consequently they achieve much higher accuracy scores [Nguyen et al., 2019]. Unlike an unsupervised model, a supervised learning model does have a training set that consists of training data (X) along with its corresponding target values (Y). A supervised model tries to map X to the corresponding Y. Hence, a supervised model tries to predict Y from X [Goodfellow et al., 2016] An example of a supervised model is a classification model that tries to identify cats. Such a dataset is filled with labelled pictures of cats and non-cats. After training, the model should be able to predict whether a new image is a cat or not.

An additional type of learning is called reinforcement learning. This kind of learning does not have an explicit dataset, instead, it interacts with the environment to maximise its reward function. This kind of model is called an agent. When the agent executes an action, it gets feedback, which is either a positive or a negative reward. The current action of the agent affects the current reward, but it also influences the reward in the future, e.g., getting over the finish line. [Goodfellow et al., 2016, Bishop, 2007] An example of this would be an agent that tries to find its home. Here making a step towards the home will increase the reward, while stepping further away from the home will decrease the reward. This type of learning is relevant to this research as it could be used to add an interactive user interface to the trained model. It could return a processed invoice to the user to receive feedback on the detection performance.

The last type of learning that will be discussed here is transfer learning. Transfer learning entails the problem of how to transfer information between partially related problems.

Generally, a trained model performs better than a non-trained model. Hence, a trained model environment could speed up the training process of similar new environments. [Quiñonero-Candela et al., 2008] Therefore, a model that is previously trained on identifying text could be used to fine-tune a model that wants to find specific text within a document. One common way to transfer the information of a trained model is to initialise the new model with the optimised weights of the trained model. For this to work, the two models must have the same model structure. This type of learning coincides with the notion of pre-trained models that can be fine-tuned. A lot of research has been done on this topic in the machine learning community. Therefore, there are a lot of pre-trained models, like BERT (described in Section 3.3.1), which is pre-trained on the 800 million text corpus by BooksCorpus, by [Zhu et al., 2015], and the English Wikipedia, which has 2500 million words. The long continuous sentences are used to better understand the relationship between words and sentences. As a result, a model like BERT can be fine-tuned for a wide variety of tasks, such as question and answering, and language inference.

3.2 Decision trees

The classification and localisation of objects can be done in many ways. One way could be to first generate the bounding boxes and the text with an OCR tool and then classifying the discovered boxes with a classification model. There are many classification models; the ones that are used in this research are decision trees and Artificial Neural Networks. The deep learning models are explained in later sections, while this section describes a non-Deep Learning approach.

One common classification model is a decision tree. A decision tree categorises the input based on its features. All samples start in one beginning node, which is split many times, the final node indicates the class. A decision tree algorithm tries to find the correct places and splitting criteria to split the data to find an accurate classification algorithm. The split of a node is usually based on a measure called the *information gain*. information gain calculates the gain in homogeneity after a new split, if this homogeneity is higher after the split, then the split will be made; an example is shown in Figure 1. Here, the left split shows a higher increase in homogeneity than the right split and it also shows a positive information gain concerning the node before the split.

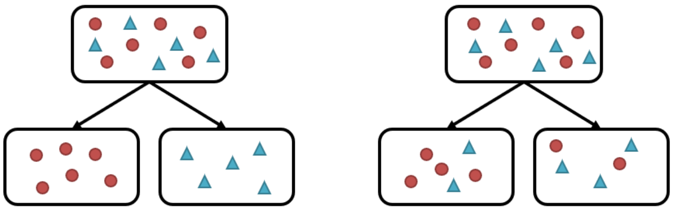


Figure 1: Example of two splits, the left figure will have a higher information gain than the right figure, as the homogeneity is higher. Furthermore, the information gain of the left split will be higher than leaving the node as is. Hence, that split will be made. As opposed to the right split, which will likely not have a higher information gain.

The information gain of one node is calculated by:

$$\begin{aligned} \text{Information gain} &= 1 - \text{Entropy} \\ &= 1 + \sum_{i=1}^{C} p_i \log_2(p_i) \end{aligned} \tag{1}$$

Here p_i indicates the probability of randomly picking a sample from class i. The entropy calculates the variance within a sample, a high variance (close to 1) means a high entropy. Hence, the goal of a split is to have a lower entropy (close to 0). A split creates 2 new nodes; hence the information gain of a split is calculated by $1-$weighted Entropy. Where the weighted Entropy is the sum of the entropies times the fraction of samples within a node. The example in Figure 1 has two classes: circle and triangle and 11 samples, with 6 circles and 5 triangles. Therefore $p_{circle} = 6/11$ and $p_{triangle} = 5/11$ before the split. As the variance between the samples is lower within the nodes after the left split than after the right split, the information gain of the left split will be higher. Hence, a decision tree model will choose the split instead of the right split.

A further improved decision tree is a model called LightGBM. This model trains a modified Gradient Boosting Decision Tree (GBDT), which is an ensemble/combination of a series of decision trees. GBDTs are useful for processing various datasets. Additionally, they have a solid theoretical presentation, they are simple to implement, and they have a high prediction accuracy. [Khoshrou and Pauwels, 2019] Many researchers have compared the performances between decision tree models, most of them concluded that LightGBM gave the best results. [Al Daoud, 2019, Al-Shari et al., 2021, Yang and Zhang, 2018]

The ensemble is optimised by gradient boosting, which is the technique of training all the models within the ensemble sequentially and adding the loss functions to get one main loss. Moreover, instead of having one weight update function for the whole model, it has a separate update function for each model, where the highest weights are assigned to the models with the highest error rates. The steps of the GBDT model can be found in [Rao et al., 2019]. [Rao et al., 2019, Khoshrou and Pauwels, 2019].

Unlike a normal GBDT, LightGBM has some modifications. Its modifications are the use of Gradient-based One-Side Sampling (GOSS) and Exclusive Feature Bundling (EFB). The modifications of Light-GBM are made to speed up the training and to improve the model even further. [Ke et al., 2017]

The first modification, GOSS, is made to put more focus on the undertrained models within the ensemble. As was said before, all models have their own loss function and gradient, and their own update function. A model with a small gradient has a small training error, while an instance with a large gradient has a large training error. The models with high gradients are the ones that must be further trained. Hence, the models with a smaller gradient have a lower impact on the training procedure. Therefore, GOSS proposes to randomly drop the models with low gradients instead of solely assigning more weight to the models with the higher gradients.

The second modification, EFB, was made to significantly speed up the training without changing the accuracy. EFB works with the sparsity of the feature space, which can be large, but rarely takes nonzero values. Hence, the bundling of the features can be done more efficiently. Instead of representing the features as a table, they are seen as graphs, which significantly diminishes the size of the feature space and thus the training time. Here the features are seen as vertices, and they connect to an edge if they are both not mutually exclusive.

3.3 Artificial Neural Networks

An Artificial Neural Network (ANN), also known as a multi-layer perceptron, is one of the best-known deep learning algorithms. They are used for many different problems, such as classification, object detection, and regression. The goal of an ANN is to approximate some function f that maps the input variables $\mathbf{x} = (x_0, ..., x_D)^T$ to the target values $\mathbf{y} = (y_0, ..., y_D)^T$. An example of input (x_i) could be a business document with label (y_i) "invoice".

There are several different types of ANN architectures, some of them will be discussed in this section. First, the general ANN structure is described in Section 3.3.1. After that, some of the specific types are explained, namely, Auto-encoders (Section 3.3.2), Convolutional Neural Networks (Section 3.3.3), and Recurrent Neural Networks (Section 3.3.4, as these models are common solutions to IE and object detection problems. The last two sections will describe the training process (Section 3.3.5) and the evaluation process (Section 3.3.6) of an ANN.

The references that are used throughout this section are [Goodfellow et al., 2016] Chapters 6, 9, 10, and 14, [Bishop, 2007] Chapters 1 and 5, [Aghdam and Heravi, 2017] Chapter 3, and [Maggipinto et al., 2018].

3.3.1 General Architecture of a Neural Network

Neural Networks typically consist of many different functions, meaning that the main function f is of the form $f(\mathbf{x}) = f^{(L)}(f^{(L-1)}(...f^{(1)}(\mathbf{x})...))$, where a function $f^{(i)}$ corresponds to its respective hidden layer i. There are three types of layers, namely the input, hidden, and output layers. The input layer consists of the input variables $\mathbf{x}$ and the output layer consists of the output variables $\mathbf{y}$. Unlike the input and output layer, there can be multiple hidden layers. A depiction of a Neural Network is shown in Figure 2.

As is shown in the figure, each hidden layer has hidden nodes (also called neurons). Hidden nodes $(z_i^{(l)})$ are variables in the hidden layer that are composed of the outputs of the previous layer, and together they form the input of the next layer. Each hidden layer (l) has its own amount of hidden nodes $m^{(l)}$. A hidden node is not only a linear combination of its inputs, which is called an *activation*, but it is also transformed by a differentiable, nonlinear *activation* function $h(\cdot)$. Hence $z_j^{(l)}$ is of the form:

$$z_k^{(l)} = h(a_k^{(l)}) = h\left(\sum_{j=1}^{m^{(l)}} \omega_{kj}^{(l)} z_j^{(l-1)} + b_{kj}^{(l)}\right) \tag{2}$$

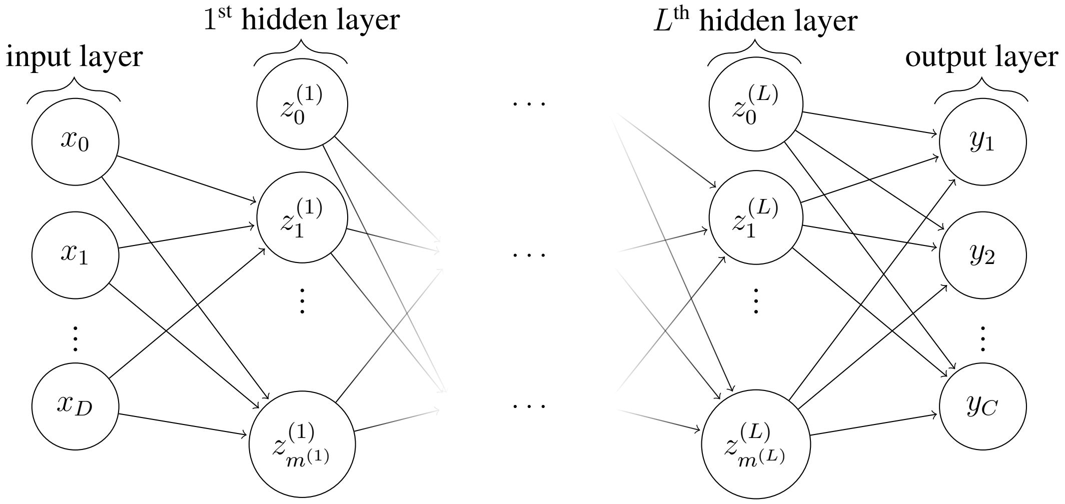


Figure 2: Network graph of a $(L+1)$-layer perceptron with D input units and C output units. The l^{th} hidden layer contains $m^{(l)}$ hidden units.

Where $z_j^{(0)} = x_0$ is the input, $z_j^{(L+1)} = y_j$ is the output, $\omega_{kj}^{(l)}$ is the weight that corresponds to the k^{th} hidden node of layer l and the j^{th} hidden node of the previous layer, $b_{kj}^{(l)}$ is the bias that corresponds to the k^{th} hidden node of layer l and the j^{th} hidden node of the previous layer and $a_j^{(l)}$ are the *activations*.

Activation functions are chosen based on the nature of the data and the assumed distribution of the target variables. There are two types of activation functions, probabilistic and linear functions. While the probabilistic functions are used in the last layer of a neural network, the linear functions can be used throughout the whole network.

The activation functions that are used in this research are sigmoid, SoftMax, and ReLU.

The sigmoid (or logistic) activation function is a probabilistic function, which entails that it produces an output between zero and one. The sigmoid (σ) function is given by:

$$\sigma(a_k) = \frac{1}{1 + \exp(-a_k)} \tag{3}$$

The next function is the SoftMax function, which is a generalised logistic activation function. This function is used for multi-class classification, and it is also called the normalised exponential function. The function is a 'soft' maximisation function that gives the most weight to the class that it thinks that the input belongs to. The SoftMax function is given by:

$$\text{SoftMax}(a_k) = \frac{\exp(a_k)}{\sum_j \exp(a_j)} \tag{4}$$

While the previously mentioned functions are used for classification, the Rectified Linear Unit (ReLU) is used for faster convergence, and it can also make a model sparser. Sparsity entails that some weights will become close to or equal to zero. This is a helpful quality as not all neurons have to be activated to find the best result. Hence, sparsity results in faster and more concise models, and they often have better performance and less overfitting than their counterparts. The ReLU function is given by:

$$\text{ReLU}(a_k) = \max\{0, a_k\} \tag{5}$$

There are a few generalisations of the ReLU function. They are all given by:

$$\text{gen\_ReLU}(a_k, \alpha_k) = \max\{0, a_k\} + \alpha_k \min\{0, a_k\} \tag{6}$$

Here α_k represents a nonzero slope. There are multiple generalisations of the ReLU function. The first generalisation is the Absolute value rectification ($|a_k|$), which fixes $\alpha_k = -1$, ending up with just the absolute value of a_k. The second function is called the leaky ReLU, which fixes α_k to a small value, e.g., 0.01. The last generalisation is the parametric ReLU, or PReLU, which treats α_k as a learnable parameter. The parameter is then optimised alongside the weights of the layers.

The generalised ReLU functions were developed to avoid the dying ReLU problem. The dying ReLU problem happens when a neuron is stuck in the negative range. These neurons tend to not recover, which results in many neurons not being used. The unused neurons are ignored and will not be learned from. By using either of the generalised versions, the neurons cannot get stuck, which also ensures that the model can still learn from them.

Instead of using activation functions, some models use attention heads. These models are called Transformers. Attention head functions are described as the mapping of a query and a set of key-value pairs to an output value. Similar to an activation function, the output is the weighted sum of the values. However, instead of using all hidden neurons in a layer to form the output, an attention head first calculates a compatibility score between the nodes, which is used to decide which nodes are important enough to include in the output. The rest of the nodes are ignored. Moreover, the compatibility scores are used as the weights as well.

One of the most used attention head is called the Dot-Product Attention, which is given by:

$$\text{Attention}(Q, K, V) = \text{SoftMax}\left(QK^T\right) V \tag{7}$$

Here Q is a matrix that holds all queries, K holds all the keys and V holds the values. The queries represent the vector embeddings of the input words, while K and V represent the previously generated words and embedding values, respectively. Moreover, in this instance, the SoftMax function calculates the compatibility score between the words.

In addition, an attention head its input is the output of all previous layers in the network, instead of just the layer previous to the attention head. Consequently, the network has a global dependency amongst all layers, which in turn improves the computational performance of the model. A further explanation on Transformers can be found in [Vaswani et al., 2017].

One commonly known Transformer is the Bidirectional Encoder Representations from

Transformers (BERT) model, by [Devlin et al., 2018]. BERT is a language representation model that was designed to pre-train deep bidirectional representations from unlabelled text. This is done by both conditioning on the left and the right context in all layers of the network. To ensure that the model can only look at the context, the model is pre-trained to perform two tasks: Masked Language Model (MLM) and Next Sentence Prediction (NSP). MLM randomly masks some of the tokens in a sentence, which then have to be predicted based on their context. While NSP is the task of predicting whether two input sentences are consecutive sentences or not. The framework of the model is divided into two steps: *pre-training* and *fine-tuning*. The pre-training is used to learn the low-level features of the model. This is done by training the model on a large amount of unlabelled data. The weights of the pre-trained model are then used as the initial weights in the fine-tuning step, where the model is trained again. Unlike the data in the pre-training step, the data in the fine-tuning step is labelled. The pre-trained model is trained on 1.28 million images, and it is added to the transformer library in Python. Many NLP tasks can be solved by fine-tuning the BERT model, e.g., sensitivity analysis, question answering, and IE.

3.3.2 Autoencoder

An autoencoder (AE) is an unsupervised neural network, which tries to learn the structure of its input. Its characteristic is that it is trained to attempt to recreate its input to its output, i.e., $\mathbf{y} = \mathbf{x}$. Consequently, an autoencoder has two parts, an encoder ($\mathbf{h} = f(\mathbf{x})$) and a decoder ($\mathbf{r} = g(\mathbf{h})$). Usually, an autoencoder has a symmetric structure, i.e., the encoder and decoder have the same number of layers. An example is shown in Figure 3. The encoder encodes the input to a lower dimension, while the decoder decodes the lower dimension back into the input. The decoder is trained to make an approximate mapping, instead of an exact one. This ensures that the model learns the structure and important parts of the input, instead of the details. Hence, autoencoders are a popular pre-trained model structure for transfer learning in unsupervised situations. A pre-trained autoencoder can be very beneficial for classification and other modelling tasks, as it already knows the structure of the input.

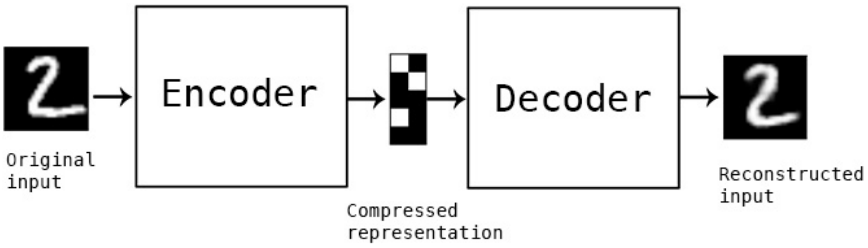


Figure 3: Example of symmetric autoencoder structure, this image is taken from *The Keras Blog* [Chollet, 2016].

An autoencoder is trained by minimising the difference between the input and its reconstructed output. The training process of a general ANN is described in Section 3.3.5. After training the autoencoder, the encoded representation is usually used as a feature. As this feature holds the information about the structure of the input, it can be used as the backbone of other models, like classification, anomaly, or object detection. This transfer learning approach takes the trained encoder and uses the encoded feature as the input of the next model.

Autoencoders are also used for information retrieval, this approach is called semantic hashing. This method stores database entries as binary code vectors, which are matched to the queries in the input data. It has been applied to both textual input and images. [Salakhutdinov and Hinton, 2009]

A variation of the autoencoder is the sparse autoencoder. This autoencoder minimises the loss function with an additional penalty ($\Omega(\mathbf{h})$). The model then tries to minimise the difference between the input and the reconstructed output plus the penalty. Such a model is typically used to learn features for another task, such as classification. A sparse autoencoder is taught to only respond to unique statistical features of the dataset, hence it can be a strong backbone.

Another variation is the denoising autoencoder. This autoencoder tries to find the original data from noisy corrupted data. Hence, this model learns the structure of the original data and matches that to the noisy input to denoise it. These kinds of autoencoders are usually used to unblur images or to crop and rotate them into the right position.

A popular variation of the autoencoder is a Convolutional Auto Encoder (CAE). This type of AE uses convolutional and pooling layers to encode and decode images, which are explained in Section 3.3.3. The trained encoder could then be used for problems like information extraction or classification of the images.

3.3.3 Convolutional Neural Networks

The main model of this research, the (Faster-)RCNN, is a further improved Convolutional Neural Network. A Convolutional Neural Network (CNN) is a specialised neural network that works best with grid-like topologies (e.g., images). A CNN can be used for object detection, reading text, or classification (e.g., of a document type). This section will describe the convolutional layers and some improved versions of the CNN.

Convolutional layers

What differentiates a CNN from a typical NN is the convolutional layers. A convolutional layer uses a so-called convolutional function, which is a weighted average of several measurements. It obtains less noisy data and creates features from the input. In a CNN, this function is represented as a filter (also called a kernel), this filter is a $f \times f$ matrix (or tensor) which convolves the input to a lower dimension. One example of CNN input is the RGB representation of an image. Here the convolutional layer can be used to detect the edges in the image and hence it can find the shapes of, for example, faces and letters.

The convolutional layer has 4 parameters: filter size (f), padding (p), stride (s) and number of filters (k). A filter size of (3×3) in a one-channel input tensor entails that the filter is a cube of 3 by 3 pixels. The stride of such a layer describes the step size of the filter.

When the step size is one, the filter jumps from pixel to pixel, while a larger stride means that some pixels are excluded; the operation is shown in Figure 5a.

The next parameter is padding; padding is an operation that adds a border of white pixels around the image, which is illustrated in Figure 5b. While this parameter can be set to any number, some common choices are *valid* or *same*. *Valid* indicates that there will be no padding, while *same* indicates that there is enough padding to ensure that the input size stays the same if the stride is equal to 1. The last parameter, the number of filters, indicates the number of filters that will convolve the input, and therefore it also determines the final dimension of the output size.

The convolutional operation itself calculates the dot product between the filter and the same sized region in the input space. An example is shown in Figure 4.

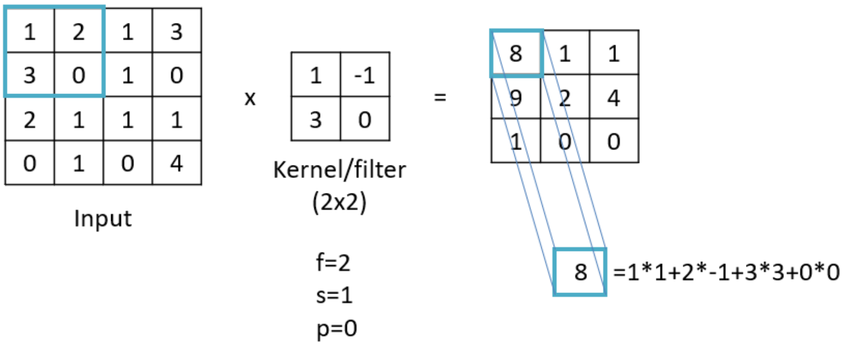
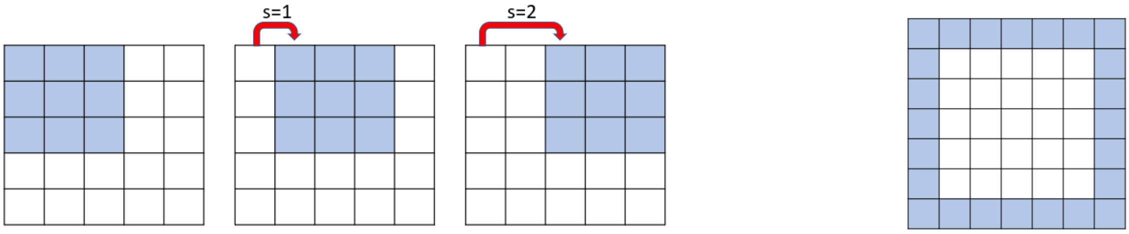


Figure 4: Example of convolutional operation on a one channel input matrix. Here the filter has a size of (2×2), a stride of 1, and zero-padding. The convolutional operation finds the dot-product of the filter and the regions within the input. These regions have the same size as the filter; one of the regions is given within the blue lines. This region moves one step to the right after it calculated the dot-product within a region. Here it can do two steps to the right and then it must go down and start from the left again. As it can step down two times as well, the output is a 3×3 matrix.

The four parameters all contribute to the output size of the layer. As was mentioned before, an input image is a $n \times n \times 3$ tensor, which means that the first filter of size $(f \times f)$ is actually of size $(f \times f \times 3)$, because it must visit each layer of the tensor. A filter reduces the size of an image, as it convolves the pixels within a region into one value. The larger the filter, the smaller the output of the convolution will be. Similarly, this also coincides with the step size, while padding enlarges the output size, the stride causes a smaller output size. If the input of a convolutional layer is given by $(27 \times 27 \times 3)$, i.e., one image of 27 by 27 pixels, with $f = 3$, $s = 2$, $p = 0$ (*valid*) and $k = 5$; then the output size is given by:

$$\begin{aligned}
\text{output\_size} &= \left(\left\lceil \frac{n + 2p - f}{s} + 1 \right\rceil \times \left\lceil \frac{n + 2p - f}{s} + 1 \right\rceil \times k \right) \\
&= \left(\left\lceil \frac{27 + 2 \cdot 0 - 3}{2} + 1 \right\rceil \times \left\lceil \frac{27 + 2 \cdot 0 - 3}{2} + 1 \right\rceil \times 5 \right) \\
&= (13 \times 13 \times 5)
\end{aligned} \tag{8}$$



(a) Stride of 1 and stride of 2 with filter size (3,3)

(b) Padding of 1

Figure 5: Example convolutional parameters

As mentioned before, to get the same dimensions of the image (n) after convolution, then the padding should be set to *same*. The actual padding size will then be determined by the function:

$$p = \frac{f-1}{2} \tag{9}$$

An example of 2 consecutive convolutional operations is shown in Figure 6, where the first convolutional layer has 6 filters of size (7x7), and the second layer has one filter of size (5x5). This figure shows the feature engineering of a CNN model. All filters in a convolutional layer produce a new image, which holds different information about the input. Some try to find the edges, texture, or shapes of the object, while other focus on the background or contrast. The following convolutional layers combine the information of the previously generated images, which in this example, intensifies the edges. While other filters in the second layer could intensify the shape or texture of the object. These features are then used to find a pattern within a class, which are used to ultimately classify whether an image belongs to a class or not.

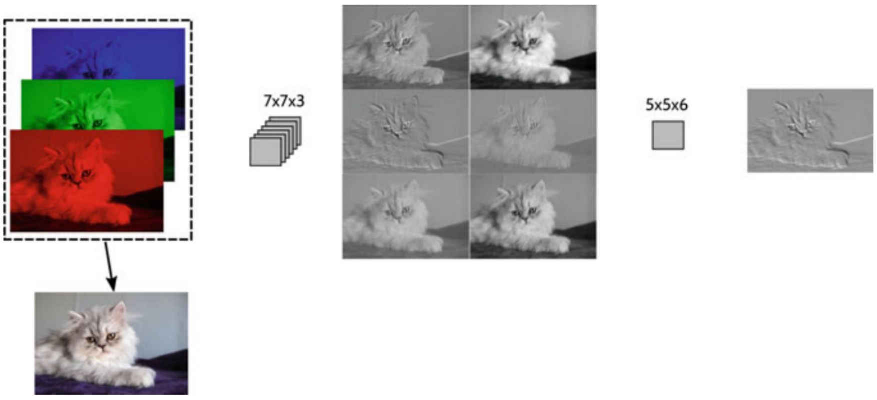


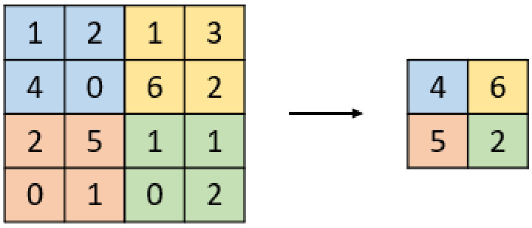
Figure 6: Example of convolutional operations with the RGB (red, green, and blue) channels image, which is a three-channel input. The image is taken from [Aghdam and Heravi, 2017]

Another characteristic of a CNN is that it uses a concept called parameter sharing. Parameter sharing refers to using the same weight parameters for more than one function or layer of the model. Instead of having a different weight for every element in every layer, a CNN uses the same values in the filter for every position in one convolutional operation.

As can be seen in Figure 4, this filter is used on the whole input, instead of just that one region.

Similar to the traditional NN, a CNN can also use activation functions. Usually, the ReLU function is used after the convolutional layer to make the CNN a sparse CNN. The other layers that are normally in a CNN are the pooling and the fully connected layers. A pooling layer statistically summarises the output of a convolutional layer. Like the convolutional operation, a pooling function operation also uses a filter, but instead of calculating the weighted average of the position, it calculates the max (MaxPooling), average, min, etc. The most commonly used pooling layer is the MaxPooling layer with a filter size of 2 and a stride of 2, an example is depicted in Figure 7. A pairing of one convolutional layer and one consecutive pooling layer is typically seen as one layer in the model.

A drawback of the basic pooling and convolutional layers is that they do not focus on the context of a pixel. This led to the introduction of the Spatial Pyramid Pooling layer and the Atrous Convolutional layer. [Chen et al., 2018] To ensure that the context is considered during training, a dilation parameter is added to the filter. The dilation rate determines the space between the pixels in a filter. A normal convolutional layer has no space between the pixels, a filter with size 3×3 is spanned over a region of 3×3, whereas a filter of size 3×3 with a dilation of 1 is spanned over a region of 5×5 pixels. A representation of this can be seen in Figure 8. The figure also depicts the formation of a spatial pyramid. This pyramid is created by using multiple atrous pooling layers with increasing dilation rates.



1	2	1	3
4	0	6	2
2	5	1	1
0	1	0	2

4	6
5	2

Figure 7: Example of MaxPooling with $f = 2$ and $s = 2$. It shows that the maximum of each filter position is returned.

Atrous convolution is shown to have a better performance than normal convolution in some object classification tasks. One of these tasks is when objects within the same class have different font sizes. An example of this is the article table on an invoice. Some invoices hold many lines with purchased goods, while others do not, which results in many different shapes and sizes. Furthermore, atrous convolution also tends to have a better performance than normal convolution if an input image has a context-dependent structure. Consequently, they perform well on invoice classification and information extraction tasks. Additionally, they are also known to reduce scale imbalance problems within object detection tasks. [Oksuz et al., 2020]

Another type of convolutional layer is a Graph Convolutional layer, where multiple graph convolutional layers form a Graph Convolutional Network (GCN). A GCN takes a graph as input, this graph contains a feature description for each node, a description of the structure of the graph, which is usually an adjacency matrix that identifies the edges, and the

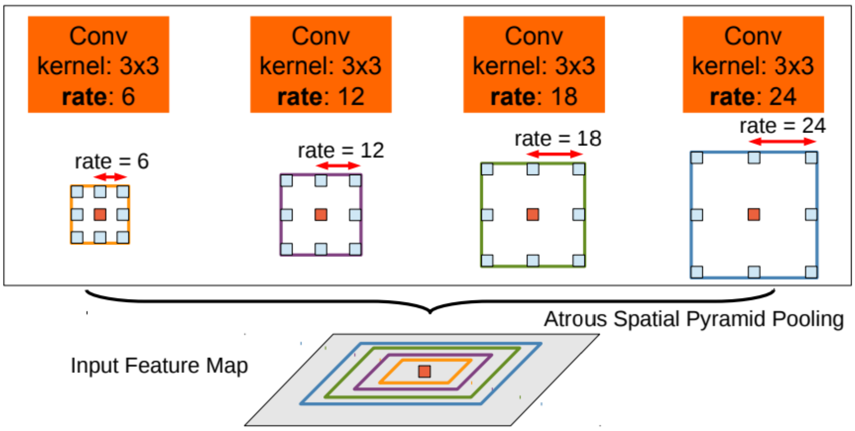


Figure 8: Example of dilation rate in a filter. It shows the Spatial pyramid of the input feature map, where each layer increases its dilation. Here the first atrous layer has a dilation of 6, which means that there are six pixels between the pixels that are shown in blue that are skipped. The image is taken from [Chen et al., 2018]

labels of the nodes. The GCN layers are then used to find a general structure within a graph. This general structure can then be used to find fields, such as invoice dates, on an invoice.

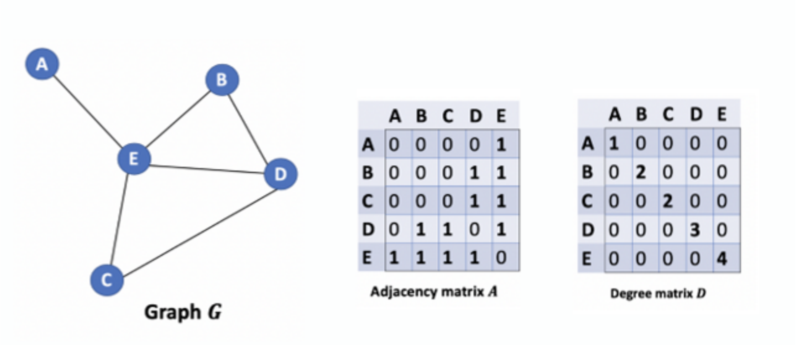
Instead of using a fixed filter size to convolve features, a GCN layer takes a node, finds all its neighbours, and convolves those features. So, instead of taking the weighted sum of a fixed filter, it finds the weighted sum of the features in the neighbourhood. An example of the input of a GCN is shown in Figure 9. This figure shows the input graph, its adjacency matrix, which shows which nodes are connected, and the degree matrix, which shows the number of connections a node has. These inputs are then put into the graph convolutional layer, along with a feature matrix, which is given by:

$$Z = h\left((D + I_N)^{-1/2} (A + I_N) (D + I_N)^{-1/2} XW\right) \tag{10}$$

Where D is the degree matrix, A is the adjacency matrix, X is the feature matrix, I_N is the identity matrix, and W, the weights, and h is the activation function. A more extensive explanation about graph convolution can be found in [Kipf and Welling, 2016].

The last type of layer is a fully connected layer. This layer connects all neurons in the previous layer to the current layer, which linearises the output. This output is then used to classify the images.

One of the more commonly known CNN models is the 2012 winner of the yearly image classification competition called ImageNet Large Scale Visual Recognition Challenge (ILSVRC). This model is called AlexNet and was introduced by [Krizhevsky et al., 2012]. The architecture of AlexNet is shown in Figure 10. The architecture shows that the model first finds the feature space of an input image by using five consecutive convolutional layers. These features are then linearised by the fully connected layers. As the competition dataset has 1000 different classes, the output layer is a vector of size 1000, containing one 1, which indicates the class that the image belongs to, and 999 zeros.



	A	B	C	D	E
A	0	0	0	0	1
B	0	0	0	1	1
C	0	0	0	1	1
D	0	1	1	0	1
E	1	1	1	1	0

Adjacency matrix *A*

	A	B	C	D	E
A	1	0	0	0	0
B	0	2	0	0	0
C	0	0	2	0	0
D	0	0	0	3	0
E	0	0	0	0	4

Degree matrix *D*

Figure 9: Example of an input graph of a GCN and graph convolution. A is the adjacency matrix, which entails that it shows which nodes are connected by an edge. As can be seen in the graph, node A is only connected to E, hence, the fields $A_{AE} = A_{EA} = 1$, and the other values in the A column and A row are 0. D is the degree matrix, which shows the number of connections of a node. As can be seen, node D has three connections, hence $D_{DD} = 3$. This example is taken from [Pham, 2020].

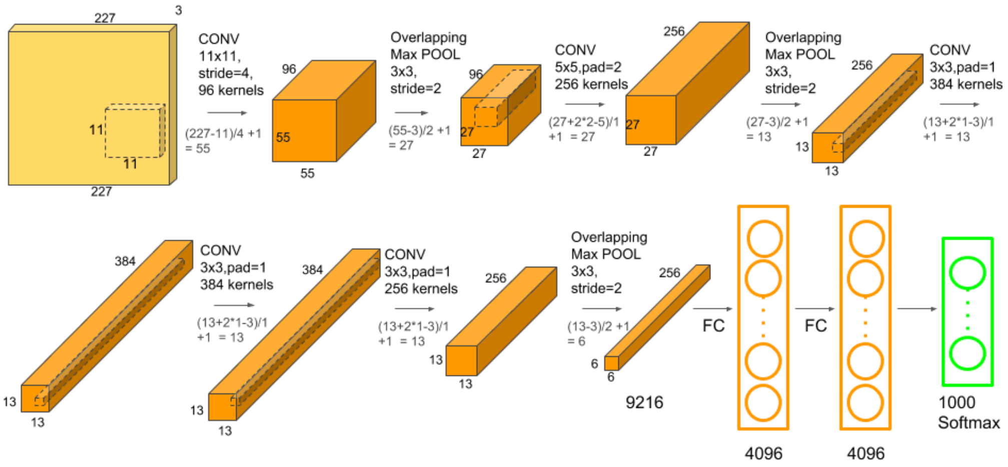


Figure 10: This is the architecture of AlexNet; it has 5 convolutional layers and 3 fully connected layers. Where a convolutional layer followed by a pooling layer counts as one layer in the model. Image is taken from [Nayak, 2018].

Improved versions

An extended CNN is the Region Based Convolutional Neural Network (RCNN), introduced by [Girshick et al., 2013], which is a model that is used for Object Detection. This model is a Convolutional Neural Network that generates a lot of category-independent region proposals in an input image and extracts features from them. After the proposed regions are found, they are classified by a CNN based on their features. The regions that are classified as an earlier defined label are bounded with a box, while the regions that are not probable to be labelled with one of the classes are ignored. An example of such a network is shown in Figure 11.

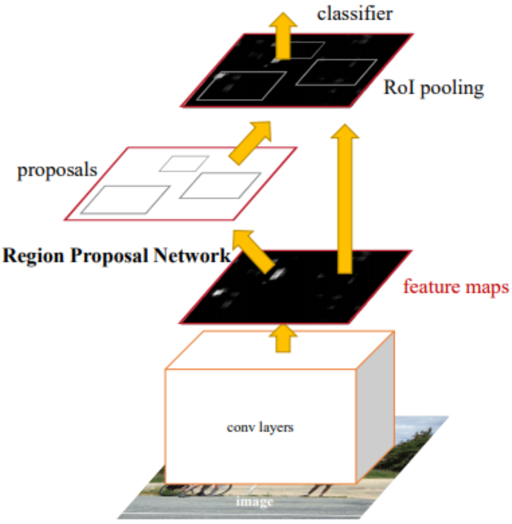


Figure 11: General RCNN architecture. The image is taken from [Ren et al., 2015]

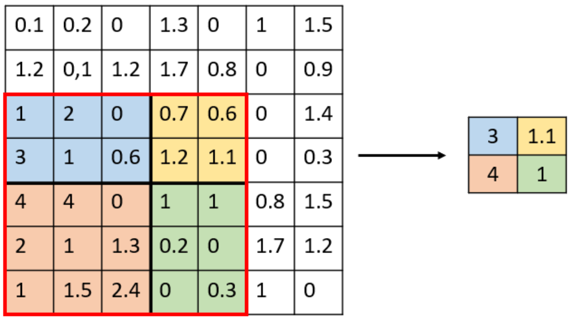
The figure shows that the model is divided into 3 main blocks: the convolutional layers, the Region Proposal Network, and the classifier. In RCNN, these three separate models are trained successively.

The first part creates a feature space of the image based on the convolutional layers of a pre-trained image classification model. In the paper written by [Girshick et al., 2013] they propose using image classification networks like AlexNet [Krizhevsky et al., 2012]. The backbone is used to create a valuable encoded feature space that is passed to the next modelling stages. This feature space is formed by passing an input image through the pre-trained model layers. However, not all layers are used, namely, only the convolutional and pooling layers are needed to create the feature space. The other layers (the fully connected and output layers) are not used in further stages.

This feature space is then passed to the next stage: the Region Proposal Network (RPN). An RPN normally consists of one convolutional layer, which is followed up by two output layers. The first output layer determines whether a region contains an object or not, and the second output layer holds the coordinates of the region. The convolutional layer is used to create anchors in the image, which are predefined regions around a pixel. A convolutional layer with a filter of size 3×3 corresponds to 9 anchor boxes per pixel, as there are three groups of three anchors. Having 9 anchors per pixel would lead to $6 \times 6 \times 9 = 324$ possible regions with a feature space output generated by AlexNet, as its feature space is of the size $(6 \times 6 \times 256)$. The correct anchors are based on the ground truth, which is defined before training.

The RPN model is then trained to successfully classify whether an anchor holds an object or not. This output is then used to find a fixed number of regions, which are called the proposals. These proposals are then passed to the next model, which does the final object classification. The first layer of this model is an ROI (Region of Interest) Pooling layer. This layer takes the proposed region coordinates and finds the values that correspond to that region in the feature space. This subsection in the feature space is then pooled to a fixed output size. An example of ROI pooling is depicted in Figure 12. ROI pooling is necessary as a normal layer in a network expects that the input regions have the same size. The pooled features are then passed to the last part of the model, which classifies the object and finds the coordinates.

To ensure that both output layers of the RPN model and the classifier models are correlated, the classification and regression losses are either added or multiplied. [Jiang et al., 2020] This ensures that the model tries to minimise the loss of both output layers, instead of mainly focussing on optimising one of them.



0.1	0.2	0	1.3	0	1	1.5
1.2	0,1	1.2	1.7	0.8	0	0.9
1	2	0	0.7	0.6	0	1.4
3	1	0.6	1.2	1.1	0	0.3
4	4	0	1	1	0.8	1.5
2	1	1.3	0.2	0	1.7	1.2
1	1.5	2.4	0	0.3	1	0

3	1.1
4	1

Figure 12: An example of ROI pooling. It has a 7×7 feature space, a 5×5 region proposal (framed in red)., and a 2×2 fixed output size. As can be seen the maximum of each region is returned.

A further development of the RCNN model is the Faster-RCNN model, which was first proposed by [Ren et al., 2015]. Instead of only training the three parts successively, they are also trained as one model. So, instead of training each part separately and having the input of the previous stages fixed, the Faster-RCNN model trains all previous layers alongside the layers in the next part. This not only speeds up the prediction process, but it also leads to more accurate object detection.

3.3.4 Recurrent Neural Networks

Another specialised type of NN is the Recurrent Neural Network (RNN). Like the CNN, RNNs also work with two-dimensional image data. However, unlike CNNs, RNNs are specialised in sequential data instead of one still image, i.e., a video. Examples of sequences of data are sequences of text (a sentence), images (a video), and prices (stock market tracker).

The task of an RNN is to accurately predict the next word, image, or price of the sequence or within a sequence. There are two main design ideas, namely, that the output depends on the entire prefix of the sequence up until that point (its history or memory) and the model only uses one set of parameters across all positions in the sequence. Unlike a traditional NN, the RNN receives input at every layer, excluding the output layer, instead of just the first layer. Hence, the hidden node at time t is given by:

$$\mathbf{z}_t = g\left(\mathbf{U}\mathbf{x}_t + \mathbf{W}\mathbf{z}_{t-1} + \mathbf{b}\right) \tag{11}$$

Where $\mathbf{U}$ is the weight matrix of the input transformation, $\mathbf{W}$ represents the hidden weights, $\mathbf{b}$ is the bias vector, and g is a nonlinear activation function, e.g., sigmoid or tanh. The hidden states can be transformed by a parameter matrix $\mathbf{V}$, an activation function h, and another bias vector $\mathbf{c}$ to the output (the next part of the sequence) at time t, the equation is given by:

$$\hat{\mathbf{y}}_t = h(\mathbf{V}\mathbf{z}_t + \mathbf{c}) \tag{12}$$

An example of an RNN architecture is shown in Figure 13. The figure on the left shows the basic structure of an RNN, while the network on the right shows the unrolled network, which shows every layer.

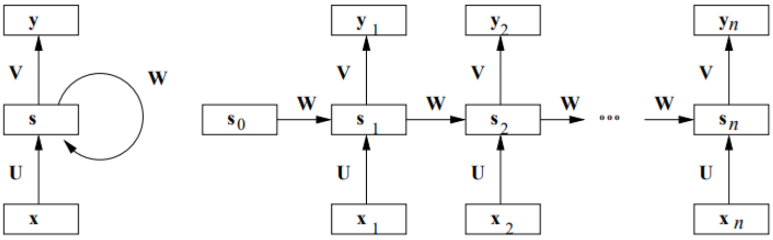


Figure 13: Basic and unrolled RNN architecture. It shows the continuous steam of input and output throughout the model. The image is taken from [Leskovec et al., 2014].

A refinement of the RNN is the Long Short Term Memory neural network (LSTM), which is another popular model that is used to train an object detection model. An LSTM is a Neural Network that replaces the hidden vectors with memory blocks, which are equipped with special gates, namely, an input, forget, and output gate. A gate can be seen as an opening of the memory block, which does not let all information pass through it. The different gates decide which information can pass, namely, which new information should be added to the memory (input gate), which of the current memory is not relevant in the current stage of the model (forget), and which part of the newly made memory should go to the next layer (output). These gates ensure that only the important information of the previous learning epochs is remembered, instead of all learned information. This structure ensures that long-range dependencies are easier to find and preserve. Hence, it is a strong character and word classifier. [Naseer and Zafar, 2019, Goyal et al., 2018]

Another advancement is the bidirectional RNN (BRNN), which was introduced by [Schuster and Paliwal, 1997]. This model splits the neurons in a part that is responsible for the next state and a part that is responsible for the previous state. The general architecture of a BRNN model is shown in Figure 14. The advantage of using a bidirectional model is that unlike the other two models, it does not assume that the input size is equal to the output size, as it trains the model in both directions. A further development of the BRNN and the LSTM led to the Bidirectional Long Short Term Memory neural network (BiLSTM) model, by [Schuster and Paliwal, 1997], which is now one of the more popular AI models.

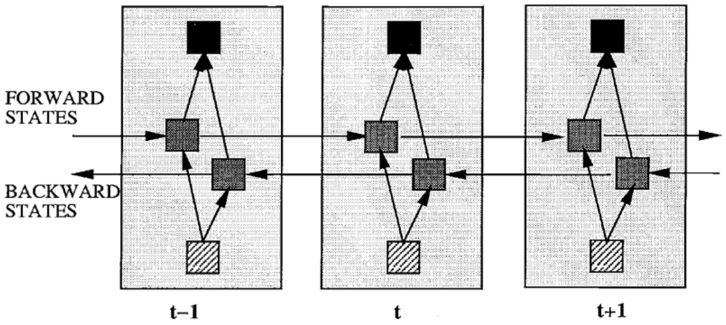


Figure 14: General structure of the bidirectional Recurrent Neural Network, which is shown unfolded in time for three time steps. The image is taken from [Schuster and Paliwal, 1997].

3.3.5 Training a Neural Network

The goal of a Neural Network is to find the weights that minimise the error ($E(\mathbf{w})$) between the true targets and the mapping function f. There are three types of loss functions: losses for regression problems, losses for single-class classification, and losses for multi-class classification. Loss functions for regression are used in problems where the predicted value should be a continuous value, like price or coordinate prediction. All categories have many different loss function options. The losses for regression problems that are used in this research are: the Mean Squared Error (MSE) and Mean Absolute Error (MAE), the loss function that is used for single class classification is binary cross-entropy, and the loss function that is used for multi-class classification is categorical cross-entropy.

Loss functions measure the error between the target ($\mathbf{y}^{\text{true}}$), which is either the true value of x or the label y, and the predicted values $\mathbf{y}^{\text{pred}}$.

The MSE is given by:

$$\text{MSE}(\mathbf{y}^{\text{true}}||\mathbf{y}^{\text{pred}}) = \frac{1}{M}\sum_{x=1}^{M}\left[y_x^{\text{true}} - y_x^{\text{pred}}\right]^2 \tag{13}$$

The MAE is given by:

$$\text{MAE}(\mathbf{y}^{\text{true}}||\mathbf{y}^{\text{pred}}) = \frac{1}{M}\sum_{x=1}^{M}|y_x^{\text{true}} - y_x^{\text{pred}}| \tag{14}$$

The binary cross-entropy loss function can only be used in classification problems with one class. Hence, the targets (y) are given by a 0 (does not belong to the class) or a 1 (belongs to the class). It is given by:

$$\text{BCE}(\mathbf{y}^{\text{true}}||\mathbf{y}^{\text{pred}}) = -\frac{1}{M}\sum_{x} y_x^{\text{true}}\ln\left\{y_x^{\text{pred}}\right\} + (1 - y_x^{\text{true}})\ln\left\{1 - y_x^{\text{pred}}\right\} \tag{15}$$

The categorical cross-entropy loss function can be used for problems with multiple classes as well as problems with one class. Here the classes are also represented as a 0 or a 1, but then there are C labels per input sample, one for each class. Other models prefer giving each class a number from 1 to C. The categorical cross-entropy for the binary representation is given by:

$$\text{CCE}(\mathbf{y}^{\text{true}}||\mathbf{y}^{\text{pred}}) = -\sum_{x} y_x^{\text{true}} \ln \left\{ y_x^{\text{pred}} \right\} \tag{16}$$

Where $x \in 1, ..., M$, and M is the amount of input samples.

The weights of the layers are found by an optimisation strategy. Such a strategy starts at the input nodes and makes one forward pass, i.e., it goes through the whole network from the input to the output layer. After the pass is finished, the loss is measured, and the weights are updated. This is called an epoch. These new weights are then passed back into the model to start the next epoch.

There are many optimisation techniques. The optimisation techniques that are used in this research are gradient descent, Stochastic Gradient Descent (SGD), and Adam. All optimisers use a learning rate η. A small learning rate leads to a more reliable, but slow training process. While a larger learning rate speeds up the training, it could lead to a training process that does not converge.

Gradient Descent is given by:

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta \nabla E(\mathbf{w}^{(\tau)}) \tag{17}$$

Unlike gradient descent, SGD makes updates based on one data point at a time, which could be based on one sample or one batch of samples, instead of the whole loss function. It uses the fact that the maximum likelihood for a set of independent observations of an error function comprises a sum of terms, one for each data point. This sum of terms is given by: $E(\mathbf{w}) = \sum_{n=1}^{N} E_n(\mathbf{w})$. Therefore, the SGD update formula is given by:

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta \nabla E_n \mathbf{w}^{(\tau)}) \tag{18}$$

A further development of the Stochastic Gradient Descent method led to the Adam optimiser, which was first introduced by [Kingma and Ba, 2015]. This optimiser updates the weights based on the first and the second moment of the gradients. Particularly, it calculates an exponential moving average of the gradient and the squared gradient. Moreover, it also treats the learning rate as a per-parameter rate, this means that all weights have their own learning rate instead of them all having the same one. Furthermore, there are two additional rates (β_1 and β_2); they control the decay of the moving averages.

3.3.6 Evaluating a Neural Network

A model aims to minimise the loss function and to make accurate predictions. This is achieved by minimising the loss function of the model. The model learns from the training dataset and tests its generalisability on the test set. A high performance on both the test and the train set indicates that the model performs well. While a high performance on the train set and a low performance on the test set indicates a bad performance. This problem is called overfitting. Overfitting means that a model fails to learn the general structure of the input by modelling the specific features. This happens when a model has significantly more parameters than training samples, i.e., it is too complex. In the context of a deep learning model, a model can become too complex for the input data when it has too many layers and hidden units.

The accuracy and the F1 score are commonly used in classification and object detection tasks to compare and evaluate models. These scores are calculated based on whether the predicted value is equal to the actual value. Hence, it is not used in regression problems, as those allow unequal, but similar predictions as well.

These scores are thus based on these four numbers per class: true positives (tp), true negatives (tn), false positives (fp), and false negatives (fn). Here the positives indicate the samples that are predicted as part of a class, while the negatives are predicted to not be a part of the class. Furthermore, true indicates that the prediction was correct, while false indicates that the prediction was incorrect.

There are two types of classification problems: classifying between one class, e.g., positive, or negative, and multi-class classification, e.g., whether an image holds a cat, a dog, or a hamster. Both types use accuracy and the F1 score to assess the model performance. The accuracy of a one-class problem is calculated by dividing the number of correctly classified (the true positives and the true negatives) samples by the total amount of samples. While the F1 score is calculated by:

$$\text{F1} = \frac{2\text{tp}}{2\text{tp} + \text{fp} + \text{fn}} \tag{19}$$

The accuracy of multiple classes (C) is given by:

$$\text{Accuracy} = \frac{\sum_{i=1}^{C} p_{ii}}{\sum_{i=1}^{C}\sum_{j=1}^{C} p_{ij}} \tag{20}$$

Here, p_{ij} indicates the number of samples that belong to class i, but are classified as class j. A downfall of the accuracy measure is its sensitivity to class imbalance, which arises when the number of samples per class strongly deviates. For instance, if one of the C classes holds 80% of the samples, the model is likely to classify each object as this class. This would give a high accuracy score, while it is not an accurate model. There are several methods to deal with class imbalance, like upsampling the minority classes or downsampling the majority classes. However, instead of changing the balance within the data, other accuracy metrics have been introduced to minimise the influence of the class imbalance. These scores calculate the accuracy per class instead of over the whole sample. The scores are called precision and recall, which are given per class by:

$$\text{Precision}_c = \frac{p_{cc}}{\sum_{i=1}^{C} p_{ic}} \tag{21}$$

$$\text{Recall}_c = \frac{p_{cc}}{\sum_{j=1}^{C} p_{cj}} \tag{22}$$

As can be seen from equation 21 the precision score measures the fraction of correctly classified samples of class c compared to all samples that were classified as class c. Additionally, equation 22 shows that the recall score calculates how often samples of class c are classified correctly compared to all samples in class c.

The F1 score can also be calculated based on the recall and precision scores, namely by:

$$\text{F1}_c = \frac{2 \times \text{Precision}_c \times \text{Recall}_c}{\text{Precision}_c + \text{Recall}_c} \tag{23}$$

However, due to the lack of annotation in unsupervised or semi-supervised learning, the accuracy scores are not a completely valid way of evaluating the model. While unsupervised models are normally only evaluated based on their loss function, semi-supervised models can also be evaluated based on an adjusted accuracy metric. This metric usually gives a lower weight to the false negatives, as they could have been true positives.

3.4 Object detection

One specific task for a machine learning model is Object Detection. Object Detection is the problem of classifying whether an object is present on an image and locating it. Hence, finding an important part of information on an invoice can be generalised to the Object Detection problem. An example of an object could be the invoice number or the invoice date. One approach of object localisation is to use a multiscale sliding window, like [Felzenszwalb et al., 2010]. This sliding window scans the whole image by visiting every pixel. A drawback of this approach is that it is computationally expensive. Hence, lately, Artificial Neural Networks have been used for Object Detection [Bhattacharjee et al., 2020], for example by Convolutional Neural Networks (CNN, section 3.3.3) or Bidirectional Long Short Term Memory (BiLSTM, section 3.3.4).

The evaluation of object detection models depends on the accuracy of the classification and the localisation. While the accuracy of the classification can be calculated by the accuracy, precision, recall, or F1 score, the coordinates cannot. Unlike classification, which has a single output number, localisation requires 4 output values per object. These values are the bounding box coordinates of the location of the object on the image. Regression losses, which are explained in Section 3.3.5, are normally used to assess the prediction quality of these coordinates. However, a more applicable measure to assess the quality of the bounding box is the Intersection over Union (IoU) score. The IoU measures the overlap-to-union ratio between two boxes; it is given by:

$$\text{IoU} = \frac{\text{area of Intersection}}{\text{area of Union}} \tag{24}$$

Some examples of IoU scores are given in Figure 15. The left figure shows a poorly predicted box, while the right figure shows a good fit. The middle figure shows a fit that is neither bad nor good.

Object detection can be done with an unsupervised and supervised training set, however, a supervised set leads to more accurate predictions. In this context, a fully supervised

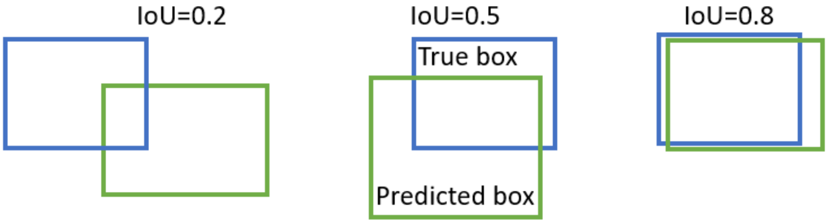


Figure 15: Examples of predicted bounding boxes (green) and the actual bounding box (blue) of the object. The example shows a bad prediction (0.2), an indifferent prediction (0.5), and a good prediction (0.8).

training set provides the label and location coordinates of an object. As was mentioned before, the dataset of this research is unsupervised, but what many papers state is that when a small portion of the data is labelled, it will lead to a much better performance. This sample is mostly called the ground truth, and it is usually hand labelled before training. The three kinds of partially labelled training types are called weakly-supervised, mix-supervised, and semi-supervised. [Nguyen et al., 2019]

Weakly-supervised learning entails that the input dataset does have labels, but these labels are only on image-level. This means that the label just specifies that there is a train on the image, while not locating it. A mix-supervised training set is where both image level-based targets and bounding boxes are used. Whereas semi-supervised sets consist of labelled and unlabelled data; the labels can also be accompanied by bounded box coordinates. [Nguyen et al., 2019]. A visualisation of the four types is shown in Figure 16.

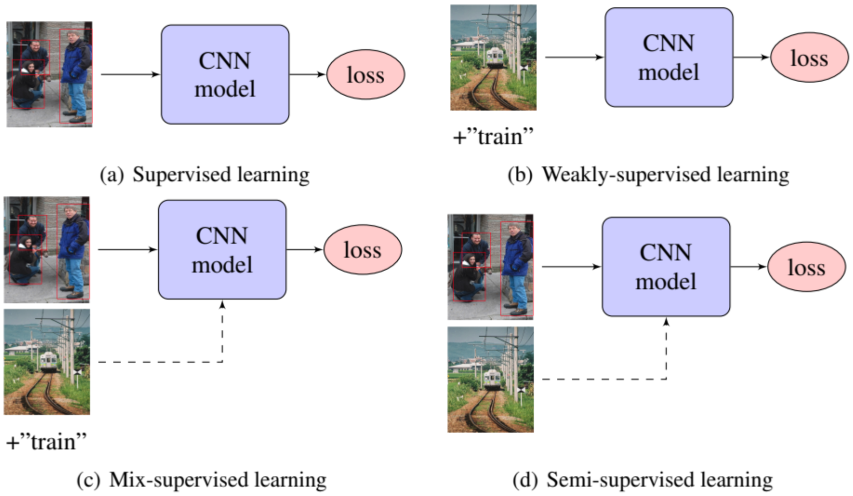
To further improve the object detection in this research, the unlabelled dataset will be labelled before the model training. These labels will be bounding boxes and classes. However, as not all bounding boxes and class labels can be found, this set will never be fully supervised. Therefore, this dataset will be semi-supervised.

3.5 Document Analysis and Recognition

After the object position has been found, the content must be read. This is a problem called Document Analysis and Recognition (DAR), which tries to automatically extract information from paper documents. There are several approaches to reading text from documents, but the three main approaches are *rule-based*, *conventional machine learning*, and *deep learning*. [Marinai, 2008, Xu et al., 2019]

There are two types of rule-based approaches, namely bottom-up and top-down. The difference between these two methods is that in the top-down approach the broad physical structure of the document is assumed to be known, while in bottom-up it is not. Hence, bottom-up methods have to locate the characters as well. Unlike top-down approaches, which already know where to split the documents to find the information. [Marinai, 2008, Xu et al., 2019]

To find the structure, bottom-up methods try to detect the connected components of black pixels on an image. These components are seen as the basic computational units in the



(a) Supervised learning (b) Weakly-supervised learning

(c) Mix-supervised learning (d) Semi-supervised learning

Figure 16: Training settings for Object Detection. (a) Supervised learning (b) Weakly-supervised (c) Mix-supervised learning (d) Semi-supervised learning. The image is taken from [Nguyen et al., 2019]

document images, as their combinations form the characters. The characters are found by combining the connected components into higher-level structures through heuristics. After that, they are labelled according to their structural features. [Xu et al., 2019]

The other two approaches use models like ANN, Support Vector Machines (SVM) or Gaussian Mixture Models (GMM), etc. These models are used to classify the pixels, which are then used for prepossessing, layout analysis, detecting reading order, recognising text, character segmentation, and page and document classification. While conventional machine learning approaches usually have one layer, deep learning uses multilayer Neural Networks to solve these problems. [Xu et al., 2019]

A subclass of DAR is Optical Character Recognition (OCR). OCR is the problem of automatically identifying text from for example images of documents, speech, radiofrequency, etc. In the case of images of a document, which are also called Visually Rich Documents (VRD), an OCR system automatically reads all text on the VRD. In the early stages of OCR, this was done by pattern and template matching. However, as there are a lot of types of documents with many different templates, this became infeasible. Hence, Artificial Intelligence models are used nowadays, especially Artificial Neural Networks. [Eikvil, 1993, Marinai, 2008]

The character recognition process of VDRs is divided into three steps. The first step is called *segmentation and prepossessing*, which identifies and locates the characters on the image and increases the image quality. The second step, *classification*, classifies what the characters are, e.g., which letter or number. This is normally done with a neural network such as (Bi)LSTM or CNN. And the last step, *contextual processing*, is executed to check the recognised results based on their contextual information. [Eikvil, 1993, Marinai, 2008]

In this context, IE is the problem of returning the important information of such a document. This can be quite difficult, as, for example, an invoice could have multiple dates on it, but there is only one correct invoice date.

3.6 Knowledge Acquisition

After extracting the text from a document, the meaning has to be found. This meaning or relation can be found through the Knowledge Acquisition framework. Knowledge acquisition, as described by [Urbani, 2020], is the process of acquiring knowledge from unstructured text; the process is shown in Figure 17.

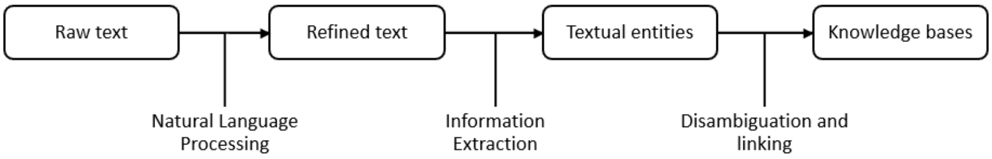


Figure 17: Example of the Knowledge acquisition diagram. This example shows the steps that are taken to acquire knowledge.

The process starts with raw text, which is then fed into a Natural Language Processing (NLP) framework to obtain refined text. After that, the refined text is used to find information such as names or relationships. Lastly, the information is put into a Knowledge Base (KB). A KB resembles a library; it translates the factual knowledge into associations between entities and relations. Recently these Knowledge Bases have been expressed as Knowledge Graphs (KG) [Janev et al., 2020], an example of a relationship between entities in a Knowledge Graph format is depicted in Figure 18.

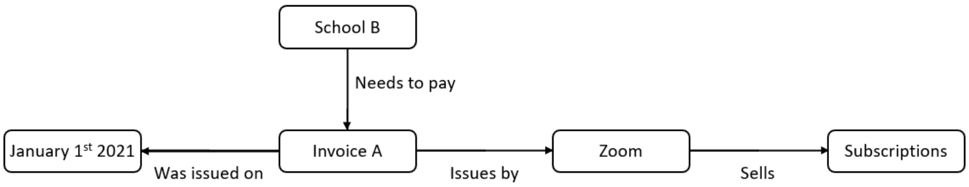


Figure 18: Example of a Knowledge Graph. It shows the linking between the entities through their relations.

The first step of the acquisition is the NLP framework. This framework consists of 5 steps. The first step is called tokenisation and it splits up a character sequence into tokens (words, terms, or entities). The next step is either stemming or lemmatisation. Stemming reduces a token to its root, i.e., it removes all unnecessary suffixes of a word. Whereas lemmatisation reduces a token to its base form, i.e., its dictionary form. The difference between these techniques is that the output of lemmatisation is always a word, while the output of stemming might not be a word. An example of lemmatisation and stemming is the token '*studies*'; its suffix is '*es*', hence the stemming output is '*studi*', while the lemmatisation output is '*study*'. The next step is stopword removal, which removes all tokens that are considered stopwords. Examples of stopwords are of, in, by, a, and an. They are removed due to their lack of information gain when trying to find relationships between words. After that, all left-over tokens are tagged with a POS tag (Part of Speech tag). These tags indicate the type of a token, which can be a functional word or a content word. A functional word is a word that ensures that a sentence is grammatically correct, while a content word carries the meaning of a sentence. The last step is the parsing step, this step constructs a tree that represents the syntactical structure of the sequence of tokens.

As NLP parses complete sentences, it is less applicable to VRDs if they are filled with incomplete sentences. As invoices are such documents, which have sentences like “Invoice number: 2343”, that does not have a verb, basic NLP packages like NLTK (by [Bird et al., 2009]) and SpaCy ([Honnibal and Montani, 2017]) are less applicable.

The second step of the acquisition is called IE. There are two types of IE, namely Named Entity Recognition (NER) and Relationship Extraction (RE). NER tries to find and classify names in the text, which could be names of persons, companies, dates, locations, etc. Whereas RE tries to find semantic relationships between two or more entities, examples of semantic relationships are married, employed by, or lives in.

The last step is to disambiguate or link the entities (or relations) to each other. This step concatenates all previous information into one knowledge base.

4 Related work

Several researchers have tried to solve the invoice IE problem, which led to many different approaches. The first methods that were introduced were Rule-Based template matching methods, like [Riloff, 1993, Kim and Moldovan, 1993]. These methods pre-defined the templates of the invoices and used OCR to read the pre-classified fields.

The current most famous OCR engine is Tesseract, which was originally developed by Hewlett-Packard, but it was taken over by Google in 2006. Tesseract [Google inc, 2019] is an open-source OCR package that reads all characters and/or words from an image and places a bounding box around them. The model that is used to read and classify these characters is a Long Short Term Memory Neural Network (LSTM). Tesseract has an open-source Python library called `pytesseract`, which is an easy and fast OCR tool.

A drawback of solely using OCR, templates, and rules is that they are not generalisable. Moreover, they are not only highly dependent on the preassigned template, but they are also highly dependent on the intrinsic dependency of the language of the model, such as grammar, sentence direction and structure, dependency of words, etc. For instance, European countries use a ‘,’ to indicate a decimal point in a number, while English-speaking countries use a ‘.’. Unlike these methods, machine learning methods do not depend on the language or a template of an invoice to locate a specific field. Hence, recent approaches adapted Deep Learning models. Although the Deep Learning approaches tend to perform much better, as they base their classification on the whole input instead of solely looking at the separate word boxes, some researchers have also used traditional classification methods such as decision trees, K-Nearest-Neighbours, Naive Bayes, etc, to solve the problem [Tarawneh et al., 2019].

Many of the Deep Learning approaches are entries of the ICDAR 2019 Challenge on "Scanned receipts OCR and key information extraction" (SROIE), by [Huang et al., 2019]. Although the competition ended in May 2019, many new invoice and receipt IE models still use the SROIE dataset to test their improvements. The problem of this competition is divided into three tasks: 1. Text localisation, 2. OCR, 3. Key information extraction, where most approaches follow the structure shown in Figure 19.

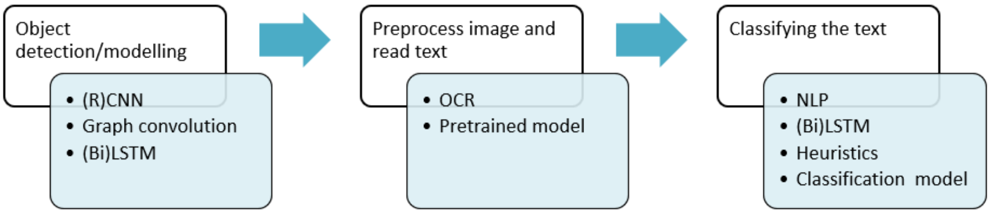


Figure 19: Main structure of related modelling techniques.

The highest scoring entries in the SROIE competition are based on NLP and Deep Learning techniques, which are explained in Section 3.6 and 3.3, respectively. NLP techniques specifically target text, while Deep Learning techniques target text and image features.

The Deep Learning approaches are divided into two groups: traditional Neural Networks and Transformers, which are both described in Section 3.3. Many different Neural Networks can solve the invoice IE problem. Some try to model the invoice as a graph [Liu et al., 2019a, Rastogi et al., 2020, Yu et al., 2020]. For instance, the Graph Convolution model by [Liu et al., 2019a], which had an F1 score of 87.3%. This model finds a graph representation of the text segments in a document. Where the text segmentation is composed of the text itself and the text position on the document, which are both generated by an OCR engine. The text segments are then modelled as the nodes of the graph, while the edges represent the visual dependencies, like relative shape or distance, between the nodes. The model then uses graph convolution to learn the structure of the invoices. The full model architecture is described in [Liu et al., 2019a].

Others try to solve the problem by using object detection models like (Faster-) RCNN [Ghosh, 2021, Jun et al., 2019, Shi et al., 2015, Zhao et al., 2019] or BiLSTM [Jiang et al., 2019, Patel and Bhatt, 2020, Schuster and Paliwal, 1997]. One of these models, which is called *Convolutional Universal Text Information Extractor* (CUTIE), by [Zhao et al., 2019], was introduced after the competition. This approach uses a CNN architecture to perform object detection. However, instead of using the invoices as the direct input of the model, they perform some preprocessing steps to form a gridded text, where a grid is a matrix representation of the pixels of an image. Normally, the input consists of three grids, one for each RGB channel. A gridded text is also a pixel representation of the image. However, instead of numbers, the matrix holds the text or the text embeddings of the image.

The input of the CUTIE model is made by first using an OCR engine to read and find the position of the text on the receipts. The text is then positioned in a grid that resembles the receipt or invoice. This ensures that the spatial and contextual information is preserved. After that, the text is embedded by an embedding layer, which forms the final input. The resulting input is a grid that contains scattered data points, where the words are closer together in some grids and farther apart in others. As a result, Zhao et al. decided to use Spatial Pyramid Pooling and atrous convolutional layers to preserve the multiscale context. A complete description of the architecture can be found in [Zhao et al., 2019]. Zhao et al. concluded that their performance gain was mainly achieved by using the atrous layers, text embeddings, and the grid positional mapping, instead of solely using the visual features of the receipt itself. The model has an F1 score of 86.7%.

However, a drawback of the object detection approach is that most of the objects are small. The small sizes lead to having multiple objects in one pixel, which causes inaccurate results. A solution to this could be to increase the dimension of the input images. However, having large images leads to a large dataset and a lengthy training process. Hence, some researchers proposed to first look for larger fields in the invoice that hold the information, like an article table or a paragraph of information about the seller, instead of immediately localising the specific fields [Ghosh, 2021, Hoque et al., 2020, Kazdar et al., 2019, Acuña et al., 2019]. Here the regions are found by an object detection model like Faster-RCNN, which classifies and localises the tables and the paragraph. The text in a region is then processed by an OCR engine and classified by entity-relationship mapping using Natural Language Processing techniques, or by using pre-defined heuristics [Brauer et al., 2011].

One of the most commonly known RCNN object detection models is You Only Look Once (YOLO), which was introduced by [Redmon et al., 2015]. This model locates the objects by reframing the problem as a regression problem instead of just a classification model. The regression problem tries to find the right class label by using the pixels and the bounding box coordinates of the object. Here the bounding boxes are found by a single CNN, which simultaneously predicts multiple bounding boxes and their class probabilities.

The YOLO architecture, which is depicted in Figure 20, consists of 24 convolutional layers and 2 fully connected layers. The convolutional layers extract the features from the input, while the fully connected layers predict the class probabilities and the bounding box coordinates. These bounding boxes are made on the pre-specified S by S grid space of the image, where the model predicts B bounding boxes for each grid cell. Additionally, the model predicts the confidence and C class probabilities for those boxes. The predictions of the model are encoded in an $(S \times S \times (B \cdot 5 + C))$ tensor.

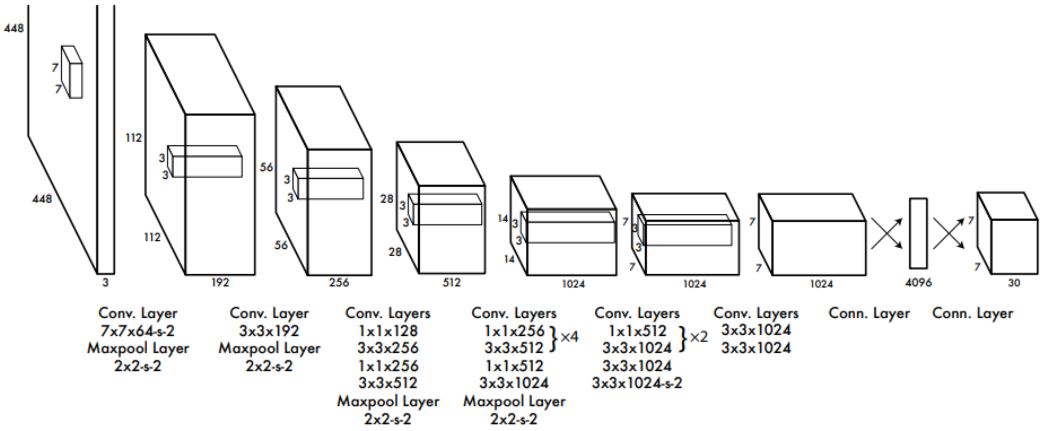


Figure 20: YOLO CNN architecture, taken from [Redmon et al., 2015]. The paper its input are images of size $(448 \times 448 \times 3)$, and the output is an $(7 \times 7 \times 30)$ tensor, whit $S = 7$, $B = 2$, and the number of classes to predict $C = 20$.

As was stated in the Background section, (Faster-)RCNN models use a pre-trained model to find the structure of its input. Many of these models were entries to a very commonly known competition for image classification, namely, the ImageNet Large Scale Visual Recognition Challenge (ILSVRC). A lot of models that perform very well in this competition are CNN models. One of these winning models is called AlexNet, which was first introduced by [Krizhevsky et al., 2012]; it is also described in Section 3.3.3. This model is one of the more accurate models that use significantly fewer layers than other winning models.

Unlike exclusively using image features, many researchers have concluded that using both image features and text features in a Deep Learning architecture will lead to the most accurate results: [Xu et al., 2019, Hong et al., 2021, Garncarek et al., 2020, Powalski et al., 2021]. They particularly mention the importance of using spatial and contextual information, as many other approaches tend to focus on just extracting the information from the text.

Most of these models use a well-known language embedding tool, BERT, by [Devlin et al., 2018], to create their text embedding features. BERT is a pre-trained language modelling transformer, and it is used in a lot of IE approaches as a backbone; it is described in

Section 3.3. Many NLP tasks can be solved by fine-tuning the BERT embedding model, e.g., sensitivity analysis, question answering, and IE. Some models that use this embedding combined with image features are [Xu et al., 2019, Garncarek et al., 2020, Shi et al., 2015, Liu et al., 2019b]. They show that a strong text embedding significantly increases the performance.

One of these models is the LayoutLM model, introduced by [Xu et al., 2019], which has an F1 score of 96.04%. In addition to using the BERT embedding, LayoutLM incorporates two more features: Document Layout Information and Visual Information. Document Layout Information concerns the relative position of words in the document, while Visual Information concerns the visual features of the image, such as the whole image. The Visual Information could indicate a layout, or word-level visual features such as underlining or boldface, which could indicate importance. This results in three input embeddings: the BERT word embedding, the positional embedding, and the image embedding. The text and positional coordinates are acquired by an OCR tool, like Tesseract.

Like the BERT model, the LayoutLM model is first pre-trained and then fine-tuned for a specific task. The model is pre-trained to perform two tasks: MLM and Multi-label Document Classification. After pre-training the model, the model can be fine-tuned. The more relevant fine-tuning tasks for this research are receipt template classification and document classification. A complete explanation about the architecture can be found in [Xu et al., 2019].

In contrast to the SROIE dataset, the dataset in this research is not perfectly annotated. Hence, many models would have significantly worse performances if they would be fine-tuned for this research. Therefore, the object detection to information extraction approach was chosen for this research, instead of training a pre-trained language model. Furthermore, several heuristics, like in [Brauer et al., 2011, Riloff, 1993, Kim and Moldovan, 1993], have been implemented to ensure a better performance. Another approach, using classification models like decision trees, is also experimented with as they could also enhance the performance as well. Some researchers, like [Ju et al., 2019, Zisou et al., 2020], have successfully combined an (R)CNN and a LightGBM classifier to get better results than using a single model. This leads to three different approaches, a Rule-Based model, a decision tree-based algorithm, and an Object Detection model that could be accompanied by a Rule-Based model or a decision tree. Instead of a Rule-Based approach, many researchers opt for an NLP approach, however, as most of the text on an invoice is not part of a consecutive sentence, this will not perform well. Hence, this approach is not experimented with.

Similar to the contestants of the SROIE competition, this research also follows the structure in Figure 19, as it led to outstanding results. The model description can be found in Section 6.

5 Data description and preparation

As was mentioned before, SoliTrust has a lot of data at its disposal. All their clients upload their data to the datamodel, which is then transformed into the fixed database format. The provided invoices are recorded in this database as well. The invoices and the database counterparts are used to prepare the create the input of the models. While some models only require an image of the invoice, others require an annotated image or a text (type) map. The data and its preparation are described in this section.

5.1 The provided data

The provided data is split into two parts: databases and PDF invoices. Both are described in their respective sections.

5.1.1 The Databases

The SQL databases hold many different tables that describe some part of the business process of a client. However, many of them are not relevant for this research, the tables that will be used in this research are:

- Information about the invoices
- Information about the known creditors of a company
- A table that holds tax codes, which are linked to tax percentages
- A table that holds information about the PDF file that is licked to a certain invoice
- Information about sales invoices

In addition to the tables, the invoices themselves are needed to train a model. However, most invoices are either deleted or not saved after they are recorded into a database. Hence, most of the databases do not apply to this research. Two clients that did provide PDFs are Company A and Company B. Company A is a middle school and Company B sells and catches fish. Hence, their databases are used in this research. Since Company A is a school, they (normally) do not have to pay taxes (VAT, BTW in Dutch). As a result, the columns that describe the tax percentage and tax amount are mostly empty, even when the invoice itself states a tax amount. A further explanation about the fields in the database is given in Appendix A.

23809 invoices were made available for this research. 2312 of those were provided by Company A, 18 were provided by SoliTrust and 21479 were provided by Company B, where 11564 are creditor invoices and 9915 are sales invoices. Since all sales invoices have the same layout, only 400 of them are used, otherwise, the model could lose generalisability.

To use the invoices, they must have a database counterpart. However only the ones that were provided by Company A have a PDF file field that links the file name to the tables in the database. The invoice table and the file table are linked with the file ID key. The others are not immediately linkable. As a result, the invoice number must be extracted from the invoice before it could be linked to its database counterpart. The linking and labelling left 10111 invoices to train on. However, 6406 of them are the sales invoices of Company B. Hence, 4105 invoices were left to train on. The training set is split into a train (80%) and a test set (20%), the train part is also split into a validation (20%) and a train set (80%). A representation of the distribution of the data can be seen in Figure 21.

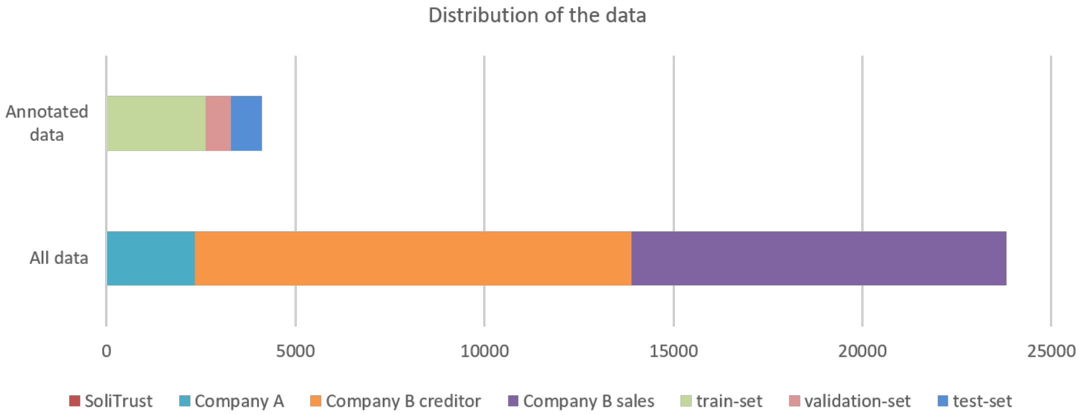


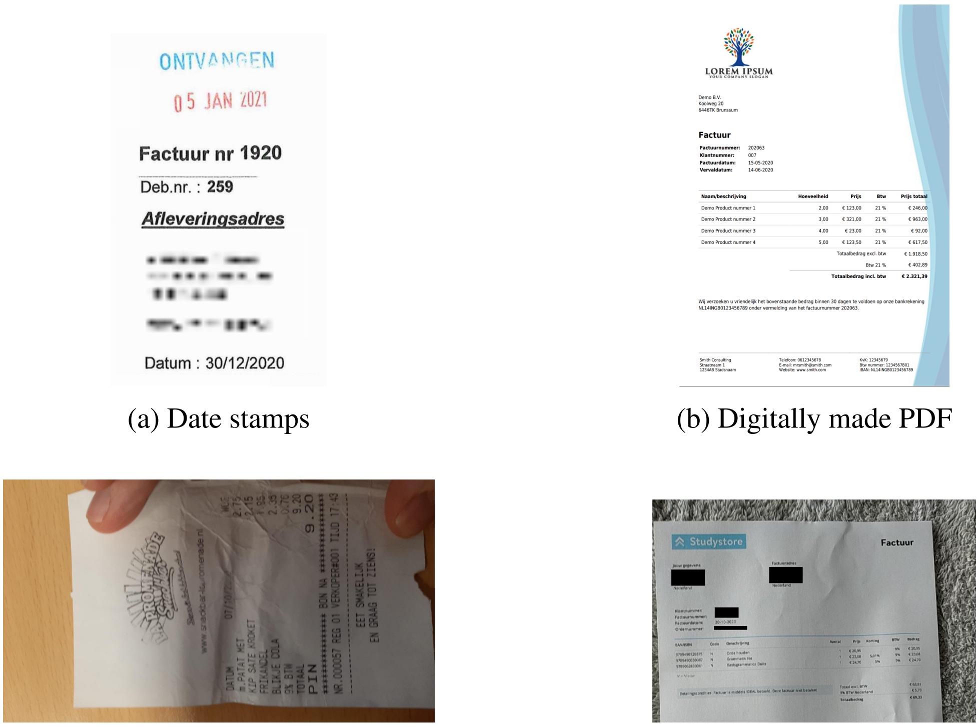
Figure 21: Distribution of the provided data in percentages. All data shows the initial distribution of the 23809 invoices, while Annotated data show the distribution of the leftover 10111 invoices.

Most of the information that is on an invoice is recorded in the invoice information table. This table includes data such as invoice date, total amount, and external reference. However, most columns are either empty or partially filled in. One of these partially filled in columns is the currency column. As most of the invoices are issued in euros, some companies, including Companies A and B, do not register the currency of the invoice when it is issued in euros. A further description of the tables is shown in Appendix A.

Some of the columns that are not filled in are columns that are fixed in another table. For instance, the columns creditor name, creditor bank account number, and creditor address. Instead of recording these fields, the field that holds a creditor ID is filled in. With this filled-in field, an invoice can be linked to the information that is already known and fixed in the creditor table. Similarly, some invoice entries hold a tax code instead of a tax percentage or amount. The tax code can then be used to link to the tax codes table, which includes the tax percentage that belongs to a certain tax code. However, in some instances, neither the fields nor the linkable IDs were filled in.

5.1.2 The invoices

The provided data consists of unlabelled PDFs of invoices. Most of the invoices are digitally made and send, while some have been printed and scanned. The send invoices are usually of good quality, which makes it easier to read them. The scanned invoices are more difficult to read, due to their additional noise. Some of the noise that has been encountered is shown in Figure 22. Figure 22a shows an example of noise that is widespread among invoices, the date stamp. A lot of companies record the date of receival ("ontvangen") instead of the date that is stated on the invoice itself. Figure 22b shows a digitally made invoice with a colourful background. The colourful background causes less contrast between the background and foreground. This could lead to less accurate predictions or even no predictions at all. Figure 22c shows a rotated image that is taken with a phone, it has a fold, a shadow, and some background noise. The last sub-figure, Figure 22d, shows a slightly skewed and very pixelated invoice. All these types of noise make it harder to read the information on the invoice and to match it to the database counterpart.



(a) Date stamps

(b) Digitally made PDF

(c) Picture of a bill

(d) Badly photographed invoice

Figure 22: Examples of noisy data

An invoice holds a lot of information, but not all information is relevant or recorded into the database. The important fields (the classes) in this research are shown and described in Table 2.

As most of the information on an invoice is unimportant, the dataset is a severely imbalanced. The class distribution is shown in Figure 23. The left figure shows that class 30, which is the "background" or "not important" class, has significantly more samples than the other classes. The figure on the right shows the number of samples of the classes without class 30. As can be seen, the second most represented class (class 6, *Inkoopfactuur_aantal*) has around 1500 samples, which is less than 1% of the samples of class 30. In total, there are 178028 samples, where 167308 are part of class 30, which is 94%. The balance within the test set is most likely similar to that of the train set.

Table 2: Descriptions of fields in an invoice. All fields that start with _ have a database table name in front of them.

Field name	Description
_externereferentie	This is the invoice number, which every invoice should have
_factuurdatum	This is the invoice date. All invoices have one invoice date Although multiple dates could be on the invoice, only one of them is the invoice date.
_Artikel_code	This field represents the code of a purchased article.
_aantal	This field indicates how many items are purchased of one article. This field can have multiple entries on one invoice.
_prijs	This field represents the price of one product per article. This is always included.
_omschrijving	This field holds the description of a purchased product. This field is normally included.
_brutobedrag	This field represents the total amount per article before discount and taxes. This is always included.
_korting	This field represents the discount, which could be per article or a total amount. This field is rarely filled in.
_nettobedrag	This represents the total before taxes, which is always included.
_btwpercentage	This field shows the tax (BTW) percentage. This field is normally on every invoice. However, some products or organisations are exempt from taxes.
_btwbedrag	This filed represents the tax amount, which is normally included.
_totaalbedrag	This is the total payable amount, which is always included.
_valuta	This field shows the currency of the payable amount, which is always included.
_contractperiode_van	This field shows whether there is an article that has a subscription and when that subscription started. This field is not on every invoice.
_contractperiode_tot	This field shows the end date of a subscription. It is also not on every invoice.
_bankrekening	This is the bank account number of the creditor, which is normally on every invoice.
Crediteurnaam	This is the name of the creditor, which is normally on every invoice as well.
Woonplaats	This is the place of residence of the creditor, which is normally on every invoice as well.
Crediteur_kvk	This is the registry number of the creditor, in the Netherlands it is called KvK, but other registry numbers are also filled into this field. this field is normally on every invoice as well.
e-mail adres	This is the e-mail address of the creditor, which is normally on every invoice as well.
Website	This is the website of the creditor, which is normally on every invoice as well.
Totaal_prijs	This field specifies the total amount per purchased article.
BTW code	This is the tax code of the selling company.
prijs	This field specifies the price of a purchased product.
_btwpercentage_product	This field specifies the tax percentage per purchased article.

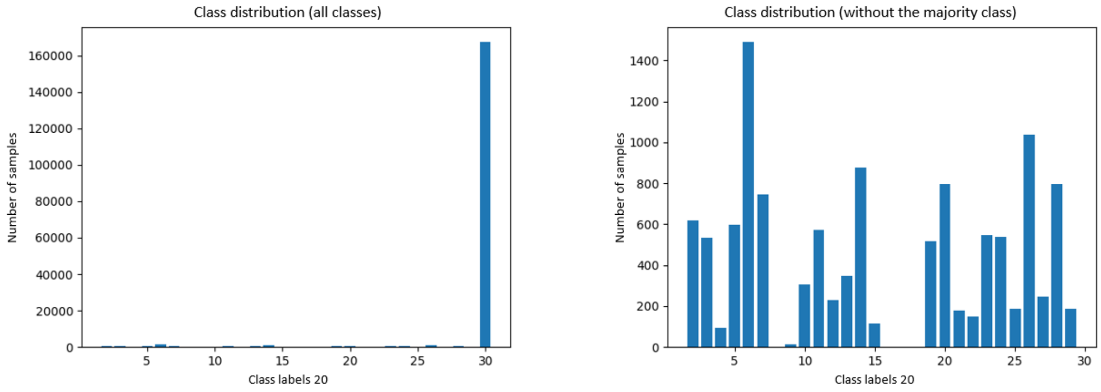


Figure 23: Class imbalance of full dataset

5.2 Preparing the data

The first step of the research is to prepare the data, which consists of preprocessing and annotating the images. These steps are inspired by these previous image preparation steps for Object Detection and Information Extraction: [Google inc, 2019, Google inc, 2021, Zelic and Sable, 2020].

5.2.1 Preprocessing

As was mentioned before, the data consists of PDF invoices and their database counterparts. The PDFs cannot be fed into a model, they first have to be transformed into a grid-like topology, like an image (JPEG format). This must be done since the JPEG format holds the RGB representation of the images, which can be fed into the model as a matrix of numbers. The module called `fitz` in the package `PyMuPDF` by [McKie and Liu, 2016], is used to convert the PDFs to JPEG format. This is done by creating three pixelmaps, one for each RGB channel, for each page in the PDF; with the function `getPixmap`. All these pixel maps are then concatenated to create the pixel map of the whole PDF.

The next step entails preprocessing the data. This step ensures that the OCR system produces more accurate reading results. However, as most of the invoices in this research are digitally made and sent, most of these preprocessing steps do not have to be executed to get the right OCR results. Hence, the preprocessing steps are only executed when the OCR engine is not able to read anything or only some parts. One crucial part is to ensure that the invoices and their text lines are not skewed. Hence, the python library `deskew`, by [et al., 2019], is used to rotate the images. First the function `determine_skew` is used to find the angle of rotation, which is then passed to the function `rotate`, which rotates the images.

Moreover, as the accuracy of an OCR tool relies on the resolution of an image, the contrast between background and foreground, and the amount of noise, they should satisfy some constraints. According to the Tesseract sources, their module works best on images with a DPI (dots per inch) of at least 300 [Google inc, 2021]. A resolution of 2480×3508

pixels is needed to ensure a DPI of 300 for one sheet of paper with a standard A4 size [Davies, 2021]. The `fitz` module is used to ensure that the DPI of the images is large enough. This is done by adjusting the `matrix` parameter in the `getPixmap` function; this parameter determines the zoom factor of the function. The standard zoom function ensures a DPI of 72, hence using a zoom matrix of $(300/72, 300/72)$ on an 8.5×11 image leads to 300 times more pixels in both width and height, namely a size of (2550×3300) and a dpi of 300.

However, a DPI of 300 does not ensure that there is no noise on the image. A common technique to remove noise from an image is called blurring. Blurring removes outlier pixels from the image while leaving most of the image intact. Four of the most used blurring techniques are average blurring, Gaussian blurring, median blurring, and bilateral filtering; the techniques are shown in Figure 24. All methods work with an $f \times f$ kernel and replace the central element of the kernel based on the criteria of the method. This criterion is different for each method. While average and median blurring simply replace the central element with the average or the median of the elements under the kernel, Gaussian blurring and bilateral filtering are more complex. For instance, the kernel size of the average and median blurring is fixed, while it is not fixed in Gaussian blurring and bilateral filtering. Their kernel sizes can be changed during the blurring process. The size of the kernel is determined by the desired standard deviation of the values within the kernel. Additionally, while the other methods tend to smooth the edges; bilateral filtering is known for removing the noise without smoothing the edges.

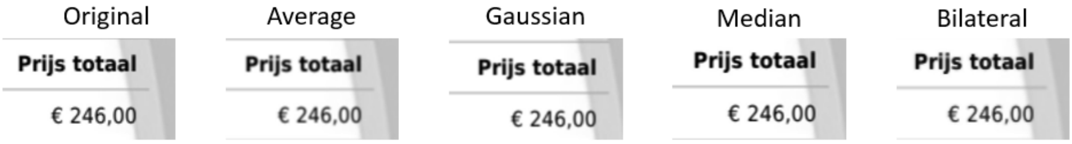


Figure 24: Zoomed in part of the image after blurring. The examples of the blurring techniques are tested on a digitally made invoice. The first figure shows the original image. The second figure shows the image after Average blurring. The second figure shows the image after Average blurring. The third figure shows the image after Gaussian blurring. The fourth figure shows the image after Median blurring. The fifth figure shows the image after Bilateral filtering. All techniques are realised with the `OpenCV` library.

As can be seen from the zoomed-in images, they all seem similar. However, according to the OCR package, they are not; the results are shown in Table 3. The table shows that after blurring the image, only median blurring produced readable letters. Hence, when an image does not provide good OCR results median blurring is used to improve the output.

Table 3: OCR results after blurring, where NULL indicated that no text was found.

	Original	**Average**	**Gaussian**	**Median**	**Bilateral**
Text	Prijs Totaal €246,00	NULL	NULL	Prijs Totaal € 246,00	NULL

Another issue that could lead to less accurate reading results is a colourful background, like in Figure 22b. A technique that is used to get rid of this is called thresholding. Thresholding is a technique that binarises an image, i.e., it converts the image to black (1) and white (0). The threshold value decides whether a pixel is turned black or white. There are many algorithms that determine the thresholding value, which are divided into global and local techniques. The global thresholding techniques pick one threshold value for the whole image, while local thresholding picks a different value for each specific region.

Three of the most commonly used thresholding techniques are Global Thresholding, Adaptive Mean Thresholding and Adaptive Gaussian Thresholding. Where adaptive techniques use the standard deviation to determine the kernel size, instead of a fixed number. The three methods are shown in Figure 25, they are realised by the python library `OpenCV`, introduced by [Bradski and et al., 2021]. Seven more techniques are shown in Figure 47 in B, these are generated with the function `skimage.filters.try_all_threshold` from the library `scikit-image`, by [Van der Walt et al., 2014]. The methods that seem to have the best results are `Global Tresholding`, `Isodata`, `Minimum`, and `Yen`. However, according to [Sezgin and Sankur, 2004, Trier and Jain, 1995], who compared several thresholding techniques, local adaptive thresholding techniques are the best thresholding techniques for document images. They concluded that the thresholding algorithms Niblack by [Niblack, 1986], and Sauvola by [Sauvola et al., 1997] performed best, which are local adaptive methods; full explanations can be found in their respective articles. The results that are produced by the Niblack and Sauvola algorithms are shown in Figure 26.

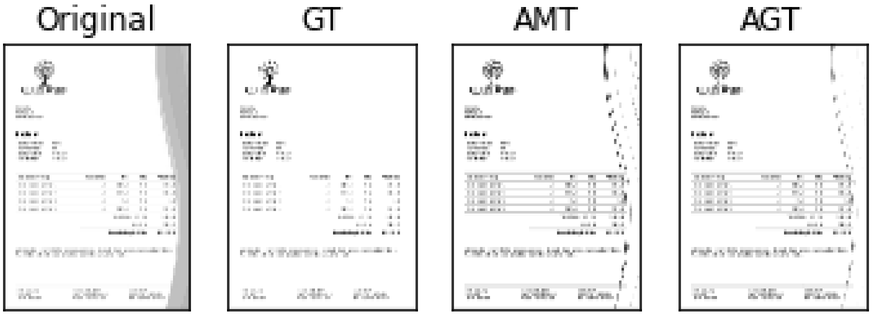


Figure 25: Thresholding techniques in `OpenCV` module. GT: global thresholding by `cv2.threshold()`. AMT: adaptive mean thresholding by `cv2.adaptiveThreshold( cv2.ADAPTIVE_THRESH_MEAN_C, (...))`. AGT: adaptive gaussian thresholding by `cv2.adaptiveThreshold( cv2.ADAPTIVE_THRESH_GAUSSIAN_C, (...))`

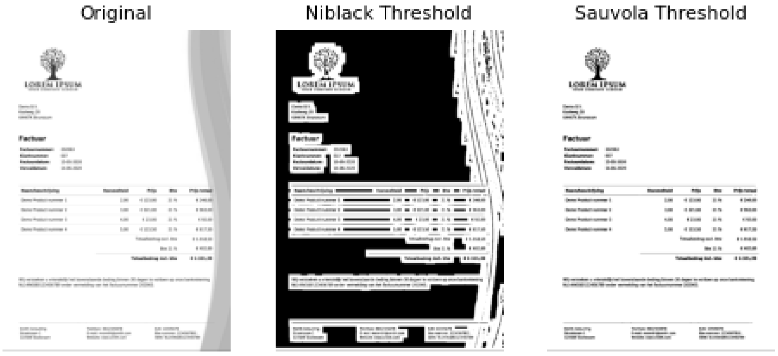


Figure 26: Thresholding results by Niblack and Sauvola, by the functions `skimage.filters.threshold_niblack` and `skimage.filters.threshold_sauvola` from the python library `skimage`.

The thresholding methods are also compared based on their OCR results, which are shown in 4. The table shows that the techniques that keep the border of the background perform worse, as the border is considered as an additional character. Even though the OCR system was able to read the text from the image, it also created extra characters. Hence, due to the accurate OCR results and the prior research conclusions by Sezgin and Sunkur, and Trier and Jain, the Sauvola thresholding method is used in this research when the original image is not readable.

Table 4: OCR results after thresholding the image, where NULL indicates that no text is found.

	GT	AMT	AGT	Isodata	Li	Mean
Text	Prijs Totaal € 246,00	Prijs Totaal: 7 € 246, 00 J	F\q i | € 246,00 |	Prijs Totaal € 246,00	Prijs Totaal € 246,00	Prijs Totaal € 246,0

	Minimum	Otsu	Triangle	Yen	Niblack	Sauvola
Text	NULL	Prijs Totaal € 246,00 |	NULL	Prijs Totaal € 246,00	mm Prijs Totaal mmm € 246,00	Prijs Totaal € 246,00

5.3 Annotating the data

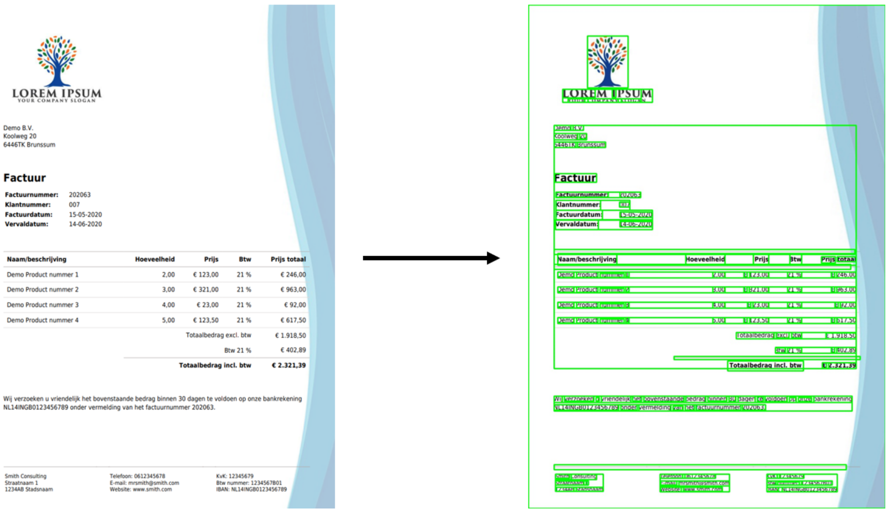
The next step is to prepare the target values of the model. As these are not provided, the images must be labelled. First, the specific fields are labelled, e.g., the invoice date, invoice number, etc. These fields are needed to test and train the models. As the RCNN model works with regions, after the annotation of the fields the regions also have to be located and labelled. The annotation is executed by a rule-based algorithm and the python OCR package pytesseract. This is the Python version of tesseract, by [Google inc, 2019].

5.3.1 Annotating the specific fields

As was shown in the previous section, there are 25 classes (Table 2 and a background class). These classes are annotated by reading the text from an invoice and labelling the produced boxes. First, the invoice is processed by the OCR engine to find the words and their bounding boxes; a representation is shown in Figure 27. After that, some additional steps are taken to merge the bounding boxes. There are several merge types. The first merge function merges boxes that are close to each other in a line, as they could be one entity, e.g., the address “Koolweg 20". The second type of merge is called the ’description merge’, this function checks whether the descriptions are split over multiple lines. For instance, if ‘Demo Product nummer 1’ had an additional line under it like ‘purchased on the 13th of May 2020’. The third function merges all boxes that are in the same line; these are used to find the article lines.

After that, all boxes (merged and not merged) are compared to the fields in the database. If a text field is equal to a field in the database, then it is immediately labelled, e.g., the invoice number of this invoice is 202063, so the box which holds “202063" is labelled as *Inkoopfactuur_externereferentie*.

The Rule-Based algorithm is based on the database counterpart of the invoices and a



LOREM IPSUM
YOUR COMPANY SLOGAN

Demo B.V.
Koolweg 20
6446TK Brunssum

Factuur

Factuurnummer:	202063
Klantnummer:	007
Factuurdatum:	15-05-2020
Vervaldatum:	14-06-2020

Naam/beschrijving	**Hoeveelheid**	**Prijs**	**Btw**	**Prijs totaal**
Demo Product nummer 1	2,00	€ 123,00	21 %	€ 246,00
Demo Product nummer 2	3,00	€ 321,00	21 %	€ 963,00
Demo Product nummer 3	4,00	€ 23,00	21 %	€ 92,00
Demo Product nummer 4	5,00	€ 123,50	21 %	€ 617,50
		Totaalbedrag excl. btw		€ 1.918,50
			Btw 21 %	€ 402,89
		Totaalbedrag incl. btw		**€ 2.321,39**

Wij verzoeken u vriendelijk het bovenstaande bedrag binnen 30 dagen te voldoen op onze bankrekening NL14INGB0123456789 onder vermelding van het factuurnummer 202063.

Smith Consulting
Straatnaam 1
1234AB Stadsnaam

Telefoon: 0612345678
E-mail: mrsmith@smith.com
Website: www.smith.com

KvK: 12345679
Btw nummer: 1234567B01
IBAN: NL14INGB0123456789

Figure 27: Image of an invoice after being processed by the function `pytesseract.image_to_data`. This method reads all the text as separate words and returns the text and its location (bounding box coordinates).

`get_type` method that gives a type to a word or sentence. The method reads the text in the box and assigns a type; the possible types are shown in Table5.

As the database does not hold all information on the invoice, some additional rules, which are based on the aforementioned types, are used to find all the important fields. Some fields that are not in the database are website, e-mail address, BTW-number, IBAN, and registration number (KvK). Unlike the websites, e-mail addresses, BTW-numbers, and IBANs, which are found by using their assigned types, the registration number could not be annotated like this. Instead, a Google search query was used to retrieve some of the registration numbers from the web. The recovered registration numbers, which were around 1000, are then used to find the counterpart in the invoice.

Furthermore, most databases do not record the specific prices and quantities of the purchased articles. Instead, they record the total amount. Hence, these fields had to be found by creating a rule that found the article lines. This rule specifies that when a line holds a quantity, description (or a combination of words and numbers) and at least one price then it could be an article line. The line is then split into numbers, prices, and a description. The description is immediately labelled as *Inkoopfactuur_omschrijving*; the annotation of the other fields follows some additional rules. Normally there is one number in an article line, hence the first or only number is labelled as *Inkoopfactuur_aantal*. The first price in the line is labelled as *Inkoopfactuur_prijs* and the last price in the line is labelled as *totaal_prijs*. The other prices, which could be a discount or tax, are ignored, as invoices can hold neither, both or either of them, it is impossible to hardcode this label. Similarly, percentages are also ignored. As all invoices have their own article line template, the method is not robust. However, most invoices in the given dataset adhere to these rules. The process of finding the article lines is depicted in Figure 28.

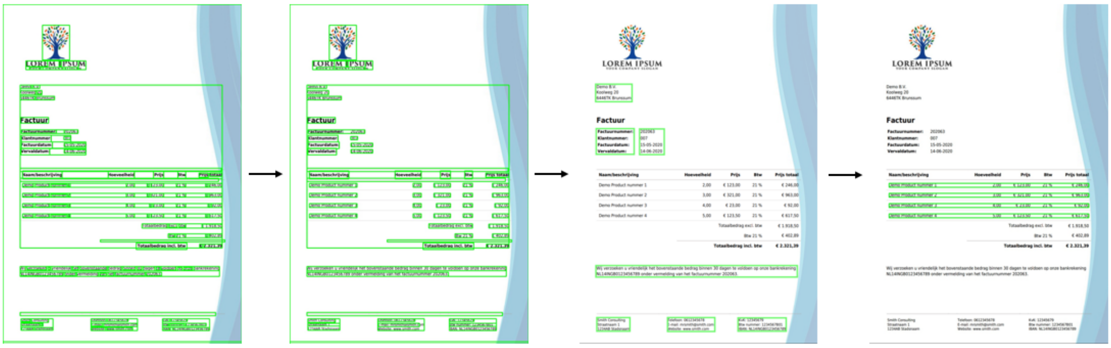
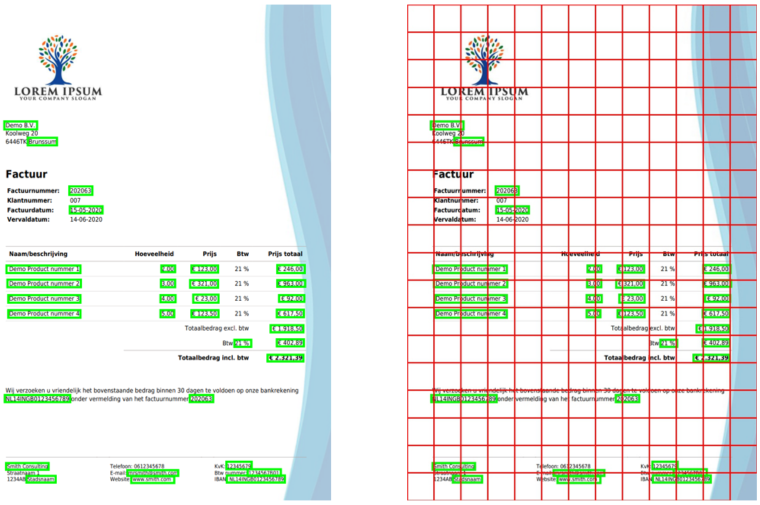


Figure 28: Depiction of the merging to find the article lines and the specific fields. The first image shows the boxes that are generated by tesseract. The second image shows the first merge step, where words are very close (one space apart). The third image shows the merging of close lines, as they could be part of one description. The last image shows the article lines.

5.3.2 Annotation important regions

After all these steps all specific objects are annotated, a representation is shown in Figure 29. However, as the feature space of the model is fairly small, which is shown in the right image in Figure 29, some objects end up in the same pixel. Classifying an object within such a pixel is difficult, as it belongs to multiple classes.



LOREM IPSUM
YOUR COMPANY SLOGAN

Demo B.V.
Koolweg 20
6446TK Brunssum

Factuur

Factuurnummer:	202063
Klantnummer:	007
Factuurdatum:	15-05-2020
Vervaldatum:	14-06-2020

Naam/beschrijving	Hoeveelheid	Prijs	Btw	Prijs totaal
Demo Product nummer 1	2,00	€ 123,00	21 %	€ 246,00
Demo Product nummer 2	3,00	€ 321,00	21 %	€ 963,00
Demo Product nummer 3	4,00	€ 23,00	21 %	€ 92,00
Demo Product nummer 4	5,00	€ 123,50	21 %	€ 617,50
			Totaalbedrag excl. btw	€ 1.918,50
			Btw 21 %	€ 402,89
			Totaalbedrag incl. btw	**€ 2.321,39**

Wij verzoeken u vriendelijk het bovenstaande bedrag binnen 30 dagen te voldoen op onze bankrekening NL14INGB0123456789 onder vermelding van het factuurnummer 202063

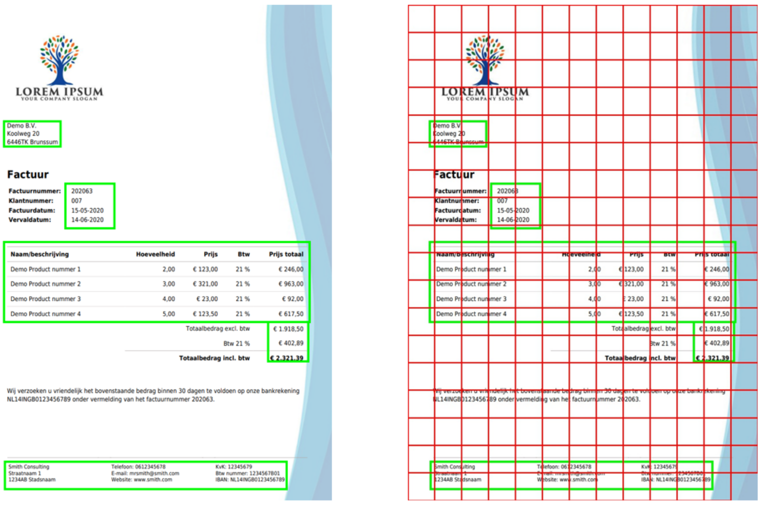
Smith Consulting
Straatnaam 1
1234AB Stadsnaam

Telefoon: 0612345678
E-mail: mrsmith@smith.com
Website: www.smith.com

KvK: 12345679
Btw nummer: 1234567B01
IBAN: NL14INGB0123456789

Figure 29: The left figure shows the annotated image, the annotated boxes are shown in green. The right figure shows the annotated image and a representation of the size of the pixels in the encoded feature space (the red lines).

Therefore, to get rid of this problem, it was decided to first locate some specific regions. These regions are explained in Table 6. The regions are found by using the earlier found fields and merging the fields into one big bounding box, an example can be seen in Figure 30. The figure also shows the pixel size of the feature space, which shows that the likelihood of having multiple objects in one pixel decreased significantly.



LOREM IPSUM
YOUR COMPANY SLOGAN

Demo B.V.
Koolweg 20
6446TK Brunssum

Factuur

Factuurnummer:	202063
Klantnummer:	007
Factuurdatum:	15-05-2020
Vervaldatum:	14-06-2020

Naam/beschrijving	Hoeveelheid	Prijs	Btw	Prijs totaal
Demo Product nummer 1	2,00	€ 123,00	21 %	€ 246,00
Demo Product nummer 2	3,00	€ 321,00	21 %	€ 963,00
Demo Product nummer 3	4,00	€ 23,00	21 %	€ 92,00
Demo Product nummer 4	5,00	€ 123,50	21 %	€ 617,50
			Totaalbedrag excl. btw	€ 1.918,50
			Btw 21 %	€ 402,89
			Totaalbedrag incl. btw	**€ 2.321.39**

Wij verzoeken u vriendelijk het bovenstaande bedrag binnen 30 dagen te voldoen op onze bankrekening NL14INGB0123456789 onder vermelding van het factuurnummer 202063.

Smith Consulting	Telefoon: 0612345678	KvK: 12345679
Straatnaam 1	E-mail: mrsmith@smith.com	Btw nummer: 1234567B01
1234AB Stadsnaam	Website: www.smith.com	IBAN: NL14INGB0123456789

Figure 30: The left figure shows the annotated image after labelling the regions. The right figure shows the annotated image and a representation of the size of the pixels in the encoded feature space. As with most of the invoices, this invoice has one box per class. From top to bottom: *Info_koper*, *Info_factuur*, *Artikel_tabel*, *Totaal_tabel*, *Info_verkoper*

Table 5: Description of the different types in this research.

Type	Description
empty	Many boxes that are generated by tesseract are empty and span over a large space on the image. These boxes are not important, and they are classified as empty.
number	Numbers could be prices, percentages, etc. As there are two types of decimal delimiters in numbers, namely, a “," and a “.", both have to be checked.
date	The dates are found through multiple regex statements.
percentage	Is assigned when a % symbol is in the entity.
BTW-code	As all Dutch BTW codes follow the same structure, and most invoices are from a Dutch seller, a regex statement is used to assign this type. The statement ensures that an entity that looks like “xxxxxxxBxx", where x can be any number between 0 and 9, is considered a BTW-code.
IBAN	Like the BTW-codes, an IBAN usually also follows the same structure. Hence, a regex statement ensures that entities that look like “yyxxyyyyxxxxxxxxxx", where y can be any letter and x can be any number, are considered as an IBAN.
quantity	This type is only assigned when a “#" is the first character in the entity.
currency symbol	Symbols like “€", “$", “£", etc.
e-mail address	Since e-mail addresses always follow the same structure, a regex statement is used to identify them. This statement ensures that entities that look like “(...)@(...).(...)", are considered as an e-mail address.
website	Like the e-mail addresses, websites also follow the same structure, namely, “www.(...).(...)". Hence, a regex statement is used to assign this type as well.
price	An entity gets this type when it has a currency symbol as its first or last character and the rest is a number, or when it has 2 decimal numbers.
name	An entity is considered a name when its lowercase representation is not equal to its current state, e.g., “Smith" $\neq$ “smith".
word	An entity is treated as a word when its characters are strictly letters.
words	Is assigned when an entity holds multiple words.
description	This type is only assigned after the description merge step. These newly merged fields are considered descriptions.
do not know	Is assigned when an entity does not adhere to the constraints of the other types.

Table 6: Description about the 5 different regions in this research.

Type	Description
Info_verkoper	This region contains information about the seller, the company who sent the invoice. It holds fields like bank account number, website, creditor name, BTW–code, etc.
Info_koper	This region contains information about the buyer, the company who receives the invoice. It holds fields like company name, address, etc.
Info_factuur	This region contains information about the invoice. It holds fields like invoice number, invoice date, etc.
Totaal_tabel	This region contains information about the total amount. It holds fields like total amount, tax percentage, tax amount, etc.
Artikel_tabel	This region contains information about the purchased articles. It holds fields like price, quantity, description, etc.

6 Methodology and Experimental setup

This section gives a detailed description of the methods and the experimental setup. The description of a method entails an explanation of the general structure and reasoning, while the experimental setup details the conducted experiments and hyper-parameter optimisation strategies, which lead to the final model choices.

In order to solve the problem, it had to be divided into multiple subproblems. After the preprocessing of the data, an Object Detection model has to find the correct fields. These are then compared to their respective fields in the database. The last step consists of adding the acquired information to the database. Before filling the database, the important classes had to be identified. This led to 13 classes that had to be logged into the database, and 17 classes that could be found additionally, but should not be added into the database. The 13 important classes are:

Inkoopfactuur_externereferentie	Inkoopfactuur_btwpercentage	Crediteurnaam
Inkoopfactuur_factuurdatum	Inkoopfactuur_btwbedrag	Crediteur_kvk
Inkoopfactuur_brutobedrag	Inkoopfactuur_totaalbedrag	BTW_code.
Inkoopfactuur_korting	Inkoopfactuur_valuta	
Inkoopfactuur_nettobedrag	IBAN-nummer	

As stated by Section 4, three models are implemented to solve the research problem, a Rule-Based NLP model, a non-Deep Learning, and a Deep Learning model. This is done to compare different techniques in finding a solution for which techniques can be used to check the reliability of the database counterparts of PDF invoices. Additionally, like [**?**], who integrated a LightGBM classifier into a CNN model, these methods could also be combined to form an even better model.

This section is divided into 4 parts. First, the Rule-based algorithm is explained (Section 6.1, then the decision tree model and its experimental setup will be explained (Section 6.2. Thereafter the RCNN (Deep Learning) model and its experimental setup will be described (Section 6.3). After that, the process of checking the database counterparts and creating the desired output are described (Section 6.4).

6.1 Rule-Based

The first approach to solving the research question is to use a classic knowledge acquisition technique. This technique is called Rule-based knowledge acquisition. Here the raw text is classified by predefined rules. Before classifying the fields, they must be found. The proposed regions are found by using `pytesseract` and the merge functions that are described in Section 5.3. Additionally, all the proposed text is cleaned and provided with a type; the types are described in Table 5.

The classification is done by first finding the aforementioned regions: buyer information, seller information, invoice information, article table and total table. The text within the regions is then classified as one of the many classes. The division of which fields are present in which region are shown in Table 7. The classification of the fields is described per region.

Table 7: The 5 regions and the information that can be found in those regions. As can be seen there are 5 regions: *Info_koper*, which holds information about the buyer, *Info_-Verkoper*, which holds information about the seller, *Info_Factuur*, which holds information about the invoice, *Artikel_tabel*, which holds information about the purchased goods, and *Totaal_tabel*, which holds the information about the total amount.

Info_Koper	***Info_Verkoper***	***Info_Factuur***	***Artikel_tabel***	***Totaal_tabel***
Name	Name	Number	Article codes	Gross amount
Address	Address	Date	# Articles	Net amount
	E-mail address		Prices	Total discount
	Website		Discount	Tax percentage
	Tax number		Currency	Tax amount
	IBAN		Contract period	Total amount
	Registration ID		Descriptions	Currency
			Total per article	
			Tax percentage	
			Tax amount	

Information about the invoice

The first region is *Info_factuur*. This region is found by assuming that it contains a date. There can be multiple dates on an invoice, although the first one is usually the invoice date. However, when companies use a datestamp, like in Figure 22a, the first date will be that one. Hence, when the first date is found in the upper corner of the invoice and there are more dates on the invoice, then the second date will be labelled as the invoice date. Otherwise, the first date will be labelled as the invoice date.

The next step is to find the neighbours of the invoice date (factuur datum). As the invoice number is usually next to, directly above or directly under the invoice date, it was decided that the region entails 2 lines above the invoice date and 3 lines beneath the invoice date. The invoice number is then found by finding each text field that has either type: word, name, number, or do not know, as they could all be an invoice number. The first one is then labelled as the invoice number (externe referentie).

Additional fields like client number and expiration date are usually also in this region. These could be found as they are normally the second entity with a number and the second date in the region, respectively.

Information about the buyer

The next region is *Info_koper*. This region is found by assuming that it always contains a postal code, which was added as an additional type in this stage. It is found by using a regex statement which ensures that entities that match “xxxx_yy (...)", where x is any number, y is any (capital) letter, and _ indicates that there could be a space, are marked as a postal code. After finding the postal code, the region is made, which consists of the line with the postal code and the two lines above and underneath it. If this region does not hold an e-mail address or a website, then it is seen as the *Info_koper* region. As this region normally only contains the name and the address of the buyer, the first box with the type “name” is labelled as “Koper naam” (name of the buyer). Additionally, the box with the postal code can be labelled as the buyer postal code.

Information about the seller
The third region is *Info_verkoper*. Like the previous region, it is assumed that this region also contains a postal code. However, unlike the previous region, this one should have at least an email address or a website in it. Furthermore, this region is also larger, as contains much more information. Hence, it was decided that this region consists of the line with the postal code, the 5 lines above and the 4 lines underneath it.

Next, the box with the types "IBAN" and "BTW code" are immediately labelled as *Inkoopfactuur_bankrekening* and *BTW code* respectively. While the KvK number is found by assuring that a field is a number with exactly nine numbers (as each KvK number has nine numbers). Furthermore, the creditor name is found by finding the first name in the region.

Additionally, the e-mail address box can be labelled as creditor email, the website can be labelled as creditor website, and the postal code can be labelled as creditor postal code. After the postal code is labelled, the line above can be used to find the address of the creditor.

Information about the articles
The next region is *Artikel_tabel*, as can be seen, none of the fields are deemed as important. Hence, this region does not have to be found or labelled. However, they could be labelled. This is done by first finding the article lines, which is described in Section 5.3. Like the labelling in the aforementioned section, the fields are labelled per line. Where the first number is labelled as the article amount, the first description, name, word, or words box is labelled as the article description, the first price is labelled as the price of the article, and the last price is labelled as the total amount that was spent on the article.

Information about the total amounts
The last region is *Totaal_tabel*, which mainly consists of prices and currency symbols. First, the boxes with the type "currency symbol" are labelled as *_valuta*. Next, all boxes that hold a price are found and ordered by position. The last price is labelled as *_totaalbedrag*, while the *_brutobedrag* and *_btwbedrag* are not immediately labelled. Before they are labelled, a check is done to ensure that the 2 prices above the total amount sum up to the total amount. If they do, then the largest price is labelled as *_brutobedrag* and the other price is labelled as *_btwbedrag*. After that, the tax percentage is calculated based on the two previously labelled fields. The field that holds the exact calculated tax percentage is then labelled as *_btwpercentage*.

6.2 LightGBM

The second model is tested to see whether a simple classification model could acquire similar results. The machine learning model that is used here is a decision tree called LightGBM, which is described in Section 3.2. This model classifies the specific fields instead of the important regions, as it does not classify based on a grid or an image itself, hence, it is much faster. The proposed regions are a combination of the bounding boxes extracted by pytesseract and their merged fields; the types of merged fields are explained in Section 5.3.

The model classifies the fields based on several features. These features are shown in

Table 8. The features were chosen based on the related work section (Section 4). They are specifically based on the features used in [Zhao et al., 2019, Xu et al., 2019, Tarawneh et al., 2019]. These researchers concluded that using text as well as locational features will lead to a more accurate model. The LightGBM model is implemented by using the Python library `lightgbm`, by [Ke et al., 2017].

Table 8: Features of LightGBM

Type	Description
The normalised bounding box coordinates	As some invoices are spread over multiple pages, the normalised bounding box coordinates would be more insightful. The normalised bounding box coordinates are also used in the LayoutLM model by [Xu et al., 2019], they concluded that scaling the bounding boxes between 0 and 1000 would lead to better generalisability.
The centre point of the bounding box	The researchers in \cite{cutie} used this feature to avoid the effects of having overlapping bounding boxes. Furthermore, it is also useful when objects in the same class vary in size but have a similar centre point.
The confidence of the reading	This confidence score is assigned by pytesseract and indicates the confidence of the reading. Hence, it can be insightful.
The type of the text in the box	This is assigned by the `get_type` method described in Section 5.3.
The length of the text in a box	The number of characters in the text.

As was mentioned before, the annotation is not complete. which leads to not all classes being represented on an invoice. Moreover, many words that are on an invoice are not considered as part of a class at all. This all leads to a significant class imbalance between the aforementioned classes and the "not important" class. According to [Oksuz et al., 2020], the positive samples are more important during training than the negative samples. Hence, it would not be beneficial to downsample them. Therefore, the positive classes are upsampled and the negative classes are downsampled to create a balanced dataset.

The resampling methods that are used are resampling with replacement, without replacement and SMOTE (Synthetic Minority Over-sampling Technique). Resampling with replacement is used to upsample a minority class, while resampling without replacement is used to downsample a majority class. Both of these methods randomly pick a sample from the data. However, the method with replacement allows picking the same sample multiple times, while the method without replacement does not. These methods are implemented by using the Python library `scikit-learn`, by [Pedregosa et al., 2011].

The other method, SMOTE, which was introduced by [Bowyer et al., 2011], is a well-known solution method for class imbalance. This method randomly oversamples the minority classes and downsamples the majority class. However, instead of oversampling the dataset by randomly adding one of the samples in the dataset, SMOTE randomly picks k samples (neighbours) and uses them to make one synthetic sample. A synthetic sample is a combination of the features of the feature spaces of its neighbours, e.g., the average of the coordinates, or the most frequent text type. SMOTE is implemented by using the Python library `imbalanced-learn`, by [Lemaître et al., 2017].

The experimental setup can be found in Table 9.

Table 9: Experiments of the model

Experiment	Input	Model	Comparison metrics
1	Pytesserect+merge	LightGBM	Accuracy and F1
2	Pytesserect+merge	LightGBM+upsampling positives	Accuracy and F1
3	Pytesserect+merge	LightGBM+upsampling positives + downsampling negatives	Accuracy and F1
4	Pytesserect+merge	LightGBM+SMOTE	Accuracy and F1

6.3 RCNN

The third and main model is a Region-based Convolutional Neural Network (RCNN), which will detect the objects. As was stated in the Background section, an RCNN model is divided into three modelling steps: the pre-trained backbone, the Region Proposal Network and the object classification and localisation stage. The model architecture can be seen in Figure 31. Traditional RCNN models use a pre-trained classification model as their backbone. However, these models require a fully supervised dataset, instead of a semi-supervised one. Hence, the pre-trained feature extractor had to be adjusted. Instead of using a pre-trained model from the Keras Python library, a new feature extractor was made, namely a convolutional autoencoder (CAE), as these are applicable in unsupervised problems.

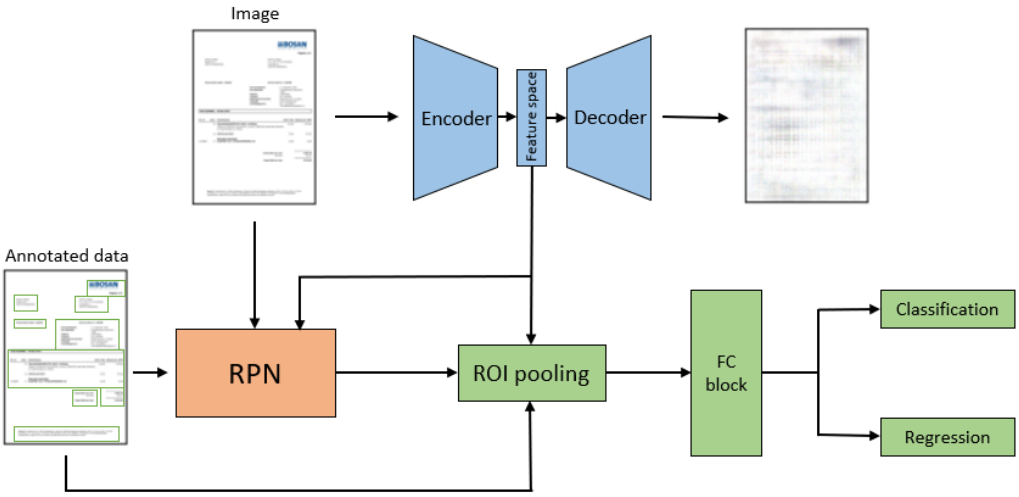


Figure 31: Architecture of the final model with its input. Stage 1 (blue): pre-training a CAE to learn the general structure of the invoices. Stage 2 (orange): Region proposal network to classify whether an area is foreground or background. Stage 3 (green): classify and localise the objects.

Unlike the backbone model, the RPN and Object Detection stages require at least some annotation. Therefore, the data had to be annotated by hand before training, which is described in Section 5.3.

The final output of the RCNN model will be the text and bounding box coordinates of the desired classes, which are: seller_information, buyer_information, invoice_information, total_table and article_table. Afterwards, these have to be further processed by a knowledge acquisition framework to find the specific fields.

It was decided to implement an RCNN model instead of a Faster-RCNN model due to a lack of time and resources. As the training of the three separate models took quite some time, there was not enough time and memory to run the three parts simultaneously.

As multiple researchers concluded that text features would make the localisation more accurate, some testing was done with an additional channel. This channel is added as the fourth channel of the input, alongside the RGB channels. The channel is a grid representation of the text types. For instance, '€' is located in grid point (300,260), so that space in the grid will hold the type: 'currency symbol'.

6.3.1 The backbone

As was mentioned before, the backbone of this model is a Convolutional autoencoder (CAE), which is described in Section 3.3.2. As this model learns to reconstruct an image, it holds a lot of information about the general structure of the input. Hence, the feature space of a trained autoencoder ensures better Object Detection results.

Many of the previously mentioned RCNN or Faster-RCNN models use large pre-trained models like ResNet50 by [He et al., 2015], or VGG19 by [Simonyan and Zisserman, 2015]. These models have 50 and 19 layers respectively and they are pre-trained on around a million images. As this research does not have as many images, it was decided to use fewer layers to reduce the chance of overfitting. Two models inspired the several architectures of the CAE in this research. The first model is the invoice classification model by [Kang et al., 2014]; its architecture is shown in Figure 36. The second model is AlexNet by [Krizhevsky et al., 2012], which is shown in Figure 10. These models have significantly fewer layers, but they do perform well on their classification tasks.

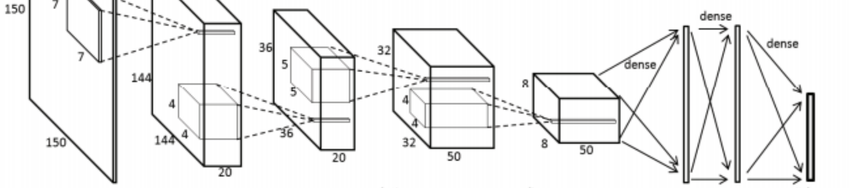


Figure 32: Structure of CNN for invoice classification based on their layout, taken from [Kang et al., 2014]

Since these models are classification models, they are used as the encoder part of the CAE. The decoders are the transposed versions of the encoders. Instead of convolutional layers, they have deconvolutional layers and instead of pooling layers they have upsampling layers. The layers are implemented using the Python libraries `Tensorflow`, by [Abadi et al., 2016], and Keras, by [Chollet et al., 2015]. Although an encoder separates the convolutional and pooling operations, a deconvolutional layer, and an upsampling layer can be combined into one operation: the transposed convolutional layer.

Furthermore, as atrous convolutional layers are shown to be beneficial for object detection tasks, there is also a model with atrous convolutional layers instead of normal convolutional layers. Moreover, most object detection and image classification models use square input images. However, invoices are normally the size of one sheet of paper. A sheet of paper has a size ratio of 1:1.41 (width:height). Hence, some of the architectures are also executed with that ratio.

The different architectures can be found in Appendix C and their results can be found in Section 7.3.1. As can be seen, there are many different architectures. However, as the size of the feature space directly influences the performance of the RPN model, the performance of the CAEs will not be the sole decision factor. Before deciding between atrous convolution or not, the feature spaces and their anchor options are compared. The final CAE is then chosen based on the overall performance of the model.

To get the best performance of the autoencoder, several optimiser strategies, loss functions, and learning rate combinations are executed to find the lowest reconstruction loss. The learning rate in the experiments ranges from 0.1 to 1e-6. Moreover, as reconstructing an image can be seen as a regression problem, the (multi) classification losses are not applicable. The experiments are shown in Table 10. Instead of trying many different optimiser strategies, it was decided to stick with Adam, as SGD was a lot slower and required a much lower learning rate before it produced reasonable results. Furthermore, Adam is one of the more popular methods in Object Detection tasks, where most of the state-of-the-art Object Detection researchers use Adam, some of them include: [Girshick et al., 2013, Ren et al., 2015, Zhao et al., 2019, Kang et al., 2014, Xu et al., 2019, Devlin et al., 2018]. They prefer using Adam as it takes advantage of feature sharing during training, which decreases the number of computations per batch, and hence the amount of memory needed to train the model.

Table 10: Experiments of CAE

Experiment	**Input size**	**optimiser**	**Loss function**	**Atrous**	**Decoder layer**
1	(150x150x3)	Adam	MSE	No	Upsample+Deconv
2	(150x150x3)	Adam	MAE	No	Upsample+Deconv
3	(150x150x3)	Adam	KLD	No	Upsample+Deconv
4	(150x150x3)	SGD	MSE	No	Upsample+Deconv
5	(150x150x3)	Adam	MSE	No	Upsample+Deconv
6	(227x227x3)	Adam	MSE	No	Upsample+Deconv
7	(227x227x3)	Adam	MSE	No	TransposeConv
8	(320x227x3)	Adam	MSE	No	TransposeConv
9	(454x454x3)	Adam	MSE	No	TransposeConv
10	(640x454x3)	Adam	MSE	No	TransposeConv
11	(640x454x3)	Adam	MSE	Yes	TransposeConv

According to many researchers, as described in Section 4, using textual features alongside image features leads to better performance, than solely letting a model train on the image features. Hence, it was decided to add a type map to the input. This type map is inspired by [Zhao et al., 2019], which made a grid representation of the embeddings of the text. However, instead of using text embeddings, the type of the text is put into the grid. Each type is represented by a number, where similar types have distant numbers, as the model could give meaning to the preassigned number instead of focusing on what the number could mean. The new input, which now consists of 4 channels, is given to the same RCNN model, however, now the input and the output size of the CAE becomes $(640 \times 454 \times 4)$.

6.3.2 RPN

A Region Proposal Network (RPN) takes the encoded feature space of an image and tries to classify whether a box could be foreground or background. As was mentioned in Section 3.3.3 an RPN model consists of one convolutional layer and two output layers. As [Girshick et al., 2013, Ren et al., 2015] suggest using a (3×3) kernel and 512 filters, these are used in this RPN model as well. The output size is determined by the number of anchors (a). Here the number of anchors is equal to 9, as $3 \times 3 = 9$.

The first output layer is the classification output layer. This layer returns a 0 (background) or a 1 (foreground). Hence, the output should be determined by the binary-cross entropy, as it determines one class. Furthermore, the activation function should also be tailored to the classification problem, which is the SoftMax function. The second output layer specifies the coordinates of the boxes; hence the output size of this layer is equal to $4 \times a$. Moreover, as this output layer predicts the coordinates, this layer should be optimised by a regression problem loss function and its activation function should be linear. The architecture of the RPN model is shown in Table 11. The model is implemented by using the Python libraries `Tensorflow` and `Keras`. As the feature space shape is not fixed yet, it is notated as hf (height of feature space), wf (width of feature space) and ff (number of filters of the feature space).

Table 11: RPN model

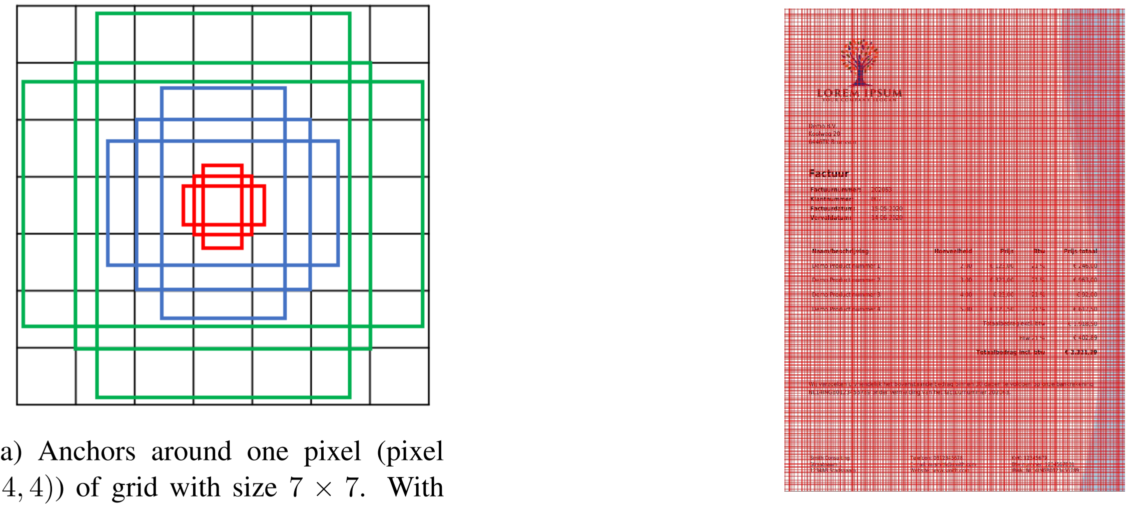
Model: "Region Proposal Network"

Layer (type)	Name	Output Shape	Param #	k	f	s	p	h
InputLayer	Input	(β, hf, wf, ff)	0					
Conv2D	conv_1	(β, hf, wf, 512)	1180160	512	(3,3)	(1,1)	*same*	
Conv2D	scores1	(β, hf, wf, 30)	4617	36	(1,1)	(1,1)	*valid*	*sigmoid*
Conv2D	deltas1	(β, hf, wf, 120)	18468	9	(1,1)	(1,1)	*valid*	*linear*

Total params: 1,203,245
Trainable params: 1,203,245
Non-trainable params: 0

The anchors are formed based on two predefined parameters: scale and ratio. As the convolutional filter size is 3×3 there are three scales and three ratios. These parameters define the size of the anchors. A representation of the anchors around one pixel is shown in Figure 33a. Here the colours indicate the different ratios, while the size within the ratio is determined by the scale parameter, giving each pixel in a feature space 9 anchors. This leads to many region proposals. The ratios in this figure are $(1 : 1)$ (a square), $({}^{1}/{\sqrt{2}} : {}^{2}/{\sqrt{2}})$ (a horizontal rectangle), $({}^{2}/{\sqrt{2}} : {}^{1}/{\sqrt{2}})$ (a vertical rectangle), which are the recommended ratios by [Ren et al., 2015].

The feature space output of AlexNet would lead to $6 \times 6 \times 9 = 324$ possible regions, as its feature space has size $(6 \times 6 \times 256)$. However, the anchors that exceed the borders of the image are normally discarded. A representation of all the generated anchors on an image is depicted in Figure 33b.



(a) Anchors around one pixel (pixel $(4, 4)$) of grid with size 7×7. With ratio 0.5 (red) 1 (blue) and 2 (green).

(b) All anchors on an image

Figure 33: Example of anchors

Since the actual regions that hold an object, which are given in the ground truth, are usually not perfectly aligned with the anchors, a ground truth cannot be provided as the preferred outcome of the RPN model. Instead, the predefined anchors are labelled as foreground or background. The labelling is based on the criteria called Intersection over Union (IoU). The IoU measures the amount of overlap between two regions, here the regions are a proposed region and a region in the ground truth. As was stated by [Ren et al., 2015], when the IoU value is greater than 0.7, a proposed region can be labelled as foreground, and when the IoU is below 0.3, it can be labelled as background. The other regions are not used, as they are indifferent. The labels and the coordinates are then used as the targets of the model.

Additionally, 11 proposed to discard regions whose overlapping with other proposed regions are too high. This technique is called Non-maximum Suppression. For instance, regions that have an IoU of 0.9 or higher are too similar, hence only one of them is kept.

The goal of the RPN model is to identify the correct regions as foreground and to give their coordinates. This output will then be given to the next part of the RCNN model: the classifier. The experimental setup of finding the preferred feature space is shown in Table 12.

6.3.3 Classification

The last part of the RCNN model is the classification model. This model consists of a ROI pooling layer, a few fully connected layers, and two output layers. As was stated in Section 3.3.3, the ROI pooling layer ensures that all input regions have the same feature space size when they enter the fully connected block. The input of the ROI layer entails

Table 12: Anchor experiments

Experiment	Feature space	Anchor ratios	Anchor sizes
1	(6x6)	$(1:1, {}^{1}/\sqrt{2} : {}^{2}/\sqrt{2}, {}^{2}/\sqrt{2} : {}^{1}/\sqrt{2})$	1-500
2	(8x6)	$(1:1, {}^{1}/\sqrt{2} : {}^{2}/\sqrt{2}, {}^{2}/\sqrt{2} : {}^{1}/\sqrt{2})$	1-500
3	(13x13)	$(1:1, {}^{1}/\sqrt{2} : {}^{2}/\sqrt{2}, {}^{2}/\sqrt{2} : {}^{1}/\sqrt{2})$	1-500
4	(18x13)	$(1:1, {}^{1}/\sqrt{2} : {}^{2}/\sqrt{2}, {}^{2}/\sqrt{2} : {}^{1}/\sqrt{2})$	1-500
5	(27x27)	$(1:1, {}^{1}/\sqrt{2} : {}^{2}/\sqrt{2}, {}^{2}/\sqrt{2} : {}^{1}/\sqrt{2})$	1-500
6	(38x27)	$(1:1, {}^{1}/\sqrt{2} : {}^{2}/\sqrt{2}, {}^{2}/\sqrt{2} : {}^{1}/\sqrt{2})$	1-500
7	(38x27)	$(1:1, {}^{1}/\sqrt{2} : {}^{2}/\sqrt{2}, {}^{2}/\sqrt{2} : 0.1)$	1-500
8	(38x27)	$(1:1, {}^{1}/\sqrt{2} : {}^{2}/\sqrt{2}, {}^{2}/\sqrt{2} : 0.15)$	1-500
9	(38x27)	$(1:1, {}^{1}/\sqrt{2} : {}^{2}/\sqrt{2}, {}^{2}/\sqrt{2} : 0.25)$	1-500

the regions that are proposed by the RPN model and the encoded feature space. The regions are then pooled into the same feature size, which is normally an $f \times f$ square. [Ren et al., 2015] proposed using a 7×7 kernel for object detection. As the size of the objects in this research and in the research of Ren et al. do not differ significantly in size, a pool size of 7×7 is also used in this part of the research. Furthermore, the ROI pooling layer also pools regions to 7×7 if they are smaller than the pooling region. Hence, the smaller objects can still be pooled.

The pooled regions then form the input batch of the next block. The batch size is normally hardcoded, where most (Faster-)RCNN models use a fixed number of proposals and a fixed batch size. The number of proposals is usually around 2000 proposals per image, while batch sizes range from 1 to 2000. [Ren et al., 2015] uses a batch size of 4, while others do not use multiple batches per image. After the ROI pooling, the fully connected block linearises the batch features into one dimension. Normally, the fully connected layers in the block are the same fully connected layers that are not used in the backbone. For instance, the feature space of AlexNet is created after the last convolutional layer (layer 5, with 9216 units), then it has 2 fully connected layers (both have 4096 units), which are followed by the output layer. As AlexNet is the inspiration of the CAE in this research, its fully connected layers will be used in the fully connected block. The ratio between the number of units in the feature space and units in the fully connected layers is 2.25. Hence, this ratio is also used in this model.

6.3.4 Hyper-parameter optimisation

The previously mentioned models all have hyper-parameters which influence the performance of the models. Hyper-parameters are fixed parameters that are established before running the model. As these parameters do not change during training, they should be chosen wisely. A common way of finding “the best” hyper-parameters is grid search. This method tries a lot of different hyper-parameter combinations and runs the model with all combinations. All combinations are tested on the same dataset with the same number of epochs. After testing all combinations, the losses are compared and the hyper-parameter combination with the lowest test loss is chosen to train on. In some cases, when the losses are similar, the fastest model can be chosen as the “best”.

The non-Deep learning model in this research only has one hyper-parameter, namely the

Table 13: RCNN model

Model: "RCNN:classifier"					
Layer (type)	Name	Output Shape	Param #	f	$h(\cdot)$
InputLayer	input_11	(None, hf, wf, ff)	0		
InputLayer	input_12	[(None, 4)]	0		
InputLayer	input_13	[(None, 1)]	0		
RoIPooling	roi_pooling	(None, 7, 7, ff)	0	(7,7)	
Flatten	flatten	(None, 12544)	0		
Dense	fc1	(None, 4096)	51384320		*relu*
Dense	fc2	(None, 4096)	16781312		*relu*
BatchNormalization	batch_normalization	(None, 4096)	16384		
Dense	scores2	(None, 5)	24582		*softmax*
Dense	deltas2	(None, 20)	98328		*linear*

Total params: 1,257,110
Trainable params: 1,257,110
Non-trainable params: 0

maximal number of iterations it is allowed to do to find the best classification strategy. This number is usually based on the number of classes of the model. For instance, a normal number is $100 \times C$. When the maximal number of iterations increases, the accuracy of the model normally also increases. However, too many iterations can lead to overfitting, which results in a poor performance on the test set.

Unlike the non-Deep learning model, the Deep learning model has more hyper-parameters. The hyper-parameters of the Neural Networks are the epochs, the number of layers, the type of layers, the kernel sizes, the strides, the number of filters, the activation functions, and the learning rate. Furthermore, the optimiser strategy and loss function also determine the performance of the model.

The RPN model has more hyper-parameters, namely, the anchor sizes and ratios. The ratios are inspired by the standard ratios of [Ren et al., 2015], they concluded that using a square, a horizontal and a vertical rectangle would be best. The sizes, however, are not inspired by the paper, as they are found by trial and error. Hence, this is done by checking each possible anchor size and picking the three sizes with the highest IoU scores. Additionally, the chosen sizes should also ensure that all classes have at least one region with an IoU of at least 0.7. This ensures that the model will see it as foreground.

The different architectures are tested with different loss functions and learning rates. The architecture and hyper-parameter combination with the lowest loss will be the final architecture. This model will then be trained longer to find an even better performance.

The performance of a model is determined by its loss, which is composed of the bias and the variance of the model. Bias is caused by inaccurate assumptions of the model, which assumptions arise during the training stage. Some examples of bias in this research could be that each date will be classified as the invoice date, while this is not true.

6.3.5 Knowledge acquisition

The next step is to extract the text from the regions and to find the specific fields. This can be done in many ways; some include using 5 separate RCNN models that specialise in localising the fields within a specific region. Others use a classification model like LightGBM or Rule-Based models. As each region holds specific fields, it was decided to use the better performing model between the Rule-Based and the LightGBM models as the last classification step.

The Rule-Based algorithm is like the Rule-Based algorithm in Section 6.1. However, unlike in the previous Rule-Based algorithm, the regions are already located.

6.4 Putting it in the Database

The output of the models will be a table that holds the coordinates, the label, and the text of a field. This output will then be compared to the data in the SoliTrust Database, for each (key, value) pair (output model, database entry). If the value and the key are equal, then the box will be green and a probability of equality of 100% is entered into the database. If they are not, then the box will be red, and a probability of equality must be calculated. This score counts the characters that are similar and divides them by the length of the key. The probability score is added to the database as well. The additional fields are not checked with the database, but they are added to the image in blue. Moreover, to check whether the classification was wrong due to labelling the wrong box or because the value in the database was not filled in correctly, another check is done. This check goes through all text on the invoice and checks whether the key is present. If it is present, then a red box will be placed around it. This check is also added to the database as a 1 (present) or a zero (not present).

7 Results

This section describes the results of the models and experiments that are described in Section 6. First, the results of the Rule-Based model are described. Next, the results of the LightGBM model are described. Third, the results of the RCNN model are described and the fourth subsection covers the overall results after matching with the outcomes with the database. All results are based on training the train set and testing the test set. As not all test images could be fully annotated, the results of the box plots are generated by invoices in the test set that at least some regions pre-assigned.

7.1 Rule-Based

The results of the Rule-Based algorithm are depicted in Figure 34. The figure shows a very high average accuracy, which is around 96%, accompanied by a much lower average F1 score, which is around 42%. The results are based on the 30 predefined classes in Table 2 and a background class. As not all classes are as important, a weighted F1 score is calculated as well. This score gives a weight of 1 to all classes that are deemed as important and a weight of 0 to the rest of the classes. The results show that the adjusted F1 score is around 42% as well. These F1 scores are quite low, however, as many of the important fields tend to not be logged into the database, those scores will always be zero, which means that even if they were located correctly, they will be seen as a fail. Interesting to note is that some outliers have a score of 1, while others have a score of 0. The exact minimum, maximum and average scores can be found in Table 31 in Appendix F.

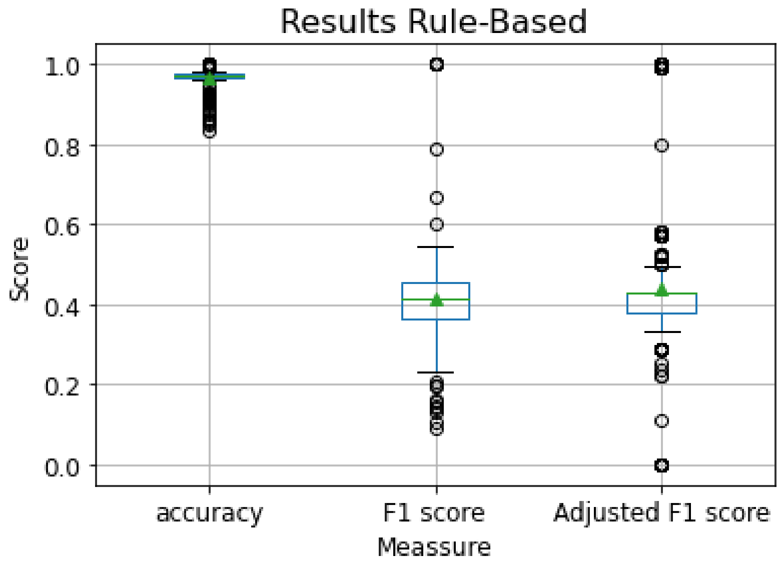


Figure 34: Results of Rule-Based algorithm. The median is given by the green line, while the mean is given by the green triangle.

7.2 LightGBM

The python package LightGBM is used for the implementation of the model. It has one input parameter that can be altered to determine the 'best' parameter. This parameter determines the maximum depth of the tree and thus it can be used to decline overfitting, as a tree that is too deep will start to classify based on the details, instead of the generalisable features. [Ke et al., 2017, Khoshrou and Pauwels, 2019]. It is optimised within the module itself, so no parameters had to be assigned beforehand. The results are shown in Table 14. The table shows many resampling techniques, which are only applied to the train set, not the evaluation set. The resampling was not applied to the evaluation set, as experiments should be comparable, which entails that they should be evaluated by the same set.

Table 14: Results of the experiments on LightGBM.

Model	**Accuracy**	**F1**
LightGBM	0.987	0.044
LightGBM+upsampling positives (5500)	0.988	0.040
LightGBM + upsampling positives (2000) + downsampling negatives (5500)	0.990	0.074
LightGBM + upsampling positives (2000) + downsampling negatives (10000)	**0.992**	0.176
LightGBM+upsampling positives (2000) downsampling negatives (20000)	0.950	0.063
LightGBM+SMOTE (downsample majority to 10000)	0.170	0.041
LightGBM+SMOTE (downsample majority to 20000)	0.280	0.057
LightGBM+SMOTE (downsample majority to 50000)	0.500	**0.112**
LightGBM+SMOTE (downsample majority to 80000)	0.001	0.004

The first experiment shows a high accuracy but a low F1 score. This indicates that most classes are not found. As was shown in Section **??**, the data is severely imbalanced, which explains the high accuracy, as all samples are probably classified as class 30. Therefore, it was decided to upsample the minority classes. However, as some classes barely have any samples, oversampling them leads to having the same few samples many times. This results in bad generalisability, and it could also lead to overfitting. First, it was decided to even out the positive classes and class 30. As there are 31 classes in total, each class should be upsampled to $^{167308}/_{30} \approx 5500$. As can be seen in Table 14, the upsampling led to a higher F1 score, which indicates less overfitting. However, it is still very overfit. Hence, it was decided to downsample the majority class to 5500 samples as well, to have no class imbalance. This led to similar results.

To further test whether a larger majority set leads to more overfitting, the model was also tested with a majority sample which was downsampled to 10000, and where the minority samples were downsampled to 2000. This led to a better F1 score, but not a significant difference. Hence, another test was done where the majority class was downsampled to 20000 and the minority classes were upsampled to 2000. However, this did not lead to a better F1 score either.

The next experiment includes using SMOTE to upsample the minority classes to the same amount of majority samples with the SMOTE algorithm. The first experiment, which was to upsample all classes to 167308 samples failed, due to a memory error. Hence it was decided to downsample the majority class as well. All four SMOTE experiments led to a worse accuracy score, which indicates that unimportant text boxes are now deemed important.

The results of the best performing LightGBM model can be seen in Figure 35. This figure shows an average accuracy score of 97% and an average F1 score of 29%. As can be seen, the adjusted F1 score is around 18%. This is even lower than the scores of the Rule-Based approach. This indicates that the LightGBM classifier tends to find unimportant classes while mislabelling the important ones. The exact minimum, maximum and average scores can be found in Table 31 in Appendix F.

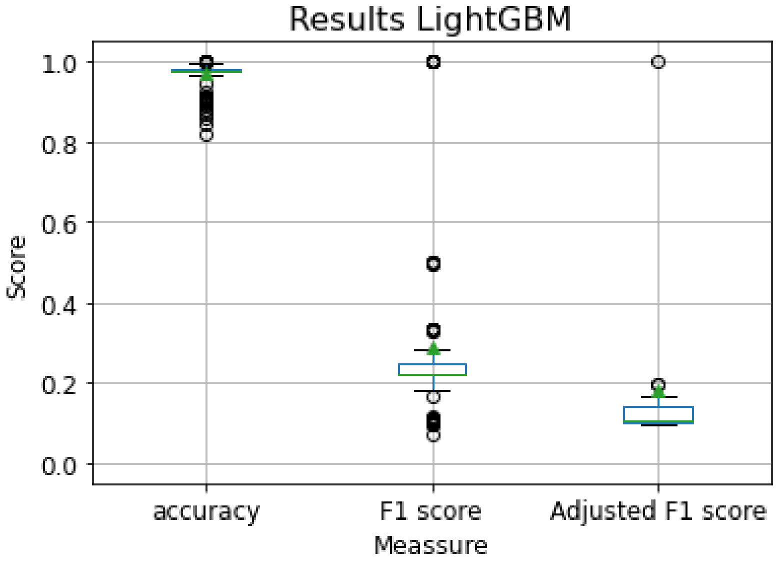


Figure 35: Test results LightGBM

7.3 RCNN

The results of the RCNN model are divided into three parts, which correspond to the three modelling stages. First, the different CAE architectures and their results are described, which consists of the final loss and the reconstructed images. Next, the anchor and RPN results are described, and the last subsection will describe the classification results.

Furthermore, as the input was too big to work with batches, all modelling stages were implemented with a batch size of 1. Also, the learning rate that was used during these experiments was 1e-6. This value was chosen as lower learning rates did not result in converging models.

The training results were generated by the Lisa server by SURFsara on a 256GB patrician of a shared GPU-node. The maximum memory per node was 256GB and the maximum run time of a job was 120 hours. A summary of the type of GPU-node that is used is shown in Table 15.

The computational time of the different RCNN stages depends on the input size and the number of computations per epoch. A summary of the estimated run times is shown in Table 16. Many trial and error runs were done to find the final models. Some were further trained after the first 120 hours, others were not.

Table 15: Summary of used GPU-node

Number	23	
Processor Type	bronze_3104	
Clock	1.70 GHz	
Scratch	1.5 TB NVME	
Memory	256 GB UPI 10.4 GT/s	
Sockets	2	
Cache	8.25 MB	
Cores	12	
GPUs	4 x GeForce 1080Ti,	11GB GDDR5X
Interconnect	40 Gbit/s ethernet	
SBU/node-hr		* 42.1

Table 16: Summary of the estimated run times per model.

	Runtime per 100 epochs	**Epochs after 120 hours**
CAE A1	8 hours	
CAE A2	12 hours	
CAE A3	15 hours	
CAE A4	40 hours	300
CAE A5	40 hours	300
RPN A4	70 hours	170
RPN A5	70 hours	170

7.3.1 Different CAE architectures

Many different CAE architectures and hyperparameter options are tested. As there are around 4000 invoices, the RCNN structure should not have too many layers. However, small models usually coincide with small input images, which leads to small feature spaces as well. As was mentioned before, due to the small objects, a small feature space is not preferred.

The first architecture (architecture 1) that is tested is based on the invoice classification CNN made by [Kang et al., 2014], which can be seen in Figure 36. The autoencoder used the convolutional and Maxpooling layers, which were followed by transpose convolutional and upsampling layers instead of the fully connected layers. The full autoencoder architecture can be seen in Figure 36.

This architecture was not used due to fact that the pixelmaps were too small. Consequently, too much information was lost. Furthermore, it also led to overlapping regions, which made it difficult for the model to learn which pixel belonged to which class. However, it was used to test different optimisers and loss functions, as this model took much less time to train.

The second architecture (architecture 3) that was tested is based on AlexNet from [Krizhevsky et al., 2012]. However, due to the square input and the small feature space, many regions overlapped. Hence, it was decided to use this architecture, but with a double input size and an A4 ratio. These architectures are also tested with atrous layers (architecture 4) and without atrous layers (architecture 5). Architecture 4 is shown in Figure 37. All full

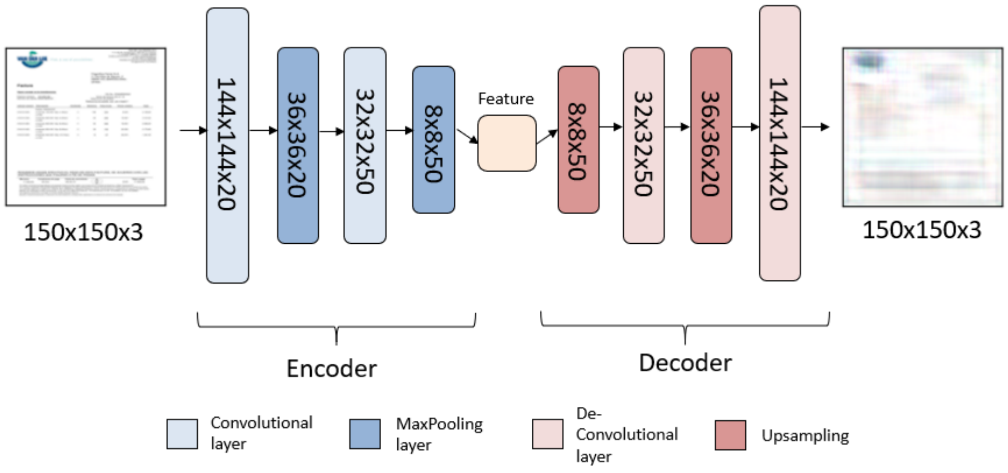


Figure 36: Convolutional autoencoder inspired by [Kang et al., 2014]

summaries of the 4 different architectures can be found in the Appendix, in Tables 24, 25, 26, , 27, and 28 respectively.

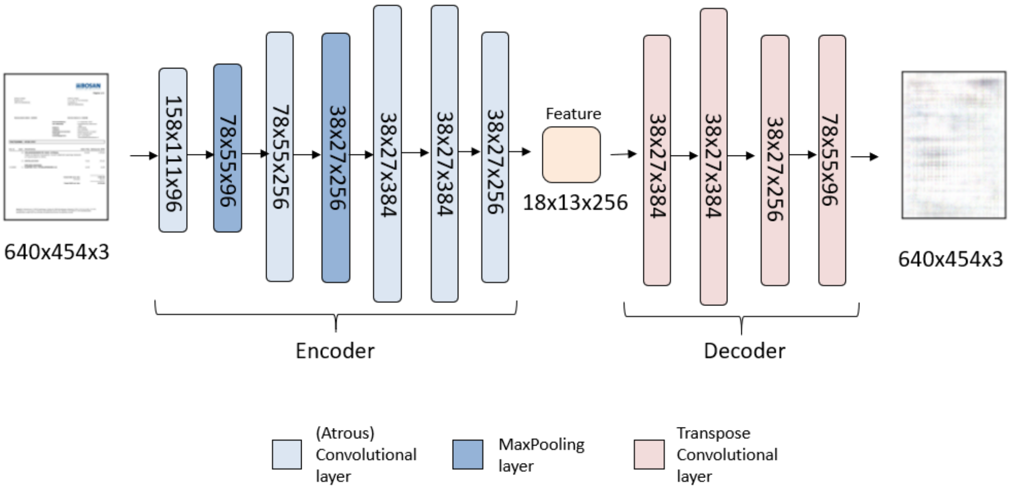


Figure 37: Convolutional autoencoder inspired by [Krizhevsky et al., 2012]. It also shows an example input and its output (the reconstructed image).

Before choosing the architecture, which is ultimately chosen by the grid comparisons in Section 7.3.2, the architectures are tested with different optimisers, loss functions, and decoder layers. The different decoder layers are convolution+upsampling, transposed convolution + upsampling, and transposed convolution alone. As all loss functions rapidly went to 0 it was decided to compare the experiments by their reconstructed images. The reconstruction results of the eight different experiments are shown in Figure 38.

The figure shows that using the loss function MSE does lead to sufficient results, whereas MAE does not. The MSE results do not significantly differ between using SGD and Adam. However, according to many researchers, who are mentioned in Section 3.3, Adam is used more often than SGD. Hence, these experiments, and previous literature conclu-

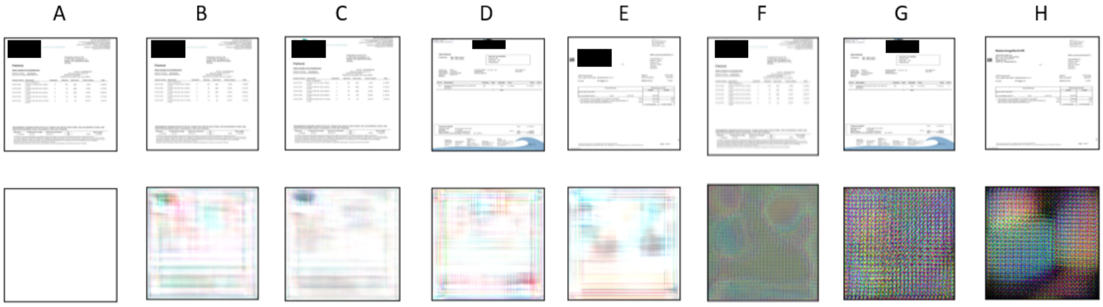


Figure 38: Different CAE results. A (2 different experiments): model 1 and 2 with loss MAE and optimiser Adam. B: architecture 1 MSE Adam 100 epochs. C: architecture 1 MSE Adam 200. D: architecture 1 MSE SGD 200. E: architecture 2 MSE Adam 200 (only transpose). F: architecture 2 (transpose+deconv) MSE Adam 50. G: architecture 2 (transpose+deconv) MSE Adam 200. H. architecture 2 (transpose+deconv) MSE Adam 200. All losses are very close 0, most of them are in the range of 0.500 to 0.000.

sions, led to the decision of using the optimisation strategy Adam and the loss function MSE. Furthermore, it can also be seen that using the transpose+deconv option did not lead to good results, hence it was decided to just use transposed convolution.

To further analyse whether atrous convolutional layers can be beneficial, which could ultimately only be decided after the classification stage, a comparison of the loss functions of architecture 4 and 5 is done. Here, both architectures are trained for 100 epochs. The comparison is shown in Figure 39.

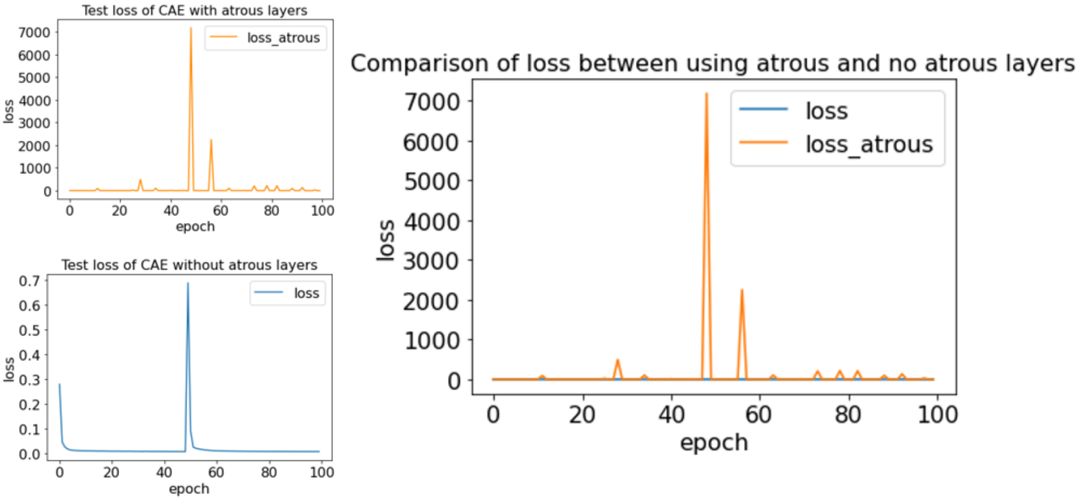


Figure 39: Loss realisations of architecture 4 (loss) and 5 (loss_atrous), which are also shown on the left.

As can be seen in the figure, the loss values are in different ranges, but they are both very close to zero. The lowest loss of the model without atrous layers is 0.0079 and the lowest loss of the model with atrous layers is 0.0085, which are very close. Hence, the model without atrous layers was tested for 5000 instead of 100 epochs, to see whether there was a chance of an even lower loss. The loss distribution is shown in Figure 40. The lowest loss that was recorded in the 5000 epochs was 0.0077, which means that the model improved by 0.0002. The figure also shows that the loss does not significantly decrease after 100 epochs. An example of the reconstructed image of this model is shown in Figure 37.

In short, five architectures were tested, where three of them had too small feature spaces to detect the objects. Hence, architectures 4 and 5, which are based on AlexNet, are trained further. The difference between the models is the type of convolutional layers, where architecture 4 uses normal convolutional layers and architecture 5 uses atrous convolutional layers. As neither of them performed significantly better at this stage, no conclusion can be made about which layer type would be better.

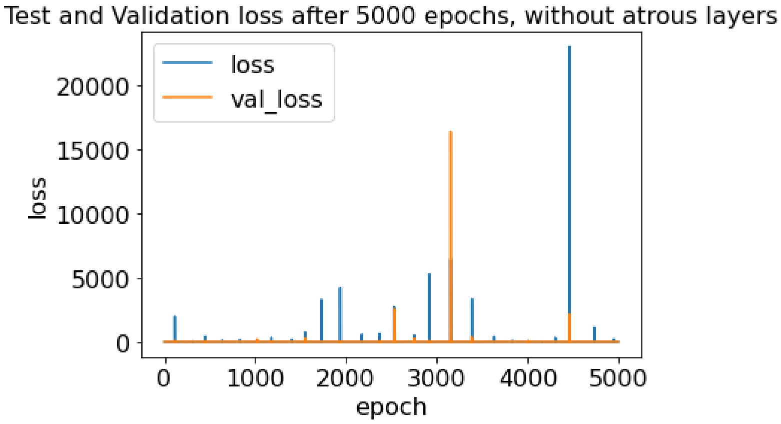


Figure 40: Loss distribution of architecture 4 with 5000 epochs

7.3.2 RPN

Anchors and grids

The next step is to determine the anchors, which in turn also determines the preferred feature space and thus the architecture. In order to find the best regions, the anchors must be similar to the ground truth boxes. As all boxes are of different frames and sizes, it can be difficult to find appropriate anchors. However, by having the right sizes and ratios, most of the ground truth will have at least some reasonable options. The anchors depend on the size, ratio, and shape of the feature space. If the size of the feature space is small, then the anchors are large, which leads to a less accurate representation of the ground truth, where some will not be located at all. Moreover, a smaller feature space also leads to fewer anchors.

As the goal of an RPN model is to find the potential locations of the objects, its predefined anchors must have some overlapping with the ground truth, i.e., at least 70%. To find the desired feature space, three feature space sizes are tested; a representation of the grids is shown in Figure 41. The figure shows the feature space size of architecture 3 and two different feature spaces of architecture 4 and 5. They are compared with the IoU scores of their anchors. The ratios of the testing procedure are the standard ratios of [Ren et al., 2015]: $(1 : 1)$ (a square), $({}^{1}/\sqrt{2} : {}^{2}/\sqrt{2})$ (a horizontal rectangle), $({}^{2}/\sqrt{2} : {}^{1}/\sqrt{2})$ (a vertical rectangle), while the size of the anchors ranges from 1 to 500 pixels on the input image. The results are shown in Table 17. As can be seen in the table, the testing is done

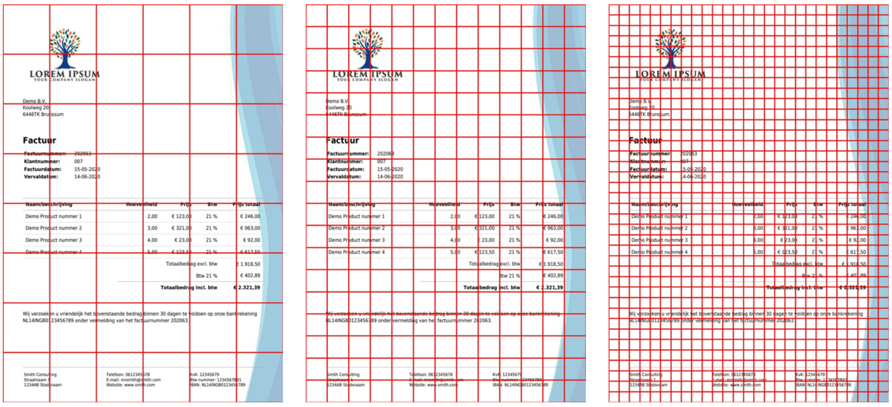


Figure 41: Representation of the feature space grids. The left figure shows the feature space of architecture 3 (8×6). The middle figure shows the feature space of architectures 4 and 5 (18×13). The right figure shows the feature space of architecture 4 and 5, but without the last pooling layer (38×27).

with 4 types of ground truth boxes. These boxes are a small square, a thick but smaller horizontal box, a wide and thin horizontal box, and a vertical box. The boxes are similar to the boxes in Figure 30.

Table 17: Results of different the IoU values of the anchors and the ground truth in different feature spaces

RPN	type	IoU<0.3	IoU>0.7	max IoU	Anchor size
6x6	horizontal_thick	1-52 151-500		0.50	115
	small_square	1-500		0.16	97
	vertical	1-70 166-500		0.41	129
	horizontal_thin	1-500		0.23	169
8x6	horizontal_thick	1-46, 114-500		0.69	78
	small_square	1-21, 72-500		0.8	48
	vertical	1-74, 166-500		0.4	129
	horizontal_thin	1-500		0.26	131
18x13	horizontal_thick	1-52, 156-500		0.58	99
	small_square	1-52, 156-500		0.58	99
	vertical	1-49, 168-500	77-104	0.81	84
	horizontal_thin	1-500		0.26	96
27x27	horizontal_thick	1-46 155-500	75-100	0.80	84
	small_square	1-21 72-500		0.59	45
	vertical	1-49 166-500	85-108	0.78	99
	horizontal_thin	1-500		0.26	112
38x27	horizontal_thick	1-45, 156-500	76-100	0.8	84
	small_square	1-45, 1-21 72-500	34-46	0.8	43
	vertical	1-49, 168-500	82-108	0.82	95
	horizontal_thin	1-500		0.26	112

The results show that the smaller feature spaces lead to inadequate anchors. As a result, all anchors are seen as background, which leads to an RPN model that cannot find any proposals. Hence, it was decided to use a larger feature space. However, instead of training a new CAE with an even bigger input size, it was decided to drop the last pooling layer of architectures 4 and 5. This decision was made because training a model with an even larger input size would lead to too long training times and memory errors, which were caused by the large increase in computations after the input size was enlarged. The new feature space has a dimension of $(38 \times 27 \times 256)$. This feature space did lead to appropriate anchors. However, not for all.

Hence, it was decided to test different ratios as well. As the thin and wide box, which is usually an article table, did not have a sufficient IoU score, it was decided to first change the ratio of the horizontal box. Three options were tested, all are a thinner version of the original one. As some article tables are thicker, they should not be too thin either. The results are shown in Table 18.

Table 18: Results of different anchor ratios.

RPN	**type**	**IoU<0.3**	**IoU>0.7**	**max IoU**	**Anchor size**
38x27 ($^1/\sqrt{2}$: $^2/\sqrt{2}$)	article_thin	1-500		0.26	112
	article_normal	1-129 431-500	198-281	0.78	235
	article_thick	1-500		0.26	112
38x27 $^2/\sqrt{2}$:0.1	article_thin	1-164 442-500	252-359	0.84	299
	article_normal	1-129 431-500	198-281	0.78	235
	article_thick	1-164, 442-500	252-359	0.84	299
38x27 $^2/\sqrt{2}$:0.15	article_thin	1-134 442-500	210-293	0.81	245
	article_normal	1-129 442-500	198-281	0.78	235
	article_thick	1-134 442-500	210-293	0.81	245
38x27 $^2/\sqrt{2}$:0.25	article_thin	1-104 348-500	210-293	0.81	189
	article_normal	1-129 442-500	198-281	0.78	235
	article_thick	1-104 348-500	210-293	0.81	189

As can be seen in the table, all types of boxes are accounted for in a feature space of (38×27) pixels and the ratios: $(1 : 1)$ (a square), $(^2/\sqrt{2} : 0.25)$ (a thin horizontal rectangle), and $(^2/\sqrt{2} : ^1/\sqrt{2})$ (a vertical rectangle). Hence, either architecture 4 or architecture 5 can be chosen as the final CAE model.

Region proposals

The next step is to classify which anchor regions are foreground and which are background. This is done by the RPN model, with the input of either CAE architecture 4 or 5. Hence, the final RPN architecture is given by Table 19, which can have two different input CAE feature spaces.

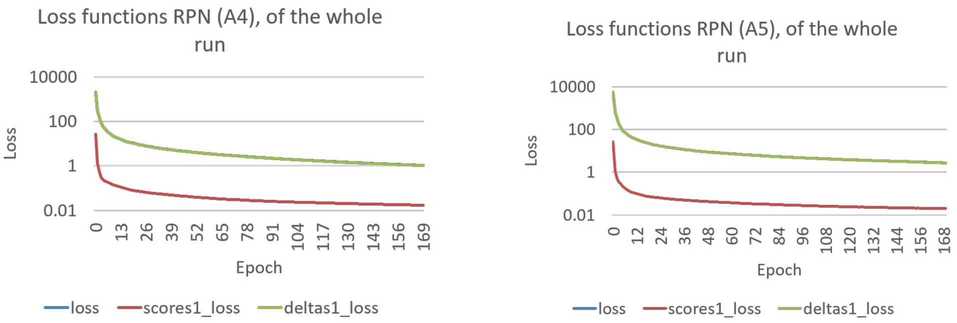
Table 19: RPN model

Model: "Region Proposal Network"

Layer (type)	Name	Output Shape	Param #	k	f	s	p	h
InputLayer	Input	(β, 38, 27, 256)	0					
Conv2D	conv_1	(β, 38, 27, 512)	1180160	512	(3,3)	(1,1)	*same*	
Conv2D	scores1	(β, 38, 27, 30)	4617	36	(1,1)	(1,1)	*valid*	*sigmoid*
Conv2D	deltas1	(β, 38, 27, 120)	18468	9	(1,1)	(1,1)	*valid*	*linear*

Total params: 1,203,245
Trainable params: 1,203,245
Non-trainable params: 0

Hence, there are two RPN models, one with architecture 4 as input and the other with architecture 5 as input. As an epoch of this model takes a long time, it was decided to train both models for 5 days and see how far they would come. Both models successfully finished 170 epochs. The training losses are depicted in Figure 42. The figures show a rapid decline in loss and a possible stabilisation after 16 epochs. Figure 43 shows the loss functions from epoch 50, which is shown to further investigate the loss functions. This plot shows that the loss functions are slowly stabilising, where the loss function of the model with CAE architecture 4 performs best.



(a) Loss Architecture 4

(b) Loss Architecture 5

Figure 42: Training loss of the two RPN models. The left figure shows the RPN model with input architecture 4 (A4) and the right figure shown the RPN model with input architecture 5 (A5).

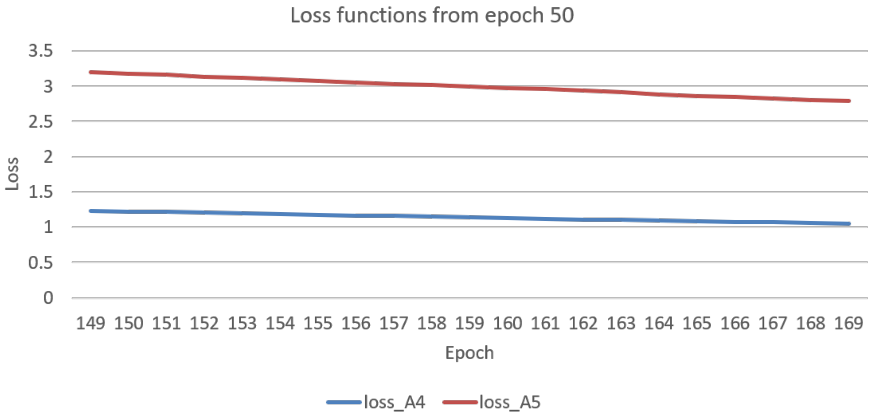


Figure 43: Both RPN models training loss from 50 epochs.

The loss function shows a very small loss; however, a low loss does not always lead to good region proposals. As the proposals must have an IoU of at least 70%, each proposal set must be checked before it can be passed to the next model. This is done to ensure that each class is represented at least once, otherwise, it cannot be located. After testing each image in the train set, none of the images had enough relevant proposals to be passed to the next stage. Hence, it was not possible to train the final stage of the RCNN model.

As the RPN did not lead to accurate region proposals it was decided to add a type map to the input. However, the computational requirements and time were not feasible for the time and memory that was left. Hence, the implementation is not tested.

7.3.3 Classification

The next step is to classify the proposed foreground regions as one of the 5 region classes. As mentioned before, like the RPN model, each class must have at least one proposed region with an IoU score of at least 70%. However, after 5 days of training, neither model resulted in any sufficient region proposals. Hence, this model was implemented, but could not be trained. The final implementation is shown in Table 20 and a visualisation of the full model can be seen n Figure 48 in Appendix E.

Table 20: RCNN model

Model: "RCNN:classifier"					
Layer (type)	Name	Output Shape	Param #	f	$h(\cdot)$
InputLayer	input_11	(None, 30, 27, 256)	0		
InputLayer	input_12	[(None, 4)]	0		
InputLayer	input_13	[(None, 1)]	0		
RoIPooling	roi_pooling	(None, 7, 7, 256)	0	(7,7)	
Flatten	flatten	(None, 12544)	0		
Dense	fc1	(None, 4096)	51384320		*relu*
Dense	fc2	(None, 4096)	16781312		*relu*
BatchNormalization	batch_normalization	(None, 4096)	16384		
Dense	scores2	(None, 5)	24582		*softmax*
Dense	deltas2	(None, 20)	98328		*linear*

Total params: 1,257,110
Trainable params: 1,257,110
Non-trainable params: 0

Therefore, it was decided to find the potential performance. This was done by performing an ablation study. An ablation study is the study of a model, where some features or modelling parts are excluded, to see how it affects the performance. Here, the RCNN modelling stages are excluded and the ground truth regions are used instead. As was mentioned in Section 6.3, the next step is to find the specific fields, which can be done in many ways. It was decided to use the Rule-Based approach, as LightGBM previously did not give good results. The results are shown in Figure 44. As can be seen from the figure, the adjusted F1 and the F1 score have a large range of values, with an average of 14% and 36% respectively, while the average accuracy score is around 98%. This is lower than the Rule-Based algorithm, as not all regions are annotated.

7.4 Final output

The final output consists of 2 new tables in the database and an image with coloured regions. The two new tables are called Bijlage_details and Matching. Bijlage_details holds the discovered information on the invoice, while Matching matches the discovered information with the logged information in the database. An example of Bijlage_details is shown in Table 21, an example of Matching is shown in Table 22, and an example of the coloured image is shown in Figure 45.

These outputs are the results of one example invoice. This invoice is from Company A and has the main characteristics that make this research more difficult. Namely, the

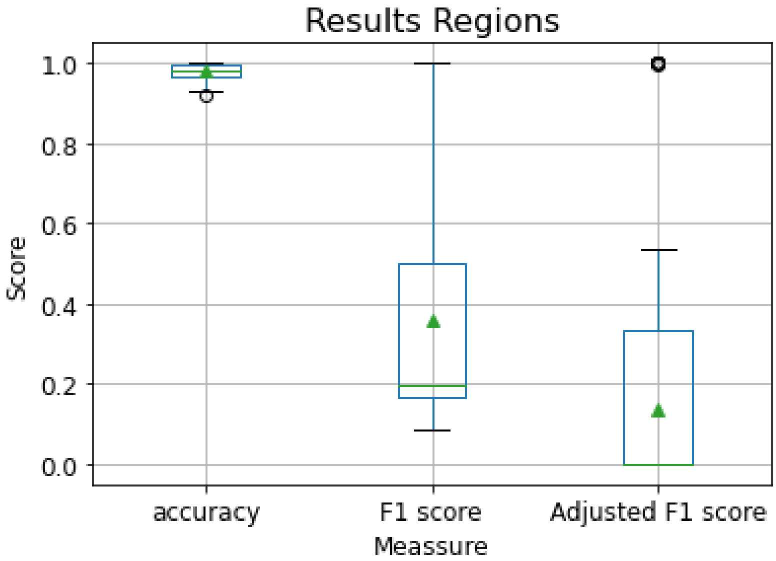


Figure 44: Test results ablation study RCNN model

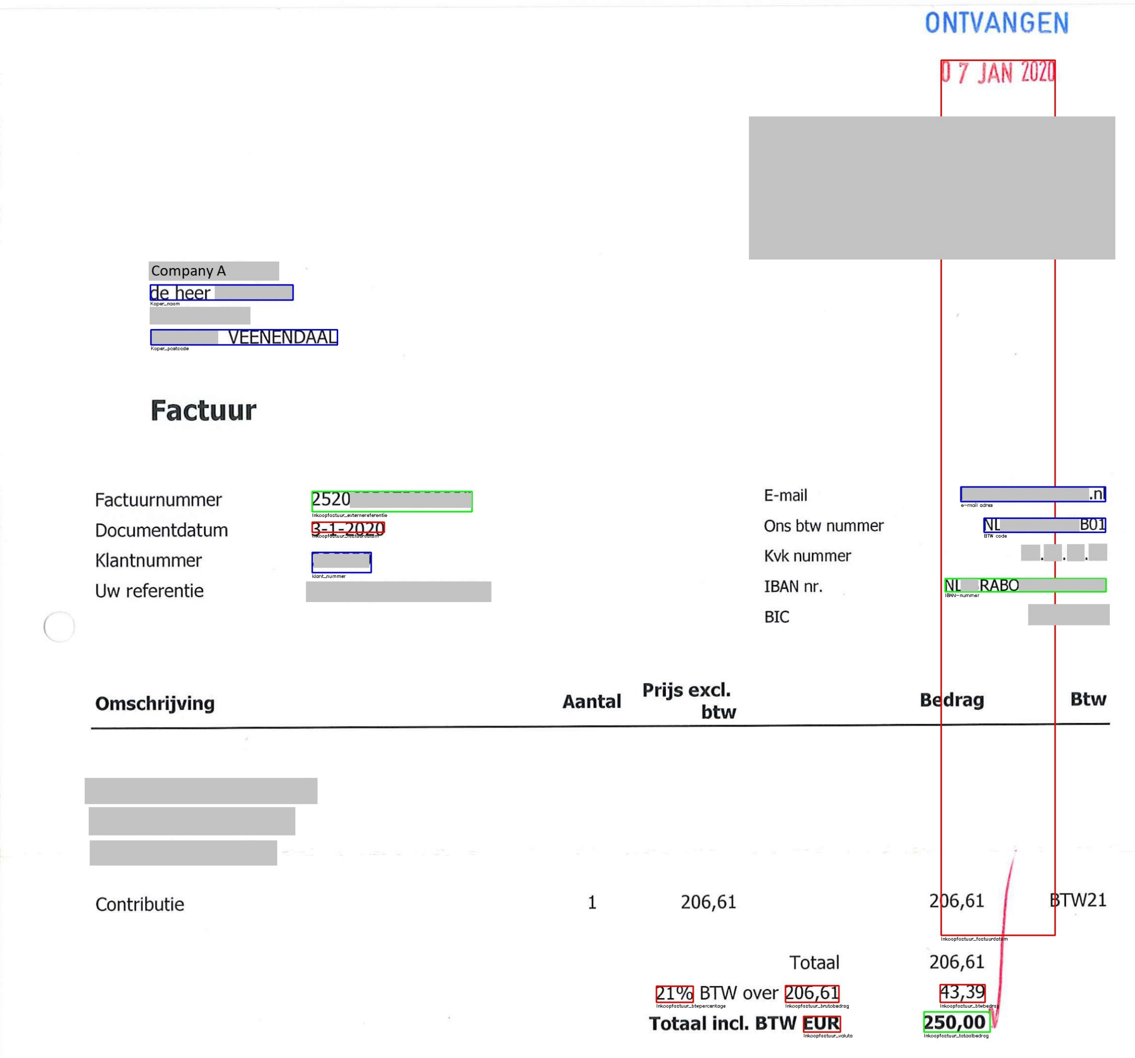
Table 21: Example output important columns in the Bijlage_details table of one invoice.

Bijlage_ID	Label	Tekst
304***	externereferentie	2520
304***	factuurdatum	3-1-2020
304***	brutobedrag	206.61
304***	korting	
304***	nettobedrag	
304***	btwpercentage	21
304***	btwbedrag	43.39
304***	totaalbedrag	250
304***	valuta	EUR
304***	IBAN-nummer	NL**RABO(...)
304***	Crediteurnaam	
304***	Crediteur_kvk	
304***	BTW_nummer	NL(...)B01

date stamp, not properly logging taxes, and the many NULL values in the database. This caused incomplete and inaccurate annotations and results. As can be seen from the figure, the found fields are mostly correct. However, due to the database counterparts, they are considered wrong proposals.

Table 22: Example output Matching table of one invoice.

Tabel_A	Kolom_A		Kans	Waarde_A	Waarde_B	In_Bron
Inkoopfactuur	factuurdatum		90	7-1-2020	3-1-2020	1
Inkoopfactuur	brutobedrag		38	250	206.61	1
Inkoopfactuur	korting		100	NULL		0
Inkoopfactuur	nettobedrag		0	250		1
Inkoopfactuur	btwpercentage	…	0	NULL	21	0
Inkoopfactuur	btwbedrag		17	0	43.39	1
Inkoopfactuur	totaalbedrag		100	250	250	1
Inkoopfactuur	valuta		0	NULL	EUR	0
Inkoopfactuur	externereferentie		100	2520	2520	1
Crediteur	omschrijving		0	**** BV		0
Crediteur	bankrekening		100	NL**RABO(...)	NL**RABO(...)	1



ONTVANGEN
07 JAN 2020

[redacted]

Company A
de heer [redacted]
[redacted]
[redacted] VEENENDAAL

Factuur

Factuurnummer	2520[redacted]
Documentdatum	3-1-2020
Klantnummer	[redacted]
Uw referentie	[redacted]

E-mail	[redacted].nl
Ons btw nummer	NL[redacted]B01
Kvk nummer	[redacted].[redacted].[redacted].[redacted]
IBAN nr.	NL[redacted]RABO[redacted]
BIC	[redacted]

Omschrijving	Aantal	Prijs excl. btw	Bedrag	Btw
[redacted]				
Contributie	1	206,61	206,61	BTW21

Totaal	206,61
21% BTW over 206,61	43,39
Totaal incl. BTW EUR	**250,00**

Koper_naam
Koper_postcode
Inkoopfactuur_externereferentie
Inkoopfactuur_factuurdatum
klant_nummer
e-mail adres
BTW code
IBAN-nummer
Inkoopfactuur_factuurdatum
Inkoopfactuur_btwpercentage
Inkoopfactuur_brutobedrag
Inkoopfactuur_btwbedrag
Inkoopfactuur_valuta
Inkoopfactuur_totaalbedrag

Figure 45: Example image output of one invoice, created by the Rule-Based algorithm.

8 Discussion

This section describes several limitations that were encountered during this research, and it will also suggest some further research opportunities. First, the limitations of the research will be elaborated on. Thereafter, the results of the models will be further explained. The third subsection will describe the impact on the host company. Finally, further research opportunities will be described.

8.1 Limitations

This research dealt with several limitations, which were mostly a direct cause of the lack of pre-annotated data. Even though there were a lot of PDFs available, many of them did not have a database counterpart that could be connected easily. Most of these invoices were the invoices of Company B, which were around 6000 invoices, where none of them had an existing link to the database. Hence, that link had to be established by using OCR to find the external reference number. However, as this method did not work for all invoices, almost 5000 of them were not linked.

Moreover, not all information in the database was one-to-one with the information on the invoice. Where some fields were not filled in at all, others were filled in incorrectly. Some of the more prominent fields that were not filled in properly were the tax amount, tax percentage, and gross amount of the invoices of Company A. This not only led to incomplete annotation of the specific fields, but it also led to incomplete region annotation. As a result, most of the invoices missed at least one of the regions that had to be annotated. This in turn also led to low F1 scores, as many of the correctly classified fields were not accurately annotated.

Other than missing many labels, the data was also severely imbalanced. As an invoice holds many words that are not considered important information, most of the proposed fields were deemed unimportant. This not only led to high accuracy scores, but it also led to bad F1 scores.

As all models would have performed better with a fully annotated dataset, neither model performed to its full potential with this dataset. Hence, it is impossible to say which model is better or which has the most potential. However, by ignoring the potential false negatives and comparing the results based on the adjusted F1-score, the Rule-Based algorithm performed best.

8.2 Results

This section will give a more extensive explanation of the results of the three separate modelling techniques. As the results depend on the level of annotation, the averages are quite low. However, as the classes related to taxes and currency are not properly annotated, and many invoice dates were logged in incorrectly due to date stamps, 5 of the 13 important classes only brought the average down. Furthermore, many of the other classes were only partly annotated, which further explains the low scores.

Rule-Based

The first tested model was the Rule-Based model, which is also the model with the highest average adjusted F1 score, which was 44%. At first glance this seems quite low, however, as around 40 to 60 percent of the annotation was done, this score is actually quite high. Moreover, after visually inspecting the results, the discovered fields were quite good as well; an example is shown in Figure 45. This can be explained by the fact that the Rule-Based algorithm is not dependent on the level of annotation, as it is a static method that does not learn from its mistakes.

LightGBM

Other than the Rule-based algorithm, the LightGBM model is highly dependent on having a large variety of annotated samples per class. Hence, upsampling and downsampling techniques were used. However, as some classes had less than 200 samples, where some even had less than 20, upsampling them to 5000 samples led to the same samples being represented more than 50 times. This overrepresentation led to severe overfitting. Where some of the LightGBM models started to resemble template matching algorithms.

However, downsampling also led to bad results, as many of the samples that were discarded are similar to the upsampled classes. As a result, many unimportant fields were mistaken to be part of an important class, which also led to much lower accuracy scores. Furthermore, all classes had less than 2000 samples. As there were around 4000 invoices, many correctly annotated fields were flooded by the fields that were not annotated. This in turn led to more fields being labelled as unimportant. This also explains the bad results after using SMOTE, where a lot of the created samples seemed to resemble fields in the unimportant class.

RCNN

The RCNN model had limitations due to the annotation as well as limitations due to its computational time. As an Artificial Neural Network learns from its mistakes, it is highly dependent on the annotation of its input data. So as many invoices did not have all regions, the model was not able to distinguish when a textual area was foreground and when it could be considered as background.

Moreover, the training of the RPN model was very computationally expensive, which was largely due to the input size. As the model was trained on an external server, Lisa by SURFsara, the training time had a maximum of five days, which was around 170 epochs per model. After further inspecting the loss functions of the RPN models, it could be seen that the losses dropped around 0.01 per epoch, which was reducing per epoch as well. This indicates that further training both RPN models could easily take 2 more weeks, which also had to be followed by at least a week of training the classification part. Hence, this did not seem feasible for this research. As a result, tuning the hyper-parameters was beyond the feasibility of this research. Moreover, by adding the type maps, the computational time increased even more. Therefore, this model was not tested further either.

This computational time could have been decreased hugely by decreasing the size of the input images. However, as Table 17 and Figure 41 showed, this not only led to the possibility of overlapping boxes, but it also led to no boxes having an IoU higher than 70%. This in turn would never result in feasible foreground regions.

The computational time also led to the decision to train the RCNN model instead of the Faster-RCNN model. Not only would this model need a lot more time per epoch, but it also required more memory to train it. As initialising and reading the data led to a memory error, it was decided to train the stages separately.

8.3 Impact on the host company

The impact of this research on the host company is of great value. As the Rule-Based method is already implemented to work with the SoliTrust datamodel and the invoices directly, it is already classifying whether the invoices (in PDF format) are one-to-one with the data in the datamodel. This not only saves time, but it is also more effective than just looking at a small subsample.

As it is only implemented for Company A, the potential of using it for more clients is still great as well. Furthermore, this strategy can also be adjusted to work with other VRD documents, such as bank statements and pay checks. As a result, many tedious tasks can be automated in the future.

8.4 Further research

There are several further research opportunities from this research. Not only can the RCNN model be further improved by training it with the type maps, but if memory allows it, the Faster-RCNN version could also be trained. This could lead to much better results. Furthermore, acquiring more data (fully and partly annotated) could also enhance the performance of the RCNN and the LightGBM model.

Furthermore, instead of trying to immediately find the five different regions, the model could be trained to first distinguish tables from paragraphs. After that, another model could be used to separate the paragraphs into the different regions.

Another potential research option would be to work with the pre-existing and pre-trained models in the Transformers library (by [Wolf et al., 2020]), e.g., BERT and LayoutLM. As these models had state-of-the-art results in supervised settings, they could have a good performance in semi-supervised settings as well. Furthermore, as they are already extensively trained on the layout of VRDs they could be less sensitive to semi-supervised data.

Further research could also focus on adding logical and probabilistic reasoning to the model, like [Manhaeve et al., 2018]. Examples of logical rules could be that the addition of the tax amount and the gross amount should be equal to the total amount. This could not only make a model smarter, but it could also lead to much better results. Another approach could be to make a model that aligns the invoices with the grid of the feature space, as it could decrease the chance of overlapping regions.

9 Conclusion

The main goal of this research was to answer the research question: *Can a traditional object detection model, such as (Faster-)(R)CNN, still be applicable in invoice information extraction when the data is not fully supervised or is a Rule-Based or a decision tree model more effective?* The goal was to examine whether state-of-the-art models have similar results when data annotation is scarce, as all the recent models assume perfect annotation.

With the follow-up questions: *Does adding textual features, which are produced by heuristics and an OCR engine, to the input of the aforementioned model increase the IE performance?*, and *Does replacing the normal convolutional layers in the (Faster-)(R)CNN architecture to atrous convolutional layers lead to better IE results?*

Before comparing the models, it is important to note that neither model could perform at its best, since the data annotation was not complete. Moreover, as the RPN model was not able to find appropriate regions, the RCNN results are based on an ablation study, where the correct regions are assumed to be proposed. Hence, the current RCNN model, excluding the ablation study, performed the worst, as it did not extract any information.

In order to decide which of the three models performed best, two performance measures are used: the F1-score and an adjusted F1-score. As the dataset was severely imbalanced, the accuracy was not applicable, whereas the F1 score gives less biased towards the majority. The results of the Rule-Based, LightGBM, and RCNN models can be seen in Figure 34, 35 and 44 respectively. As was expected the F1 scores were much lower than the accuracy scores, namely around 42%, 29%, and 36% for the Rule-Based, LightGBM, and RCNN models, respectively. This indicates that the Rule-Based model performed best overall.

To decide which model performed best in the eyes of the host company, an adjusted F1-score is used. This score only gives weight to the F1 scores concerning the classes that were classified as important. Here the models scored around 44%, 18%, and 14% for the Rule-Based, LightGBM, and RCNN models, respectively. This indicates that the best model here would be the Rule-Based model as well. This can be explained by the fact that not all regions are annotated. Hence, the RCNN ablation study was expected to have a lower score than the Rule-Based model.

The next question could not be answered yet. However, it does have potential, as multiple researchers have stated that textual features increase the performance and because there is no current performance. Hence, when there is enough capacity to train this model, it would likely produce better results.

The last question could also not be answered yet, as both the architecture without atrous convolutional layers and the architecture with atrous convolutional layers did not lead to appropriate regions. Furthermore, as can be seen from Figure 39, the CAE losses are too similar to decide which CAE architecture would be more applicable. However, from Figure 42 it could be concluded that the RPN model with Architecture 4 as its input performs better than the RPN model with Architecture 5 as input, as its loss is slightly lower. However, as both RPN models failed to produce appropriate regions, neither is good.

In short, to answer the main research question, yes, the (Faster-)(R)CNN is still applicable, however, Rule-Based methods are more likely to produce more promising results, as they are not dependent on the annotation of the data. This was expected, as Neural Networks learn from the mistakes they make when classifying a set of input data. Hence, when a region is found to be foreground, but it is not correctly annotated, the model will not learn that it is foreground.

Therefore, when using unsupervised data, it is recommended to use a Rule-Based model, as it has higher scores, and it takes much less time to perform, as it does not have to be trained. However, when a better performing model is preferred, then it is recommended to first fully annotate the data, as that would lead to much better results.

References

[Abadi et al., 2016] Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., et al. (2016). Tensorflow: A system for large-scale machine learning. In *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, pages 265–283.

[Acuña et al., 2019] Acuña, B., Martini, L. C., Motta, L. L., Larco, J., and Grijalva, F. (2019). Table Detection for Improving Accessibility of Digital Documents using a Deep Learning Approach. In *2019 IEEE Latin American Conference on Computational Intelligence (LA-CCI)*, pages 1–6.

[Aghdam and Heravi, 2017] Aghdam, H. H. and Heravi, E. J. (2017). *Guide to Convolutional Neural Networks: A Practical Application to Traffic-Sign Detection and Classification*. Springer Publishing Company, Incorporated, 1st edition.

[Al Daoud, 2019] Al Daoud, E. (2019). Comparison between XGBoost, LightGBM and CatBoost using a home credit dataset. *International Journal of Computer and Information Engineering*, 13(1):6–10.

[Al-Shari et al., 2021] Al-Shari, H., Saleh, Y., and Odabas, A. (2021). Comparison of Gradient Boosting Decision Tree Algorithms for CPU Performance. *Erciyes Tip Dergisi*, pages 157–168.

[Bhattacharjee et al., 2020] Bhattacharjee, A., Borgohain, S. K., Soni, B., Verma, G., and Gao, X.-Z. (2020). *Machine Learning, Image Processing, Network Security and Data Sciences*. Springer.

[Bird et al., 2009] Bird, S., Klein, E., and Loper, E. (2009). *Natural language processing with Python: analyzing text with the natural language toolkit*. " O'Reilly Media, Inc.".

[Bishop, 2007] Bishop, C. M. (2007). *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer, 1 edition.

[Boerse et al., 2017] Boerse, W., Heslinga, D. J., and Schauten, W. (2017). *Boekhouden geboekstaafd*. Noordhoff Uitgevers bv.

[Bowyer et al., 2011] Bowyer, K. W., Chawla, N. V., Hall, L. O., and Kegelmeyer, W. P. (2011). SMOTE: Synthetic Minority Over-sampling Technique. *CoRR*, abs/1106.1813. http://arxiv.org/abs/1106.1813.

[Bradski and et al., 2021] Bradski, G. and et al. (2021). OpenCV. https://opencv.org/.

[Brauer et al., 2011] Brauer, F., Rieger, R., Mocan, A., and Barczynski, W. M. (2011). Enabling Information Extraction by Inference of Regular Expressions from Sample Entities. CIKM '11, page 1285–1294, New York, NY, USA. Association for Computing Machinery. https://doi.org/10.1145/2063576.2063763.

[Chen et al., 2018] Chen, L.-C., Papandreou, G., Kokkinos, I., Murphy, K., and Yuille, A. L. (2018). DeepLab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 40(4):834–848.

[Chollet, 2016] Chollet, F. (2016). Building autoencoders in Keras. https://blog.keras.io/building-autoencoders-in-keras.html.

[Chollet et al., 2015] Chollet, F. et al. (2015). Keras.

[Davies, 2021] Davies, N. B. (2021). A4 Paper Dimensions. https://www.graphic-design-employment.com/a4-paper-dimensions.html.

[Devlin et al., 2018] Devlin, J., Chang, M., Lee, K., and Toutanova, K. (2018). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *CoRR*, abs/1810.04805. http://arxiv.org/abs/1810.04805.

[Eikvil, 1993] Eikvil, L. (1993). OCR - Optical Character Recognition. https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.217.4980&rep=rep1&type=pdf.

[et al., 2019] et al., S. B. (2019). Deskew. https://github.com/sbrunner/deskew.

[Felzenszwalb et al., 2010] Felzenszwalb, P. F., Girshick, R. B., McAllester, D., and Ramanan, D. (2010). Object Detection with Discriminatively Trained Part-Based Models. *IEEE Trans. Pattern Anal. Mach. Intell.*, 32(9):1627–1645. https://doi.org/10.1109/TPAMI.2009.167.

[Garncarek et al., 2020] Garncarek, L., Powalski, R., Stanislawek, T., Topolski, B., Halama, P., and Gralinski, F. (2020). LAMBERT: Layout-Aware language Modeling using BERT for information extraction. *CoRR*, abs/2002.08087. https://arxiv.org/abs/2002.08087.

[Ghosh, 2021] Ghosh, S. (2021). Invoice Information extraction using OCR and Deep Learning.

[Girshick et al., 2013] Girshick, R. B., Donahue, J., Darrell, T., and Malik, J. (2013). Rich feature hierarchies for accurate object detection and semantic segmentation. *CoRR*, abs/1311.2524. http://arxiv.org/abs/1311.2524.

[Goodfellow et al., 2016] Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press. `http://www.deeplearningbook.org`.

[Google inc, 2019] Google inc (2019). tesseract-ocr/tesseract. https://github.com/tesseract-ocr/tesseract.

[Google inc, 2021] Google inc (2021). Improving the quality of the output. https://tesseract-ocr.github.io/tessdoc/ImproveQuality.html.

[Goyal et al., 2018] Goyal, P., Pandey, S., and Jain, K. (2018). *Deep Learning for Natural Language Processing*. Apress, Berkeley, CA.

[He et al., 2015] He, K., Zhang, X., Ren, S., and Sun, J. (2015). Deep Residual Learning for Image Recognition. *CoRR*, abs/1512.03385. http://arxiv.org/abs/1512.03385.

[Hong et al., 2021] Hong, T., Kim, D., Ji, M., Hwang, W., Nam, D., and Park, S. (2021). {BROS}: A Pre-trained Language Model for Understanding Texts in Document. https://openreview.net/forum?id=punMXQEsPr0.

[Honnibal and Montani, 2017] Honnibal, M. and Montani, I. (2017). spaCy 2: Natural language understanding with Bloom embeddings, convolutional neural networks and incremental parsing. To appear.

[Hoque et al., 2020] Hoque, O. B., Rashid, M. B., and Jawad, K. M. T. (2020). Autonomous Deblurring Images and Information Extraction from Documents Using CycleGAN and Mask RCNN. In *2020 23rd International Conference on Computer and Information Technology (ICCIT)*, pages 1–6.

[Huang et al., 2019] Huang, Z., Chen, K., He, J., Bai, X., Karatzas, D., Lu, S., and Jawahar, C. V. (2019). ICDAR2019 Competition on Scanned Receipt OCR and Information Extraction. In *2019 International Conference on Document Analysis and Recognition (ICDAR)*, pages 1516–1520.

[Janev et al., 2020] Janev, V., Graux, D., Jabeen, H., and Sallinger, E. (2020). *Knowledge Graphs and Big Data Processing*. Springer, Cham, 1 edition.

[Jiang et al., 2020] Jiang, S., Qin, H., Zhang, B., and Zheng, J. (2020). Optimized Loss Functions for Object detection: A Case Study on Nighttime Vehicle Detection. *CoRR*, abs/2011.05523. https://arxiv.org/abs/2011.05523.

[Jiang et al., 2019] Jiang, Z., Huang, Z., Lian, Y., Guo, J., and Qiu, W. (2019). Integrating Coordinates with Context for Information Extraction in Document Images. In *2019 International Conference on Document Analysis and Recognition (ICDAR)*, pages 363–368.

[Ju et al., 2019] Ju, Y., Sun, G., Chen, Q., Zhang, M., Zhu, H., and Rehman, M. U. (2019). A model combining convolutional neural network and LightGBM algorithm for ultra-short-term wind power forecasting. *Ieee Access*, 7:28309–28318.

[Jun et al., 2019] Jun, C., Suhua, Y., and Shaofeng, J. (2019). Automatic classification and recognition of complex documents based on Faster RCNN. In *2019 14th IEEE International Conference on Electronic Measurement Instruments (ICEMI)*, pages 573–577.

[Kang et al., 2014] Kang, L., Kumar, J., Ye, P., Li, Y., and Doermann, D. (2014). Convolutional Neural Networks for Document Image Classification. In *2014 22nd International Conference on Pattern Recognition*, pages 3168–3172.

[Kazdar et al., 2019] Kazdar, T., Mseddi, W. S., Jmal, M., and Attia, R. (2019). What can RCNN bring to STEG invoice image? *The 1st Tunisian SMART CITIES Symposium*.

[Ke et al., 2017] Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., Ye, Q., and Liu, T.-Y. (2017). LightGBM: A Highly Efficient Gradient Boosting Decision Tree. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 3149–3157, Red Hook, NY, USA. Curran Associates Inc.

[Khoshrou and Pauwels, 2019] Khoshrou, A. and Pauwels, E. J. (2019). Short-term scenario-based probabilistic load forecasting: A data-driven approach. *Applied Energy*, 238:1258–1268. https://www.sciencedirect.com/science/article/pii/S0306261919301412.

[Kim and Moldovan, 1993] Kim, J.-T. and Moldovan, D. (1993). Acquisition of semantic patterns for information extraction from corpora. In *Proceedings of 9th IEEE Conference on Artificial Intelligence for Applications*, pages 171–176.

[Kingma and Ba, 2015] Kingma, D. P. and Ba, J. (2015). Adam: A Method for Stochastic Optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. http://arxiv.org/abs/1412.6980.

[Kipf and Welling, 2016] Kipf, T. N. and Welling, M. (2016). Semi-Supervised Classification with Graph Convolutional Networks. *arXiv preprint arXiv:1609.02907*.

[Krizhevsky et al., 2012] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). ImageNet Classification with Deep Convolutional Neural Networks. In *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc. https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf.

[Lemaître et al., 2017] Lemaître, G., Nogueira, F., and Aridas, C. K. (2017). Imbalanced-learn: A python toolbox to tackle the curse of imbalanced datasets in machine learning. *Journal of Machine Learning Research*, 18(17):1–5. http://jmlr.org/papers/v18/16-365.

[Leskovec et al., 2014] Leskovec, J., Rajaraman, A., and Ullman, J. D. (2014). *Mining of Massive Datasets*. Cambridge University Press, USA, 2nd edition.

[Liu et al., 2019a] Liu, X., Gao, F., Zhang, Q., and Zhao, H. (2019a). Graph Convolution for Multimodal Information Extraction from Visually Rich Documents. *CoRR*, abs/1903.11279. http://arxiv.org/abs/1903.11279.

[Liu et al., 2019b] Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., and Stoyanov, V. (2019b). Roberta: A robustly optimized BERT pretraining approach. *CoRR*, abs/1907.11692.

[Maggipinto et al., 2018] Maggipinto, M., Masiero, C., Beghi, A., and Susto, G. A. (2018). A Convolutional Autoencoder Approach for Feature Extraction in Virtual Metrology. *Procedia Manufacturing*, 17:126–133.

[Manhaeve et al., 2018] Manhaeve, R., Dumancic, S., Kimmig, A., Demeester, T., and Raedt, L. D. (2018). DeepProbLog: Neural Probabilistic Logic Programming. *CoRR*, abs/1805.10872. http://arxiv.org/abs/1805.10872.

[Marinai, 2008] Marinai, S. (2008). *Introduction to Document Analysis and Recognition*, pages 1–20. Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-540-76280-5_-1.

[McKie and Liu, 2016] McKie, J. X. and Liu, R. (2016). PyMuPDF. https://github.com/pymupdf/PyMuPDF.

[Naseer and Zafar, 2019] Naseer, A. and Zafar, K. (2019). Meta features-based scale invariant OCR decision making using LSTM-RNN. *Comput. Math. Organ. Theory*, 25(2):165–183. https://doi.org/10.1007/s10588-018-9265-9.

[Nayak, 2018] Nayak, S. (2018). Understanding AlexNet. https://learnopencv.com/understanding-alexnet/.

[Nguyen et al., 2019] Nguyen, N.-V., Rigaud, C., and Burie, J.-C. (2019). Semi-supervised Object Detection with Unlabeled Data. https://www.scitepress.org/Papers/2019/73456/73456.pdf.

[Niblack, 1986] Niblack, W. (1986). *An Introduction to Digital Image Processing*. Delaware Symposia on Language Studies5. Prentice-Hall International.

[Oksuz et al., 2020] Oksuz, K., Cam, B. C., Kalkan, S., and Akbas, E. (2020). Imbalance problems in object detection: A review. *IEEE transactions on pattern analysis and machine intelligence*.

[Patel and Bhatt, 2020] Patel, S. and Bhatt, D. (2020). Abstractive Information Extraction from Scanned Invoices (AIESI) using End-to-end Sequential Approach. *CoRR*, abs/2009.05728. https://arxiv.org/abs/2009.05728.

[Pedregosa et al., 2011] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., et al. (2011). Scikit-learn: Machine learning in python. *Journal of machine learning research*, 12(Oct):2825–2830.

[Pham, 2020] Pham, C. (2020). Graph Convolutional Networks (GCN). https://www.topbots.com/graph-convolutional-networks/.

[Powalski et al., 2021] Powalski, R., Borchmann, L., Jurkiewicz, D., Dwojak, T., Pietruszka, M., and Palka, G. (2021). Going Full-TILT Boogie on Document Understanding with Text-Image-Layout Transformer. *CoRR*, abs/2102.09550. https://arxiv.org/abs/2102.09550.

[Quiñonero-Candela et al., 2008] Quiñonero-Candela, J., Sugiyama, M., Schwaighofer, A., and D., L. N. (2008). *Dataset Shift in Machine Learning*. The MIT Press. https://doi.org/10.7551/mitpress/9780262170055.001.0001.

[Rao et al., 2019] Rao, H., Shi, X., Rodrigue, A. K., Feng, J., Xia, Y., Elhoseny, M., Yuan, X., and Gu, L. (2019). Feature selection based on artificial bee colony and gradient boosting decision tree. *Applied Soft Computing*, 74:634–642. https://www.sciencedirect.com/science/article/pii/S1568494618305933.

[Rastogi et al., 2020] Rastogi, M., Ali, S. A., Rawat, M., Vig, L., Agarwal, P., Shroff, G., and Srinivasan, A. (2020). Information Extraction From Document Images via FCA-Based Template Detection and Knowledge Graph Rule Induction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*.

[Redmon et al., 2015] Redmon, J., Divvala, S. K., Girshick, R. B., and Farhadi, A. (2015). You Only Look Once: Unified, Real-Time Object Detection. *CoRR*, abs/1506.02640. http://arxiv.org/abs/1506.02640.

[Ren et al., 2015] Ren, S., He, K., Girshick, R., and Sun, J. (2015). Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. In *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc. https://proceedings.neurips.cc/paper/2015/file/14bfa6bb14875e45bba028a21ed38046-Paper.pdf.

[Riloff, 1993] Riloff, E. (1993). Automatically Constructing a Dictionary for Information Extraction Tasks. In *Proceedings of the Eleventh National Conference on Artificial Intelligence*, AAAI'93, page 811–816. AAAI Press.

[Salakhutdinov and Hinton, 2009] Salakhutdinov, R. and Hinton, G. E. (2009). Semantic hashing. *Int. J. Approx. Reason.*, 50:969–978.

[Sauvola et al., 1997] Sauvola, J., Seppänen, T., Haapakoski, S., and Pietikäinen, M. (1997). Adaptive Document Binarization. In *Pattern Recognition*, volume 33, pages 147–152 vol.1.

[Schuster and Paliwal, 1997] Schuster, M. and Paliwal, K. (1997). Bidirectional Recurrent Neural Networks. *Trans. Sig. Proc.*, 45:2673–2681. http://dx.doi.org/10.1109/78.650093.

[Sezgin and Sankur, 2004] Sezgin, M. and Sankur, B. (2004). Survey over image thresholding techniques and quantitative performance evaluation. *Journal of Electronic imaging*, 13(1):146–165.

[Shi et al., 2015] Shi, B., Bai, X., and Yao, C. (2015). An End-to-End Trainable Neural Network for Image-based Sequence Recognition and Its Application to Scene Text Recognition. *CoRR*, abs/1507.05717. http://arxiv.org/abs/1507.05717.

[Simonyan and Zisserman, 2015] Simonyan, K. and Zisserman, A. (2015). Very Deep Convolutional Networks for Large-Scale Image Recognition.

[Tarawneh et al., 2019] Tarawneh, A. S., Hassanat, A. B., Chetverikov, D., Lendak, I., and Verma, C. (2019). Invoice Classification Using Deep Features and Machine Learning Techniques. In *2019 IEEE Jordan International Joint Conference on Electrical Engineering and Information Technology (JEEIT)*, pages 855–859.

[Trier and Jain, 1995] Trier, O. D. and Jain, A. K. (1995). Goal-directed evaluation of binarization methods. *IEEE transactions on Pattern analysis and Machine Intelligence*, 17(12):1191–1201.

[Urbani, 2020] Urbani, J. (2020). Lecture notes Web Data Processing Systems (XM_-40020). https://studiegids.vu.nl/en/Master/2020-2021/business-analytics/XM_40020.

[Van der Walt et al., 2014] Van der Walt, S., Schönberger, J. L., Nunez-Iglesias, J., Boulogne, F., Warner, J. D., Yager, N., Gouillart, E., and Yu, T. (2014). scikit-image: image processing in python. *PeerJ*, 2:e453.

[Vaswani et al., 2017] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L. u., and Polosukhin, I. (2017). Attention is All you Need. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

[Wolf et al., 2020] Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M., Davison, J., Shleifer, S., von Platen, P., Ma, C., Jernite, Y., Plu, J., Xu, C., Scao, T. L., Gugger, S., Drame, M., Lhoest, Q., and Rush, A. M. (2020). Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online. Association for Computational Linguistics. https://www.aclweb.org/anthology/2020.emnlp-demos.6.

[Xu et al., 2019] Xu, Y., Li, M., Cui, L., Huang, S., Wei, F., and Zhou, M. (2019). LayoutLM: Pre-training of Text and Layout for Document Image Understanding. *CoRR*, abs/1912.13318. http://arxiv.org/abs/1912.13318.

[Yang and Zhang, 2018] Yang, S. and Zhang, H. (2018). Comparison of several data mining methods in credit card default prediction. *Intelligent Information Management*, 10(5):115–122.

[Yu et al., 2020] Yu, W., Lu, N., Qi, X., Gong, P., and Xiao, R. (2020). PICK: Processing Key Information Extraction from Documents using Improved Graph Learning-Convolutional Networks. *CoRR*, abs/2004.07464. https://arxiv.org/abs/2004.07464.

[Zelic and Sable, 2020] Zelic, F. and Sable, A. (2020). A comprehensive guide to OCR with Tesseract, OpenCV and Python. https://nanonets.com/blog/ocr-with-tesseract/.

[Zhao et al., 2019] Zhao, X., Niu, E., Wu, Z., and Wang, X. (2019). CUTIE: Learning to Understand Documents with Convolutional Universal Text Information Extractor.

[Zhu et al., 2015] Zhu, Y., Kiros, R., Zemel, R., Salakhutdinov, R., Urtasun, R., Torralba, A., and Fidler, S. (2015). Aligning books and movies: Towards story-like visual explanations by watching movies and reading books. *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 19–27.

[Zisou et al., 2020] Zisou, C., Sochopoulos, A., and Kitsios, K. (2020). Convolutional recurrent neural network and lightgbm ensemble model for 12-lead ecg classification. In *2020 Computing in Cardiology*, pages 1–4. IEEE.

Appendix

A Database

The invoice table and the sales invoice tables have many empty fields. The invoice table in the Company A database has 53 columns and 9285 rows and the table of Company B database has 52 columns and 16132 rows. The number of columns that are not recorded is 32 and 26, the number of partly filled in columns is 4 and 2, and there are 17 and 28 columns that are always recorded, for Company A and Company B respectively. A representation of these numbers is depicted in Figure 46.

The sales invoices table includes similar fields to the invoice table, and it also has many empty fields. This table has 58 columns and 31396 rows, where 26 of the columns are empty, 2 are partly filled in and 28 are completely filled, which is also depicted in Figure 46.

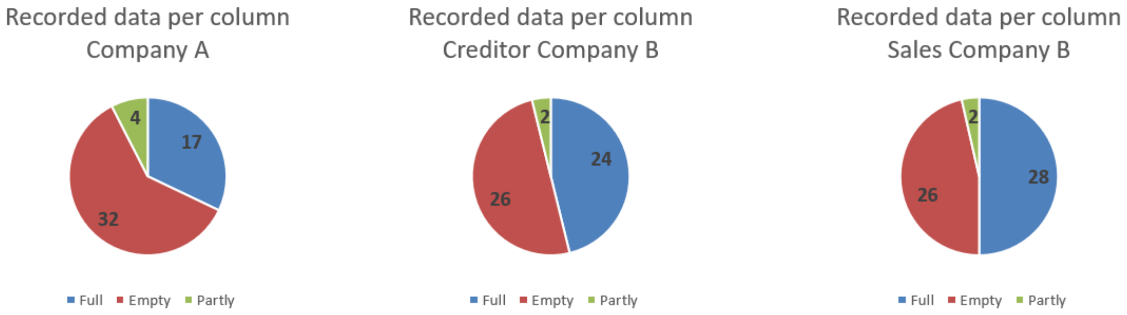


Figure 46: Distribution of recorded and non-recorded data

An example of 1 database entry of an invoice is shown in Table 23. As can be seen, many fields are empty.

Table 23: Example of SoliTrust Database entry

_Bedrijf_code	_nr	_regel	_factuurdatum	_boekjaar
1	20202716	None	4-12-2020 00:00	2020

_periode	_type	_regelsoort	_Crediteur_code	_landverzending
12	D	GROOTBOEKREKENING	51277	None

_Artikel_code	_Grootboekrekening_code	_omschrijving	_aantal	_eenheid
None	442999	Boeken en leermiddelen	None	None

_aantaluniform	_eenheiduniform	_prijs	_brutobedrag	_korting
None	None	None	417.500	None

_nettobedrag	_btwcode	_btwtype	_btwpercentage	_btwbedrag
417.500	None	None	None	0.0000

_bpmbedrag	_totaalbedrag	_valuta	_koers	_vvpwaarde
None	417.500	None	None	None

_activiteit	_Inkooporder_nr	_Inkooporder_regel	_Inkoopontvangst_nr	_Inkoopontvangst_regel
OVERIG	None	None	None	None

_Artikelmutatie_document	_Project_code	_kostenplaats	_contractperiode_van	_contractperiode_tot
None	None	None	None	None

_Object_code	_boekstuk	_uren_Medewerker_code	_uren_periode	_invoerdatum
None	202032612	None	None	None

_invoer_Medewerker_code	_autorisatiedatum	_autorisatie_Medewerker_code	_externereferentie	_Lot_code
34672.azo	None	None	202032612	None

_bankrekening	_systeem	_bijlage_id
None	None	320742

B Thresholding

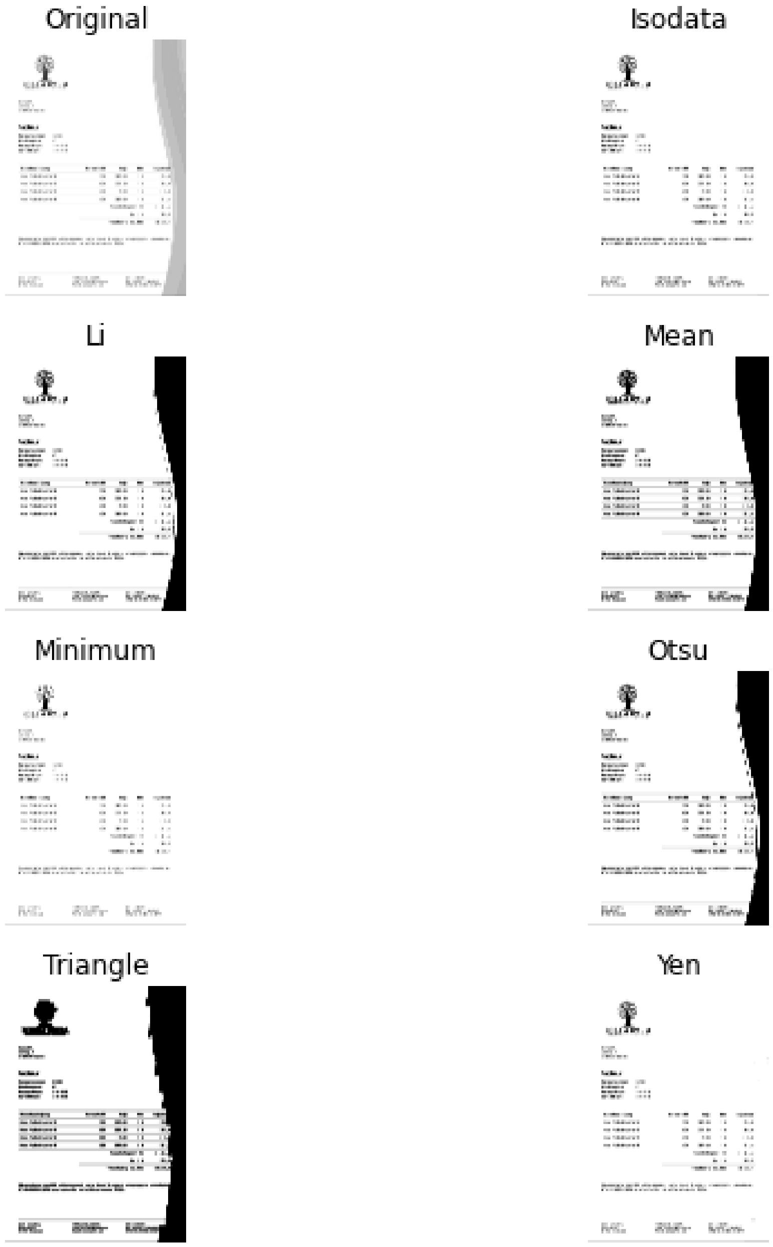


Figure 47: All thresholding methods in the module `skimage`.

C CAE architectures

Table 24: Architecture 1, inspired by [Kang et al., 2014]

Model: "cae_150"

Layer (type)	Name	Output Shape	Param #	k	f	s	p
InputLayer	Input	(β, 150, 150, 3)	0				
Conv2D	conv_1	(β, 144, 144, 20)	2960	20	(7,7)	(1,1)	*valid*
MaxPooling2D	pool_1	(β, 36, 36, 20)	0		(4,4)	(1,1)	
Conv2D	conv_2	(β, 27, 27, 256)	25050	50	(5,5)	(1,1)	*valid*
MaxPooling2D	pool_2	(β, 8, 8, 50)	0		(4,4)	(1,1)	
UpSampling2D	up_1	(β, 27, 27, 256)	0		(4,4)	(1,1)	
Conv2D	deconv_1	(β, 36, 36, 20)	25020	20	(5,5)	(1,1)	*valid*
UpSampling2D	up_2	(β, 144, 144, 20)	0		(4,4)	(1,1)	
Conv2D	deconv_2	(β, 150, 150, 3)	2943	3	(7,7)	(1,1)	*valid*

Total params: 55,973
Trainable params: 55,973
Non-trainable params: 0

Table 25: Architecture 2, inspired by AlexNet

Model: "AlexNet_1"

Layer (type)	Name	Output Shape	Param #	k	f	s	p
InputLayer	Input	(β, 227, 227, 3)	0				
Conv2D	conv_1	(β, 55, 55, 96)	34944	96	(11,11)	(4,4)	*valid*
MaxPooling2D	pool_1	(β, 27, 27, 96)	0		(3,3)	(2,2)	
Conv2D	conv_2	(β, 27, 27, 256)	614656	256	(5,5)	(1,1)	*same*
MaxPooling2D	pool_2	(β, 13, 13, 256)	0		(3,3)	(2,2)	
Conv2D	conv_3	(β, 13, 13, 384)	885120	384	(3,3)	(1,1)	*same*
Conv2D	conv_4	(β, 13, 13, 384)	1327488	384	(3,3)	(1,1)	*same*
Conv2D	conv_5	(β, 13, 13, 256)	884992	256	(3,3)	(1,1)	*same*
MaxPooling2D	pool_5	(β, 6, 6, 256)	0		(3,3)	(2,2)	
Conv2DTranspose	Deconv_1	(β, 13, 13, 384)	885120	384	(4,3)	(2,2)	*valid*
Conv2D	Deconv_2	(β, 13, 13, 384)	1327488	384	(3,3)	(1,1)	*same*
Conv2D	Deconv_3	(β, 13, 13, 256)	884992	256	(3,3)	(1,1)	*same*
Conv2DTranspose	Deconv_4	(β, 27, 27, 96)	221280	96	(4,3)	(2,2)	*valid*
Conv2DTranspose	Deconv_5	(β, 640, 454, 3)	103971	3	(24,22)	(8,8)	*valid*

Total params: 7,170,051
Trainable params: 7,170,051
Non-trainable params: 0

Table 26: Architecture 3, inspired by AlexNet, here the input size has A4 ratio

Model: "AlexNet_2"							
Layer (type)	Name	Output Shape	Param #	k	f	s	p
InputLayer	Input	(β, 320, 227, 3)	0				
Conv2D	conv_1	(β, 78, 55, 96)	34944	96	(11,11)	(4,4)	*valid*
MaxPooling2D	pool_1	(β, 38, 27, 96)	0		(3,3)	(2,2)	
Conv2D	conv_2	(β, 38, 27, 256)	614656	256	(5,5)	(1,1)	*same*
MaxPooling2D	pool_2	(β, 18, 13, 256)	0		(3,3)	(2,2)	
Conv2D	conv_3	(β, 18, 13, 384)	885120	384	(3,3)	(1,1)	*same*
Conv2D	conv_4	(β, 18, 13, 384)	1327488	384	(3,3)	(1,1)	*same*
Conv2D	conv_5	(β, 18, 13, 256)	884992	256	(3,3)	(1,1)	*same*
MaxPooling2D	pool_5	(β, 8, 6, 256)	0		(3,3)	(2,2)	
Conv2DTranspose	Deconv_1	(β, 18, 13, 384)	1180032	384	(4,3)	(2,2)	*valid*
Conv2D	Deconv_2	(β, 18, 13, 384)	1327488	384	(3,3)	(1,1)	*same*
Conv2D	Deconv_3	(β, 18, 13, 256)	884992	256	(3,3)	(1,1)	*same*
Conv2DTranspose	Deconv_4	(β, 38, 27, 96)	295008	96	(4,3)	(2,2)	*valid*
Conv2DTranspose	Deconv_5	(β, 320, 227, 3)	131331	3	(24,22)	(8,8)	*valid*

Total params: 7,566,051
Trainable params: 7,566,051
Non-trainable params: 0

Table 27: Architecture 4, inspired by AlexNet, here the input size has an A4 ratio, and the input size is twice as big as the previous model.

Model: "AlexNet_3"							
Layer (type)	Name	Output Shape	Param #	k	f	s	p
InputLayer	Input	(β, 640, 454, 3)	0				
Conv2D	conv_1	(β, 158, 111, 96)	34944	96	(11,11)	(4,4)	*valid*
MaxPooling2D	pool_1	(β, 78, 55, 96)	0		(3,3)	(2,2)	
Conv2D	conv_2	(β, 78, 55, 256)	614656	256	(5,5)	(1,1)	*same*
MaxPooling2D	pool_2	(β, 38, 27, 256)	0		(3,3)	(2,2)	
Conv2D	conv_3	(β, 38, 27, 384)	885120	384	(3,3)	(1,1)	*same*
Conv2D	conv_4	(β, 38, 27, 384)	1327488	384	(3,3)	(1,1)	*same*
Conv2D	conv_5	(β, 38, 27, 256)	884992	256	(3,3)	(1,1)	*same*
MaxPooling2D	pool_5	(β, 18, 13, 256)	0		(3,3)	(2,2)	
Conv2DTranspose	Deconv_1	(β, 38, 27, 384)	1180032	384	(4,3)	(2,2)	*valid*
Conv2D	Deconv_2	(β, 38, 27, 384)	1327488	384	(3,3)	(1,1)	*same*
Conv2D	Deconv_3	(β, 38, 27, 256)	884992	256	(3,3)	(1,1)	*same*
Conv2DTranspose	Deconv_4	(β, 78, 55, 96)	295008	96	(4,3)	(2,2)	*valid*
Conv2DTranspose	Deconv_5	(β, 640, 454, 3)	152067	3	(24,22)	(8,8)	*valid*

Total params: 7,586,787
Trainable params: 7,586,787
Non-trainable params: 0

Table 28: Architecture 5, inspired by AlexNet, here the input size has an A4 ratio, and the input size is twice as big as the previous model. With atrous layers

Model: "AlexNet_4"								
Layer (type)	Name	Output Shape	Param #	k	$f \times f$	s	p	d
InputLayer	Input	(β, 640, 454, 3)	0					
Conv2D	conv_1	(β, 158, 111, 96)	34944	96	(11,11)	(4,4)	*valid*	(1,1)
MaxPooling2D	pool_1	(β, 78, 55, 96)	0		(3,3)	(2,2)		
Conv2D	conv_2	(β, 78, 55, 256)	614656	256	(5,5)	(1,1)	*same*	(1,1)
MaxPooling2D	pool_2	(β, 38, 27, 256)	0		(3,3)	(2,2)		
Conv2D	conv_3	(β, 38, 27, 384)	885120	384	(3,3)	(1,1)	*same*	(2,2)
Conv2D	conv_4	(β, 38, 27, 384)	1327488	384	(3,3)	(1,1)	*same*	(4,4)
Conv2D	conv_5	(β, 38, 27, 256)	884992	256	(3,3)	(1,1)	*same*	(8,8)
MaxPooling2D	pool_5	(β, 18, 13, 256)	0		(3,3)	(2,2)		
Conv2DTranspose	Deconv_1	(β, 38, 27, 384)	1180032	384	(4,3)	(2,2)	*valid*	(1,1)
Conv2D	Deconv_2	(β, 38, 27, 384)	1327488	384	(3,3)	(1,1)	*same*	(1,1)
Conv2D	Deconv_3	(β, 38, 27, 256)	884992	256	(3,3)	(1,1)	*same*	(1,1)
Conv2DTranspose	Deconv_4	(β, 78, 55, 96)	295008	96	(4,3)	(2,2)	*valid*	(1,1)
Conv2DTranspose	Deconv_5	(β, 640, 454, 3)	152067	3	(24,22)	(8,8)	*valid*	(1,1)
Total params: 7,586,787 Trainable params: 7,586,787 Non-trainable params: 0								

D Type maps

The type maps are made by making a (640×454) grid that represents the invoice in types. Meaning when there is a black word in pixel (6,6), the RGB representation would be (0,0,0), while the type map would hold the number of the type of the word. The type numbers are given in Table 29. As was mentioned before, types that are close, like price and currency symbol, which fall in the same category of describing a price, have close numbers. While other types that are similar in text but different in meaning, e.g., percentage and number, have more distant numbers.

Table 29: Type map numbers

Type	Number	Type	Number	Type	Number
Empty	0	Quantity	25	Word	35
Number	300	Currency symbol	200	Words	25
Date	100	E-mail address	170	Don't know	40
Percentage	50	Website	175	Description	15
BTW code	150	Price	205		
IBAN	155	Name	30		

As was mentioned in Section 5.3, some types are found by regular expression statements, Table 30 provides the exact regex statements that are used.

Table 30: Regular expression statements to find some of the types

Type	Regular expression
Date	Found by trying every day, month, and year option, alongside many different language and delimiter options
BTW code	[a-zA-Z]{2} ?.?[0-9]{4} ?.?[0-9]{2} ?.?[0-9]{3} ?.?B[0-9]{2}$
IBAN	[a-zA-Z]{2}[0-9]{2} ?.?[a-zA-Z0-9]{4} ?.?[0-9]{4} ?.?[0-9]{0,4} ?.? [0-9]{4} ?.?[0-9]{2}$
Currency symbol	[$¢£\u20a0-\u20bd\ua838\ufdfc\ufe69\uff04\uffe0\uffe1\uffe5\uffe6]
E-mail address	[a-zA-Z0-9_.+-]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-.]+
Website	www.+[a-zA-Z0-9_.+-]+\.[a-zA-Z0-9-.]+

Price (del= '.')	€{1}[0-9.]+.?[0-9]{0,2}
	[0-9,]+.?[0-9]{0,2}€{1}

Price (del= ',')	€{1}[0-9,]+,?[0-9]{0,2}
	[0-9.],?[0-9]{0,2}€{1}

Postal code (extra)	[0-9]{4} ?[A-Za-z]{2} [A-Z]{1}[A-Za-z]+

E Full RCNN model

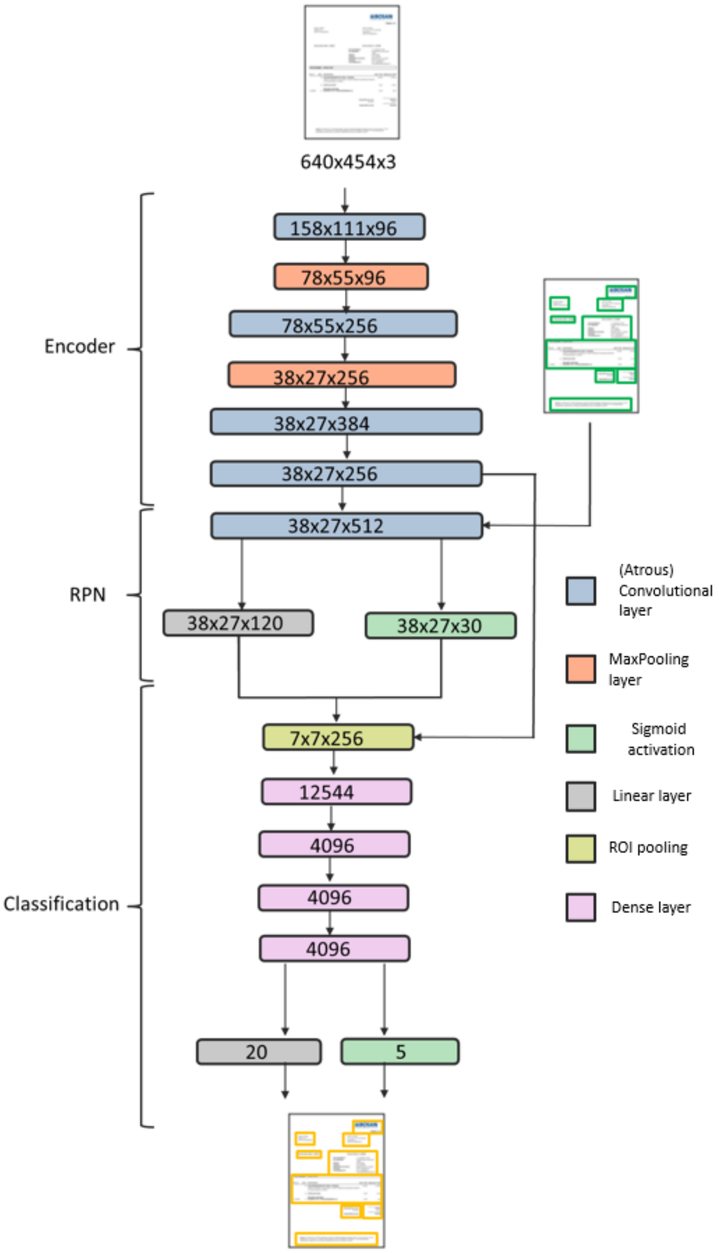


Figure 48: Full RCNN model, with all separate modelling parts. This would also be the Faster-RCNN adaptation and has also been implemented, but never tested. The input invoice is the image without boxes on it, the annotated image has green boxes, and the output of the model is shown with orange (proposed) regions.

F Results per algorithm

Table 31: Specific results of the different classification techniques

Score	Model	Min	Median	Max	Mean
Accuracy	Rule-Based	0.835	0.972	1.000	0.964
	LightGBM	0.821	0.978	1.000	0.970
	Regions	0.919	0.982	1.000	0.981
F1	Rule-Based	0.089	0.415	1.000	0.415
	LightGBM	0.068	0.221	1.000	0.288
	Regions	0.088	0.120	1.000	0.357
Adjusted F1	Rule-Based	0.000	0.430	1.000	0.441
	LightGBM	0.096	0.108	1.000	0.181
	Regions	0.000	0.000	1.000	0.138