Østfold University College

Advanced Image Recognition and AI Training

Text extraction with machine learning

Bachelor Thesis

Joakim S. Aarskog, Sondre Lillelien, Sander Riis

BO21-G27
Department of Information Technology
Østfold University College
Halden, Norway
18th May 2021

Contents

List of Figures

Chapter 1

Introduction

A lot of companies have employees that travel in their work. This travel is usually funded by their employer. In order to keep track of all the transactions generated by this travel, the company will store the receipts from the transactions. By saving the receipts in this manner, it greatly reduces the effort needed from employers to refund their employees for their purchases.

Going through these transactions manually is time-consuming and labor intensive. Because of this, creating an automated solution for extracting the data contained in these receipts, will effectively reduce the amount of manpower and manhours required by the company. The goal of this thesis is to explore and discuss these automated solutions, and present a possible solution.

1.1 Background and motivation

1.1.1 Project Group

Joakim Aarskog is a 24-year-old computer science student at Østfold University College. He has an interest in technology and photography.

Sondre Lillelien is a 26-year-old computer engineering student at Østfold University College. Interests include technology, engineering and AI.

Sander Riis is a 23 years old computer science student at Østfold University College. He likes to develop programs and learn about new technology.

1.1.2 Employer

The employer for this thesis is Simployer AS, a tech company that specializes in Human Resource Management (HRM). They deliver HRM software to 15.000 customers and have over 1.2 million users in Norway and Sweden. Simployer is passionate about giving other employers and their employees the tools to be able to effectively organize themselves. Simployer is a sizable company with over 250 employees, and a history going back 35 years. Our main contact in Simployer is Flemming Ottosen, their CTO.

1.1.3 Problem statement

As Simployer delivers HRM software to many Scandinavian companies, they have a large database containing receipts generated by their employees. These documents are stored in an unorganized manner, making manual validation of these documents difficult and time-consuming. The documents are typically in the form of an image of a receipt taken with a mobile camera. The documents lack metadata, which makes searching through them to find a specific document difficult. Simployer would like a way to organize this data automatically and attach relevant metadata to the documents. This metadata includes the price of the transaction, the company the service was purchased from, and the date of the transaction. In addition to attaching metadata to existing documents, Simployer would like to automatically generate and attach metadata to new documents coming in to their database.

1.1.4 Motivation

Artificial intelligence and machine learning is a rapidly growing field in computer science. The biggest obstacles that AI and ML faced in the past, was a lack of computing power and a lack of data. In today's world, both of these obstacles have largely been overcome as computing power keeps growing and data is a more abundant resource than before. The motivation for this thesis is to take advantage of this abundance of computing power and data, to solve real life problems.

1.1.5 Objectives

The objectives for this thesis are:

Objective 1 To create a machine learning model, that can identify desired data in an image of a receipt.

Objective 2 To have the machine learning model be capable of improving it's performance when new data is acquired.

Objective 3 To develop an API that can deliver data to the machine learning model and fetch the results.

Objective 4 To research several ways of solving the problem in order to present alternatives to the thesis employer.

1.1.6 Method

The aim of the thesis has pivoted somewhat since the start of the semester. Originally the goal was to create a machine learning model that can correctly identify desired data from a large variety of image types with different quality. The goal is still to create such a model, but more focus will be directed towards the learning module. Therefore, we have de-prioritized being able to work with a large variety of images.

The methods used for this thesis are the following:

- Deep learning literature research
- Text analysis and text extraction articles
- Clearly defining the scope and objectives of the thesis

- .NET Core articles
- Workflow management in Jira

1.1.7 Deliverables

The following items should be delivered by the thesis' end-date:

- A report detailing the entire thesis
- A machine learning model that can extract desired data from the images it is given
- An API that can deliver images to the ML pipeline and request the results of data extraction done on the images

1.2 Report outline

Chapter 2 explains the thesis' topic and scope in detail. In addition; the methods used to complete the work, along with literature the work is based on will be included here.

Chapter 3 provides a detailed walkthrough of how the thesis is designed. The design is made to efficiently and correctly tackle the problem statement.

Chapter 4 describes the implementation of the thesis. This includes how the thesis group chose to implement the design discussed in previous chapters, and the research methods used to arrive at design decisions.

Chapter 5 presents the results the thesis group's work in an objective manner.

Chapter 6 discusses the results discovered in chapter 5. This includes how the results differ from what was expected, how the findings might be relevant to the thesis' field of study, etc.

Chapter 7 concludes and summarizes the thesis in a manner that can be read by someone who did not read the entire report.

Chapter 2

Analysis

2.1 Research topic

Our project is to create a solution that allows a user to take an image of a receipt, and automatically upload the relevant data to a database. This automates the process of handling travel receipts, thus making it less time-consuming and reduces the cost. To achieve this, we will be using Optical Character Recognition software and machine learning. The extracted data from the image consists of date, total amount, and the receipt type.

The app will consist of two main parts: The OCR, and the model that reads the data. We are going to use a free open source OCR, named Tesseracts. The OCR is what will allow us to extract text from the images. This OCR will send all the text that is on the receipt in the form of an array to the model. The model will then decide what information is the correct data to extract and keep.

With the app we also need to include an API that will talk with the app and talk with Simployers server. This API gets the data from the model and will prompt the information that is selected for the user for a final validation, before sending it to the Simployer database. This will allow Simployer to easily search in the database based on the metadata that was extracted.

Another objective of this thesis is to analyse and compare solutions that already exists on the market. And then create a prototype of each solution and connect them to the API.

2.2 Tools

This section details the different tools that have been considered for use in the thesis.

2.2.1 OCR engines

There are many OCR engines that could be used for this thesis. Three main ones have been selected for consideration. These are Google Cloud Vision API, Amazon Textract, and Tesseract.

Google Cloud Vision API

Google Cloud Vision API contains a well optimised OCR and good variety of detection. The detection features are listed in figure 2.1.

Feature type	
CROP_HINTS	Determine suggested vertices for a crop region on an image.
DOCUMENT_TEXT_DETECTION	Perform OCR on dense text images, such as documents (PDF/TIFF), and images with handwriting. TEXT_DETECTION can be used for sparse text images. Takes precedence when both DOCUMENT_TEXT_DETECTION and TEXT_DETECTION are present.
FACE_DETECTION	Detect faces within the image.
IMAGE_PROPERTIES	Compute a set of image properties, such as the image's dominant colors.
LABEL_DETECTION	Add labels based on image content.
LANDMARK_DETECTION	Detect geographic landmarks within the image.
LOGO_DETECTION	Detect company logos within the image.
OBJECT_LOCALIZATION	Detect and extract multiple objects in an image.
SAFE_SEARCH_DETECTION	Run SafeSearch to detect potentially unsafe or undesirable content.
TEXT_DETECTION	Perform Optical Character Recognition (OCR) on text within the image. Text detection is optimized for areas of sparse text within a larger image. If the image is a document (PDF/TIFF), has dense text, or contains handwriting, use DOCUMENT_TEXT_DETECTION instead.
WEB_DETECTION	Detect topical entities such as news, events, or celebrities within the image, and find similar images on the web using the power of Google Image Search.

Figure 2.1: Overview of features (list from https://cloud.google.com/vision/pricing)

We can see that Google API offers everything from text recognition, to landmark location and crop Hints. The Google API is know to be industry leading in the field, so this comes as no surprise. This API will have almost everything you need to develop an application with text recognition.

The API also has a lot of features in their beta version which is not included in figure 2.1.

Feature	Price per 1000 units		
	First 1000 units/month	Units 1001 - 5,000,000 / month	Units 5,000,001 and higher / month
Label Detection	Free	$1.50	$1.00
Text Detection	Free	$1.50	$0.60
Document Text Detection	Free	$1.50	$0.60
Safe Search (explicit content) Detection	Free	Free with Label Detection, or $1.50	Free with Label Detection, or $0.60
Facial Detection	Free	$1.50	$0.60
Facial Detection - Celebrity Recognition	Free	$1.50	$0.60
Landmark Detection	Free	$1.50	$0.60
Logo Detection	Free	$1.50	$0.60
Image Properties	Free	$1.50	$0.60
Crop Hints	Free	Free with Image Properties, or $1.50	Free with Image Properties, or $0.60
Web Detection	Free	$3.50	Contact Google for more information
Object Localization	Free	$2.25	$1.50

Figure 2.2: Overview of prices (list from https://cloud.google.com/vision/pricing)

Google is the best in the field, but they are also the most expensive API to use. The prices are calculated for every 1000th request This means that you are getting the first 1000-requests free and then you will need to play 1.5$ for each 1000-request you do on for example text detection. If you have a request size of 50.000-requests this will be 49*1.5$ = 73,5 dollars (620 NOK). This will reset every month.

Amazon Textract

Amazon's AWS hosts a service called Textract. This cloud-based service lets you upload an image to their text recognition software.

Detect Document Text API This feature is the classic OCR feature that will extract text from a document. Amazon will charge you 0.0015$ per page (1.50$ per 1000) for the first million pages, when you exceed 1 million the price wil go down to 0.0006$ per page (0.6$ per 1000)

Table Extraction This is an improved version of Detect Document Text API. This will group the text in tables. This is helpful with structured data, such as financial reports. Amazon will charge you 0.015$ per page (15.0$ per 1000) for the first million pages, when you exceed 1 million the price wil go down to 0.01$ per page (10$ per 1000)

Form Extraction This detects key-value pairs in documents, for example if your document has a field named "firstName" it will pair it with "Mike". This way "first name" would be the key and "Mike" would be the value. This makes it easy to either sort it into a database or reuse the values in variables. With normal OCR the relation is lost. Amazon

will charge you 0.05$ per page (50$ per 1000) for the first million pages, when you exceed 1 million the price wil go down to 0.04$ per page (40$ per 1000)

Analyze Document API

This is a combination of the tables and forms extraction Amazon will charge you 0.065$ per page (65$ per 1000) for the first million pages, when you exceed 1 million the price wil go down to 0.05$ per page (0.50$ per 1000)

Note that Amazon changes their pricing on the region you are in (you might want to check the price for your region), they also do not include a free 1000 scans per a month.

Tesseract

Tesseract is an open-source OCR engine that supports over 100 different languages. It has the reputation of being one of the best open-source OCR engines. It was created by HP in the 80's and later made open-source in 2005 and funded by google since 2006. Most other OCR APIs today are build on top of this, like Google Cloud Vision API.

If this engine is used, code that takes advantage of the engine's built-in methods must be written. When dealing with high-quality images and computer generated pdf receipts, writing code to extract the text is relatively simple. When the images are of lower quality, more care must be given to pre-processing to improve the accuracy of the text extraction. This is because OCR software needs good contrast and little to no noise in the images. Another problem is curved text, like in a book. This might be the hardest ting to solve.

2.3 Preprocessing

Having images of ideal quality for use in an OCR is a luxury. Images will often be of low quality or taken from suboptimal angles. In order to mitigate the effects that low quality images will have on the text extraction accuracy, the images should go through a preprocessing stage before getting sent to the OCR software.

2.3.1 Scaling the image

OCR software will have the best performance when given images between 300dpi and 600dpi. Anything less will make it unreadable for the OCR and anything more will consume extra processing power with little to no improvement in accuracy.

2.3.2 Skew correction

OCR engines use line segmentation to separate the text found in an image into different lines of data. Therefore, it is important that the OCR is given images that are as straight as possible. Skew correction is the part of the pre-processing pipeline that addresses this issue.

2.3.3 Remove background noise

Image noise can reduce the accuracy of the OCR engine. Because of this, removing noise in the background is very important to improve readability. Applying a Gaussian blur is one way of reducing the noise in the image.

2.3.4 Binarization

OCR engines are usually designed to handle input images with a black-and-white color scheme. The process of converting a colorized or grayscale image to black-and-white is called binarization. Once an image has gone through binarization, the background should be white and any foreground elements like text should be black.

If your image is in RGB format, it will first need to be converted to grayscale format before thresholding techniques can be used. Adaptive thresholding is one such technique. It involves partitioning the image into smaller regions, and calculating the threshold for what should be black and what should be white, in that region.

2.4 Neural Networks

Artificial Neural Networks, or just Neural Networks as they are more commonly called, is a machine learning technique that learns by replicating the functions of the brain. Our brain has billions of neurons connected through synapses in a massive network. These neurons receive electrical signals from other neurons, and may send out their own electrical signal if their received signal was strong enough. If the paths these signals take are used often, they will strengthen, allowing for a greater portion of electricity to be transfered. This strengthening of pathways is how we as humans learn, and it is what we try to replicate with an Artifical Neural Network.

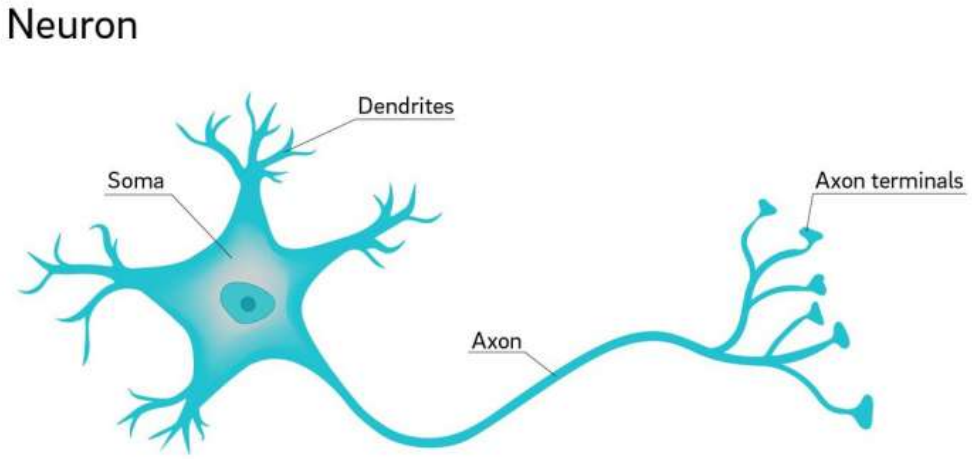

Figure 2.3: Illustration of a neuron. https://scx1.b-cdn.net/csz/news/800a/2018/2-whyareneuron.jpg

The digital version of our neurons functions much in the same way. They take inputs as numbers instead of electrical signals, and based on the strength of the connections the numbers arrived through, decide whether to fire off their own signal. This process is illustrated in figure 2.4. The input x_n is the n-th signal being passed to the neuron. The

weight w_n is the strength of the connection between the last neuron and this neuron. All the inputs are multiplied with their corresponding weights, and summed up in the neuron, along with a bias for that neuron. The activation function is what decides whether that neuron should fire of their signal to the next neuron or not. In mathematical terms, the whole process looks like this:

$$y = f(\left(\sum_{i=1}^{n} x_i w_i\right) + bias)$$

Adjusting the weights to make the neuron fire when it is supposed to it what makes the network learn.

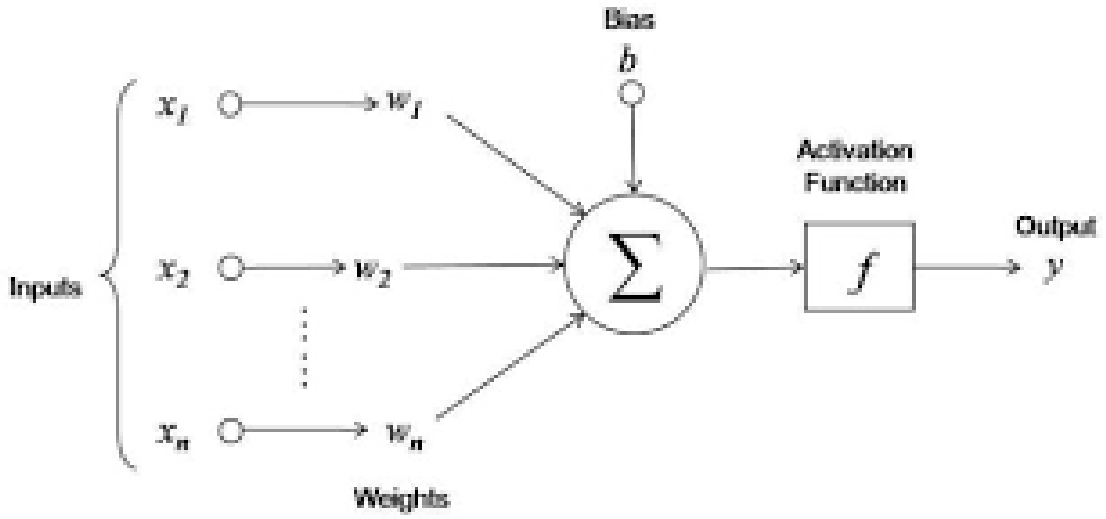

Figure 2.4: Abstraction of a neuron for use in Artifical Neural Networks. https://encrypted-tbn0.gstatic.com/images?q=tbn:ANd9GcTRTFh3jYAWBE9gH0bKPJXykku4WDgqhJvQeQusqp=CAU

2.5 Spacy

Spacy is a program that gives context and relations to sentences and words. With a foundation of machine learning. This becomes a great tool for data extraction. Spacy includes some great features that can make the production quick and easy.

2.5.1 Arcitecture

Spacy is structured around three main parts. The language class, the vocab and Doc objects. The language class is mainly focused on turning the data into doc objects. Docs contains a sequence of tokens and their annotations and the vocab contains word vectors and lexical attributes. With these spacy can centralize the strings and save memory(FOTNOTE: https://spacy.io/usage/spacy-101). The product of these three central parts of the architecture and other sub classes will provide the architecture that is shown under. (KAN MAN POENGTERE TIL BILDER PÅ DENNE MÅTEN?)

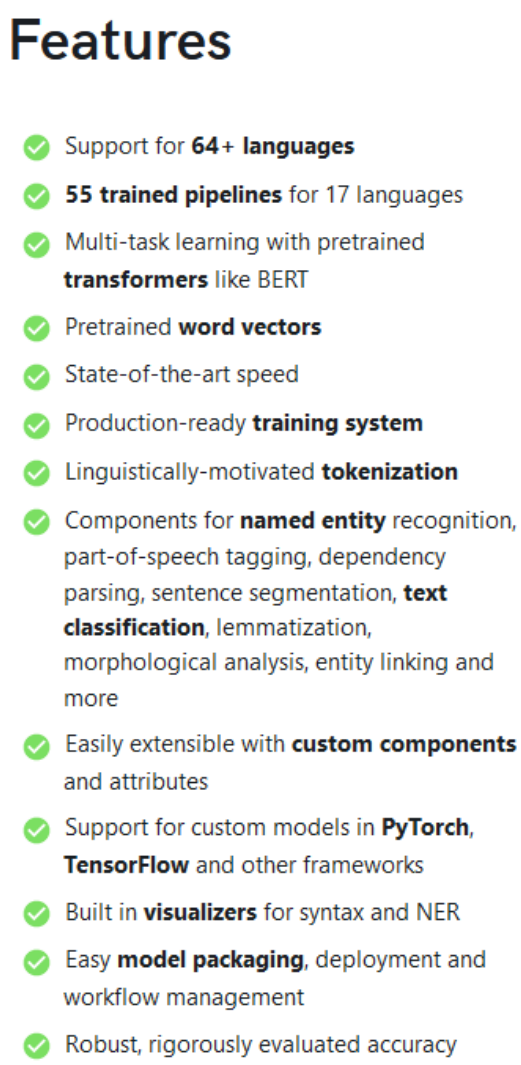

Figure 2.5: Overview of spacy features. https://spacy.io/usage/spacy-101

Spacy got a lot to offer, but we will focus on Named Entities. Which is an object that is assigned a name. Spacy does this by asking the model for a prediction. However, the model is reliant on the data sit was trained on, this means that for a costume application. Dedicated training could be necessary. If we take a look at an example from their web page. They got a simple string: Apple is looking at buying U.K. startup for $1 billion This will result with these labels

As you can see, you will get labels that describes the words. This combined with spacy's functionality to add relation to the words makes it easy to extract words or values you want. When you got the label of the words, you can ask for a relation overview. This will give you all the relations for your text. Then you can write a simple script that loops through all the words and select the once labeled as ORG. Uou can then extract all the substrings that is labeled as money. Then you would get out all the sentences that has and organisation talking about money in some form.

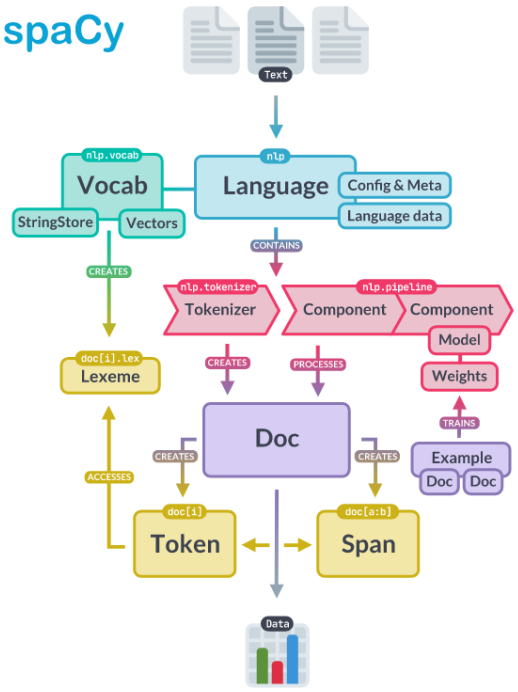

Figure 2.6: Overview of spacy architecture. https://spacy.io/usage/spacy-101

TEXT	START	END	LABEL	DESCRIPTION
Apple	0	5	ORG	Companies, agencies, institutions.
U.K.	27	31	GPE	Geopolitical entity, i.e. countries, cities, states.
$1 billion	44	54	MONEY	Monetary values, including unit.

Figure 2.7: Overview of spacy architecture. https://spacy.io/usage/spacy-101

Chapter 3

Design

The pipeline is designed to be able to take an input image of a receipt sent over http. It is going to provide metadata to that image based on its contents. This metadata includes the company name that issued the receipt, the date the receipt was made, and the price of the purchase. In order to transfer the image to our machine learning models, a .NET API is used.

The pipeline consists of a website,.NET API, database, ML API, CNN, OCR and Spacy.

The webpage will take an image input from the user. The webpage is connected with the .NET API and database. From here the image is sent to what we call is a machine learning API. This is written in java, we use it because we can't communicate directly to our python program with the .NET API. The ML API is talking to a flask API to convert the information to python and from here the image is sent to the CNN. The CNN does a prediction on the category of the image. Which is the company, this information is also returned to the API. Since all the different companies got different receipts, we figured that this was the easiest way to do it. For extracting the metadata for the companies we had one other solution that was heavily considered. That was to not use a CNN, but rather have an RNN do all the predictions. This would force us to change the alignment functionality of our pipeline. The prediction will be sent to the OCR from the CNN and the image itself will already have been sent to the OCR from the API. The prediction from the CNN will be used in the OCR to choose the right template for the image. The template is an aligned image that we have pre-loaded into the OCR model. This means that the OCR is holding one image of all the categorise we have and using this as a reference to skew the images horizontally (We need to do this because an OCR can't read skewed text). The skewing method is using something called matching, this basically finds similarities in the images and aligns the images based on these similarities. (You can read more about matching under implementation 4.5.1) We have this feature because our sample size is way too small. This forced us to create "fake" data by skewing images. This is a feature that is great to have, but in reality, not needed. This is because it is way better solutions for this. We would add a check for the image before accepting it. This would assure images that wasn't easily read by the OCR was discarded (the user would have to take another picture).

The aligned image is sendt to the second part of our OCR, where we do some simple thresholding and extracting the text from the image. The OCR sends the extracted text as a string to the sapcy model and the date to the API. The Spacy model reads the string,

adds relations to the text and evaluates the what's the total price. The date could also be evaluated in this model. (With research we have gotten the impression that its more common practise to find the date witn OCR).

Finally the data from CNN, OCR and Spacy is provided back to the user. This pipeline is illustrated in figure 3.1

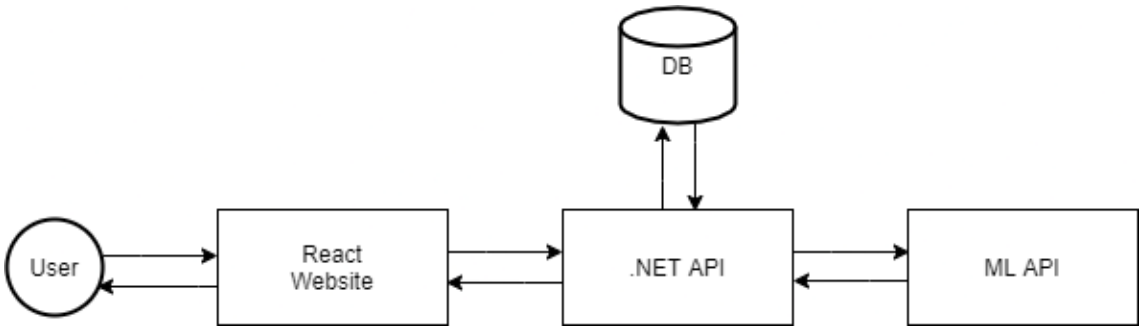

Figure 3.1: Graphic illustrating the project pipeline

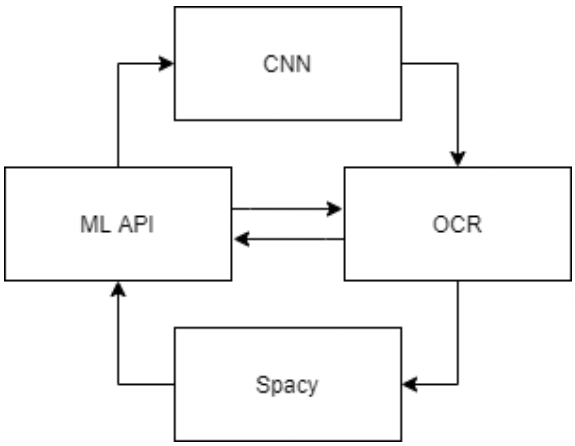

Figure 3.2: ML API pipeline

3.1 React website

The website is the first point in the pipeline and gives the user a way to interact with the pipeline. The website allows the user to upload the picture which is sent through the pipeline, and consecutively the receipt (figure 3.2).

3.2 .NET API

The .NET API exposes several endpoints that allows for sending an image to the ML API, and save an image and a receipt to a database.

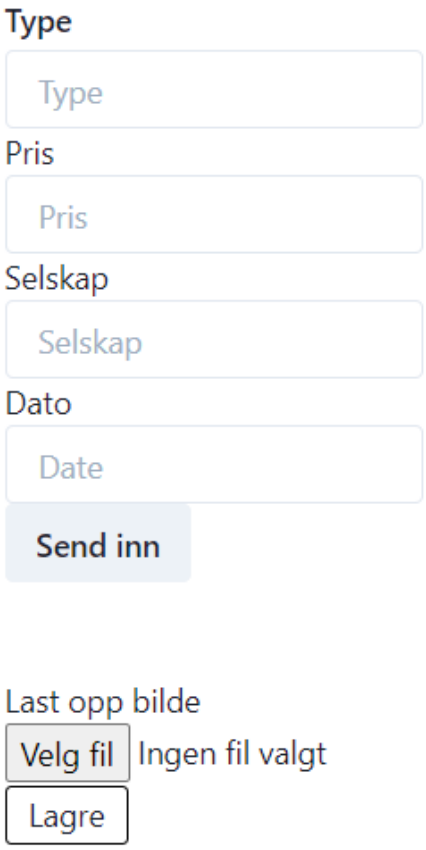

Figure 3.3: React website

3.3 ML API

The ML API works as an endpoint for the machine learning model, it utilizes the CNN, OCR and Spacy to get data from the image the user sends in, and then sends the data back to the .NET API. The pipeline is explained in figure 3.3

3.4 Dataset

In order to train our neural network models, Simployer provided us with a sample of images from their database. This dataset includes 1194 images of receipts taken with mobile cameras and generated pdfs from online sales.

This dataset is not labeled, which means we have to manually label the data we are going to use to train our networks. Because of the large variance of different types of companies in this dataset, creating labels for all of them would be very time-consuming. Instead of labeling them all, we decided to restrict ourselves to the three most common receipts found in the dataset. This means the amount of images we can use for training is greatly reduced, so in order to combat this, we used data augmentation software to generate more images. This is discussed in greater detail in chapter 4.

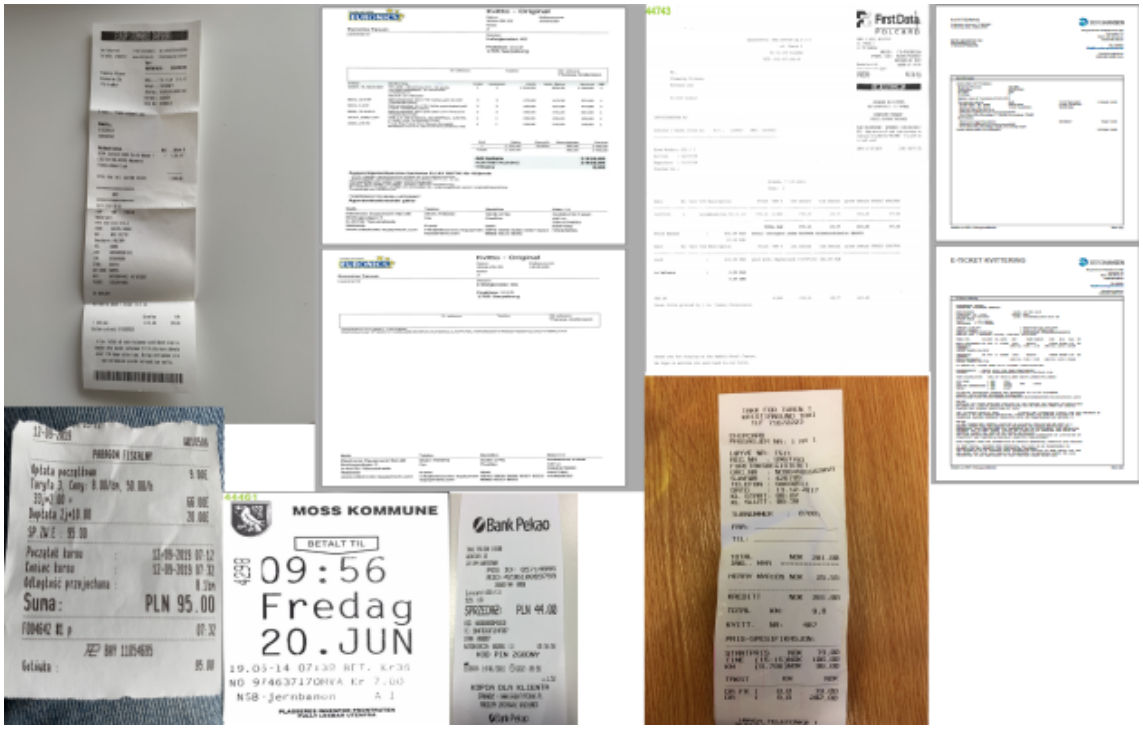

Figure 3.4: Examples of the images found in the dataset provided by Simployer

3.5 Pre-processing

A CNN does not require images of large resolution in order to accurately classify an image. In addition, training the network with large resolution images take significantly longer. Because of this, all the images that are going to be fed into the CNN are downscaled. This also solves the problem of images being of different sizes, as the CNN requires all the images to have the same size.

While the text in the image is now completely unreadable, the CNN will have no issues telling images in this format apart from each other.

3.6 Convolutional Neural network

As stated previously, we are using a CNN as an image classifier in order to determine which company issued the receipt.

Figure 3.3 illustrates the layout of our CNN. The input layer takes the pixel value of the image. Because of this, the number of nodes in the input layer has to be equal to the amount of pixels in the image. The output layer has three nodes, one for each type of receipt the network is trained to classify.

We are using a supervised learning-model to train our network. Because of this, the training data has to be labeled. This labeling can be labor intensive if the dataset you are labeling is large. Since we started with a very small amount of images before the use of data augmentation, it is a small task.

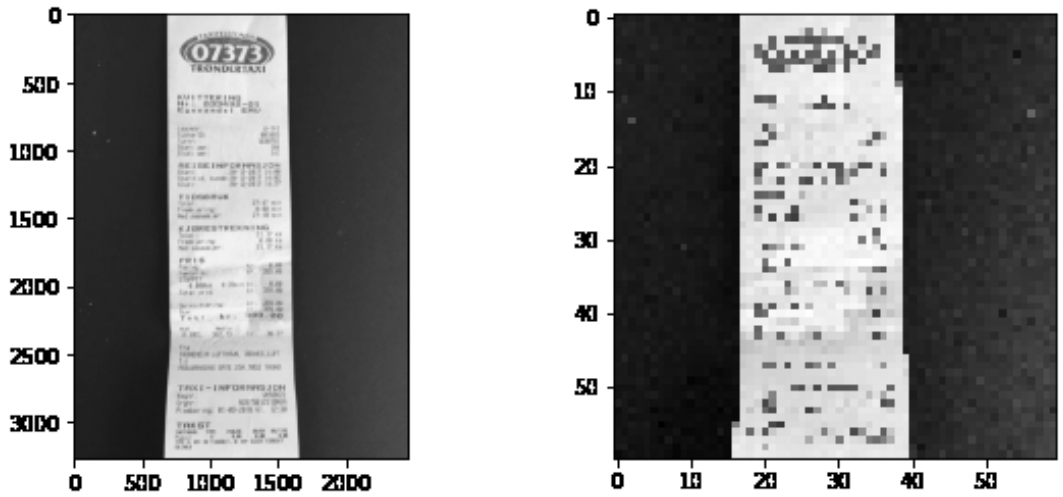

Figure 3.5: Before and after image rescaling

3.7 Optical Character Recognition

In order to extract the text found in an image, we are using the open-source OCR software Python-tesseract. Python-tesseract, or Pytesseract for short, utilizes Google's Tesseract-OCR engine.

When metadata for a particular image is requested by a user, the API will provide our implementation of Pytesseract with an image. Pytesseract will take this image as an input, and can provide outputs in various formats like xml or pdf. We will be using the output in string format, as this will serve as the input to our RNN model. Once Pytesseract has completed the text extraction, the output string is then fetched by our API to be provided to our RNN model.

3.8 Recurrent Neural Network

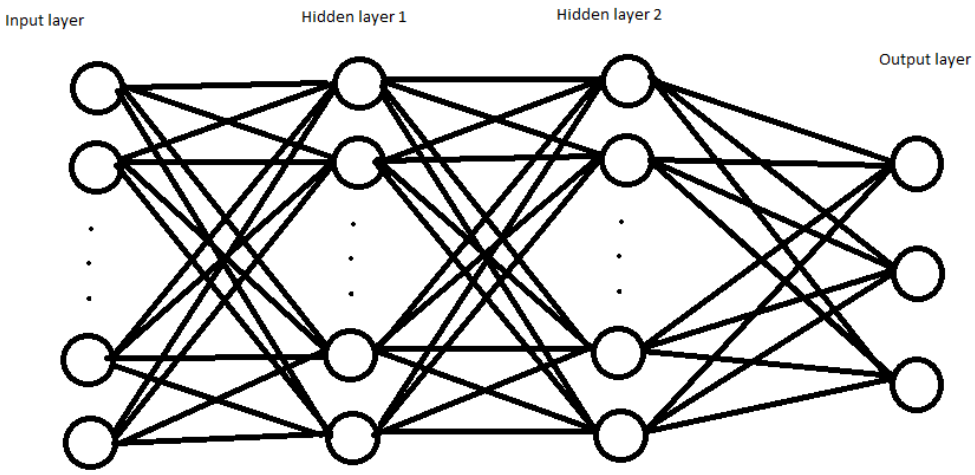

Figure 3.6: Graphic illustrating the Convolutional Neural Network (PLACEHOLDER)

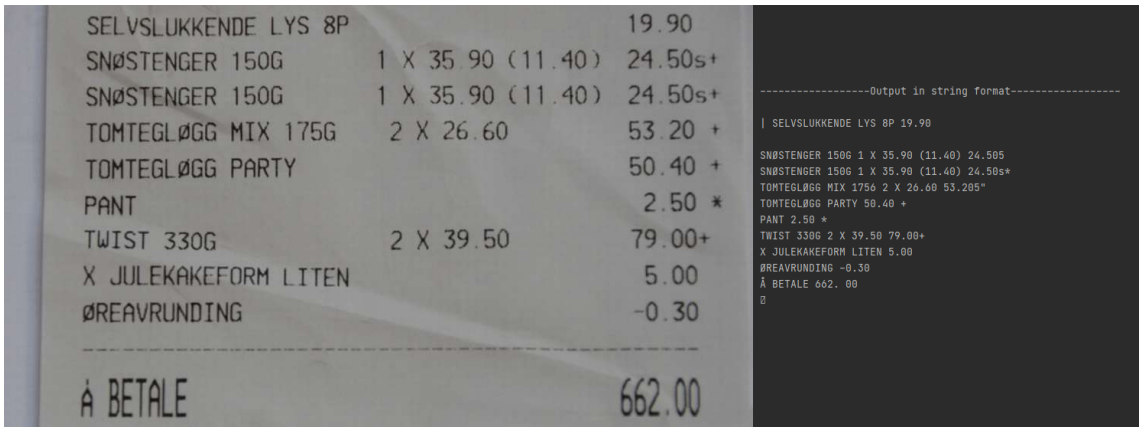

Figure 3.7: Example output of pytesseract with a test image

Chapter 4

Implementation

Chapter 4 describes the implementation of the thesis. This includes how the thesis group chose to implement the design discussed in previous chapters, and the research methods used to arrive at design decisions.

4.1 REST API

The API is a REST API coded in C# with .NET. REST stands for REpresentational State Transfer and is a software architecture style. For the API to be considered a REST API it has to follow some set guidelines and principles.

The API consists of 4 different modules: The API endpoints, DataAccess, DataAccess.Maintenance and Model.

API endpoints contains the controllers which handles the routing and requests. There are three controllers, one for receipt, one for image and one for connecting to the machine learning API. They receipt and image contain the methods GET, POST, PUT, and DELETE. While the machine learning only contains on for posting a picture.

In ImagesController the POST method uses the ImageDTO object. It initializes a new Image object, and populates the corresponding data. To set the image data, it serializes the image data from the DTO, and uses this data to for the image data.

DataAccess contains the DataContext for the database. For the sake of simplicity, we have decided to utilize a local database.

DataAccess.Maintenance contains the migrations for the project. This is to make it cleaner and to have a console app where it is possible to update the database without having to run the API.

The **Model** module contains the different models we use. There is one for receipts and one for image, there is also a data transfer object for image. The DTO is an object that carries data between processes and makes it more efficient by doing it in only one call. This is done by aggregating the data instead of sending it one by one.

4.2 ML API

The ML API is created using Flask. This has one POST route that takes in an base64 string and converts it to an image and sends it to CNN and OCR for processing. The data is then returned to the .NET API

4.3 Web solution

The web solution is made with React in JavaScript. The purpose of the web solution is to give the user an interface where they can upload an image of a receipt. And get the corresponding data from the image, and then send in the receipt with the data from the image. This is to make the process of entering the data less time consuming.

When the user uploads an image, the web solution sends a POST request to the API, then the API saves the image to the database. When the image is saved, the image is sent to the OCR for text recognition. If the output makes sense the data is returned and displayed to the user. Then if the returned data is correct the user can then upload the receipt to the database.

4.4 Dataset

As mentioned previously, dataset provided by Simployer consists of 1194 unlabeled images and PDF's. However, the variance of the types of receipts in the dataset is very large. When attempting to label the data with the corresponding company name, the amount of different companies were very large. Because of this, we end up with a very small amount of images of each type after labeling them. We decided to use the 3 most frequent companies, with the third most frequent company occurring only 19 times. At the end of this process, the amount of usable data was very small. The original dataset of 1194 images has been filtered down to 91 images, split into three different types of receipts.

Figure 4.1: Examples of labeled images

4.5 Pre-processing

4.5.1 Aligning

The first thing we make sure to do in our OCR is aligning the images. This is needed because an OCR can't reed skewed images. This means that we need to horizontal align the text as well as possible.

Matching

Matching is a good way of having a dynamic aligning algorithm. We have template images for all the different receipts types (in this case 3 images). What the matching algorithm does is to look through the receipt you feed it and see if it has any "mathces" with the template one. It then calculates how much it needs to skew the image in a given direction, to make it aligned with the template version.

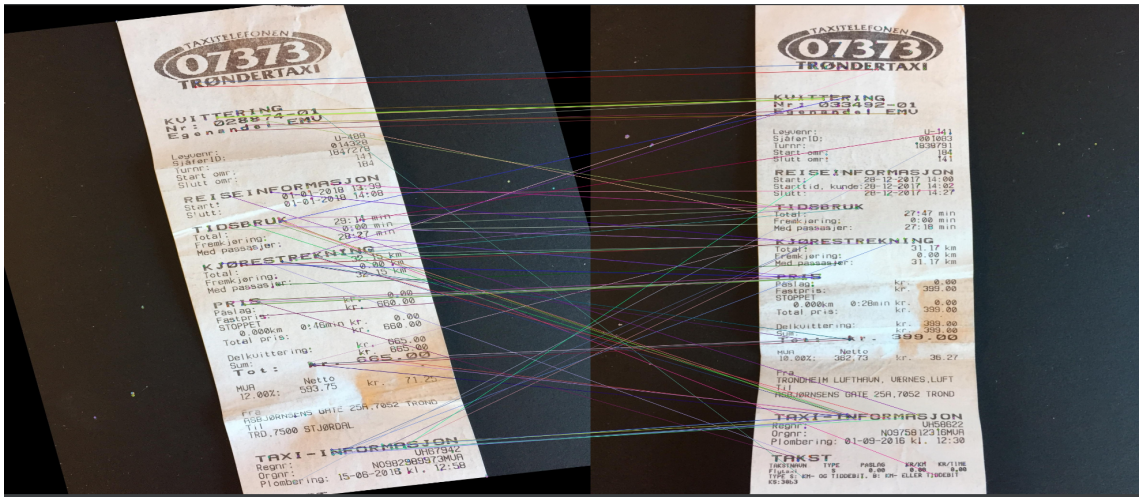

Figure 4.2: Finding matches between a given image to a template

In the image above you can see the algorithm finding matches between a Trøndertaxi receipt from two different years. Yes, these recipes is almost identical. Making this functionality dynamic for almost all receipts should be possible, (no evidence for this) by having a pool of template receipts. Then you can have a CNN check which receipts are the most similar and use the highest scoring receipt as the template image for any given receipt.

4.5.2 Data augmentation

Because of the low amount of images in the filtered dataset, data augmentation has to be done in order to give us enough images to be able to effectively train the CNN. We used an open source data augmentation tool for images for this task LINK. The augmentor takes a directory of images and applies a series of transformations on the images like scaling, skewing and rotating. This creates copies of the images with slight differences, increasing the size of our dataset. After applying enough transformations, the size of the dataset grew from 91 images to 1500.

4.5.3 Image preparation

Many of the images in the dataset are taken in different resolutions. In order to prepare the images to be passed forward to the CNN model, the images resolution is rescaled to 60x60 pixels. This significantly reduces the training time required by the network with minimal to no loss in accuracy. The images are also converted to grayscale instead of RGB.

Figure 4.3: Original image on the left, along with generated images from blur and rotation transformations

This reduces the amount input nodes required by the network, as each pixel will have one value instead of three.

4.6 Convolutional Neural Network

Our CNN is implemented using Keras. Keras is a library for tensorflow that allows for the creation of neural network models with only a few lines of code. The model consist of an input layer, two hidden layers, and an output layer. The input layer and first hidden layer are 2D convolutional layers with 32 and 64 filters. The second hidden layer, and the output layer are dense layers, where the hidden layer has 64 filters and the output layer has 3 filters. The activation function of the first three layers is Rectified Linear Unit (ReLU), while the output layer uses the sigmoid function. Maxpooling2D is used to define a stride of 2. This means that we take the biggest number with something that is called a max operation. We combine the results to get a 2x2 max pool output, down from a 4x4 format. As illustrated below.

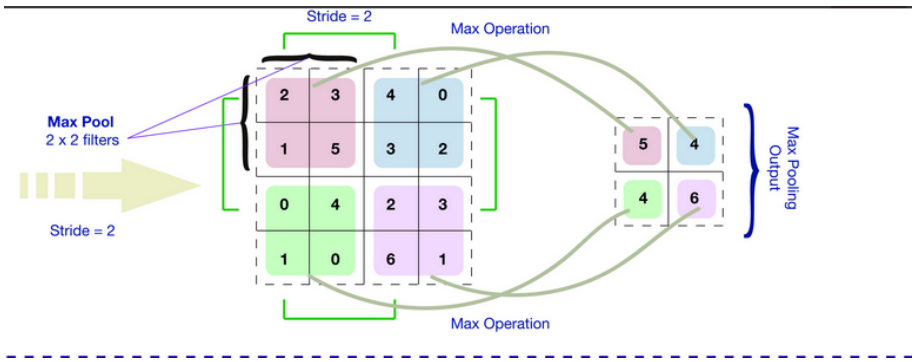

Figure 4.4: Overview for max pooling (Image from https://www.quora.com/What-is-max-pooling-in-convolutional-neural-networks)

We have also used batch normalisation with the default settings in the input and hidden layers of out model. This means that the model is sharding the data in batches. This in turn lets us train the model with a lot less training steps, and it gives us even better accuracy. The last function we will talk about in the input and hidden layers are the dropout function. This function will force the model to drop random filters during the training. This will force an adaption in the model that makes it more versatile. This is because the model can't just rely on one good path to carry its accuracy. Therefore, it will end up with a more generic understanding of the data and avoid an overfitting.

```
opt = tf.keras.optimizers.Adam()
model = Sequential()
model.add(Conv2D(32, (3,3), input_shape = X.shape[1:]))
model.add(Activation("relu"))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(BatchNormalization())
model.add(Dropout(0.2))

model.add(Conv2D(64, (3,3)))
model.add(Activation("relu"))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(BatchNormalization())
model.add(Dropout(0.2))

model.add(Flatten())
model.add(Dense(128, activation="relu"))
model.add(Dropout(0.2))

model.add(Dense(3, activation="sigmoid"))
```

Figure 4.5: CNN model

In the compile function we defined we choose to use cross-entropy as our loss function. Because this is the best for multiclass classification. The optimizer is the standard adam optimizer.

```
model.compile(loss='categorical_crossentropy',
              optimizer=opt,
              metrics=["acc"])
```

Figure 4.6: CNN model compile

4.7 OCR

The first thing we did in the OCR was

```
pytesseract.pytesseract.tesseract_cmd = 'C:\\Program Files\\Tesseract-OCR\\tesseract.exe'
img = cv2.imread('vy.jpg')
#img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
grayImg = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
adaptive_thresholdImg = cv2.adaptiveThreshold(grayImg, 255, cv2.ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY, 77, 14)
```

Figure 4.7: CNN model compile

```
NOR = 'nor'
config = "--psm 4"
hImg, wImg, _ = img.shape
boxes = pytesseract.image_to_data(adaptive_thresholdImg, lang=NOR, config=config)
```

Figure 4.8: CNN model compile

```
# #[  0       1         2         3         4         5       6      7      8       9       10     11 ]
# #['level', 'page_num', 'block_num', 'par_num', 'line_num', 'word_num', 'left', 'top', 'width', 'height', 'conf', 'text']
```

Figure 4.9: CNN model compile

```
detectedWords = []
for x, b in enumerate(boxes.splitlines()):
    if x != 0:
        b = b.split()
        print(b)
        if len(b) == 12:
            x, y, w, h = int(b[6]), int(b[7]), int(b[8]), int(b[9])
            cv2.rectangle(img, (x, y), (w + x, h + y), (0, 0, 255), 1)
            cv2.putText(img, b[11], (x, y), cv2.FONT_HERSHEY_COMPLEX, 1, (25, 25, 255), 1)
            detectedWords.append(b[11])

print(detectedWords)
cv2.imshow('adaptive_img', adaptive_thresholdImg)
cv2.imshow('result', img)
cv2.waitKey(0)
```

Figure 4.10: CNN model compile

Chapter 5

Evaluation

This chapter presents the results of the data extraction from receipts, as well as the results of each module.

5.1 API results

5.2 CNN results

Many different CNN models with slight variations in their parameters were trained and tested. Initially, the models had a low prediction accuracy, predicting the correct category about one third of the time. Tweaking was done to the model by adding dropout, data normalization and max pooling. Prediction accuracy then went up to around 95% for most models.

Validation accuracy and loss are the two metrics we are using to determine the success of a model. FIGURLINK and FIGURLINK shows how the model accuracy and loss are improving with each epoch of training. A consistent theme when training our models was the validation accuracy and loss starting to radically improve between epoch 10 and 15.

A confusion matrix was also used to give some more insight into how the model was performing for each of the different categories it had been trained on. FIGURLINK shows the confusion matrix generated from the same model as FIGURLINKACCURACY and FIGURLINKLOSS, using the same validation dataset. The confusion matrix has three throws, one for each category of receipt the model has been trained to classify. The diagonal of the matrix shows how many correct classifications the model made. The rest of the fields shows how many incorrect classifications were made. In FIGURLINK, we can see that for the second category, all the predictions made were correct. We can also see that most of the incorrect predictions were done on the first category, with 7 of the 8 incorrect predictions taking place here.

When trying to do predictions on a single image, the results were more inconsistent. Predictions were made using images from the same dataset that it was trained on, with varying success despite good accuracy from validation during training. For example, model A and model B might both have gotten validation accuracies over 95%. When tested on single images model A would always predict that category 1 was category 2, while model B would predict the categories correctly.

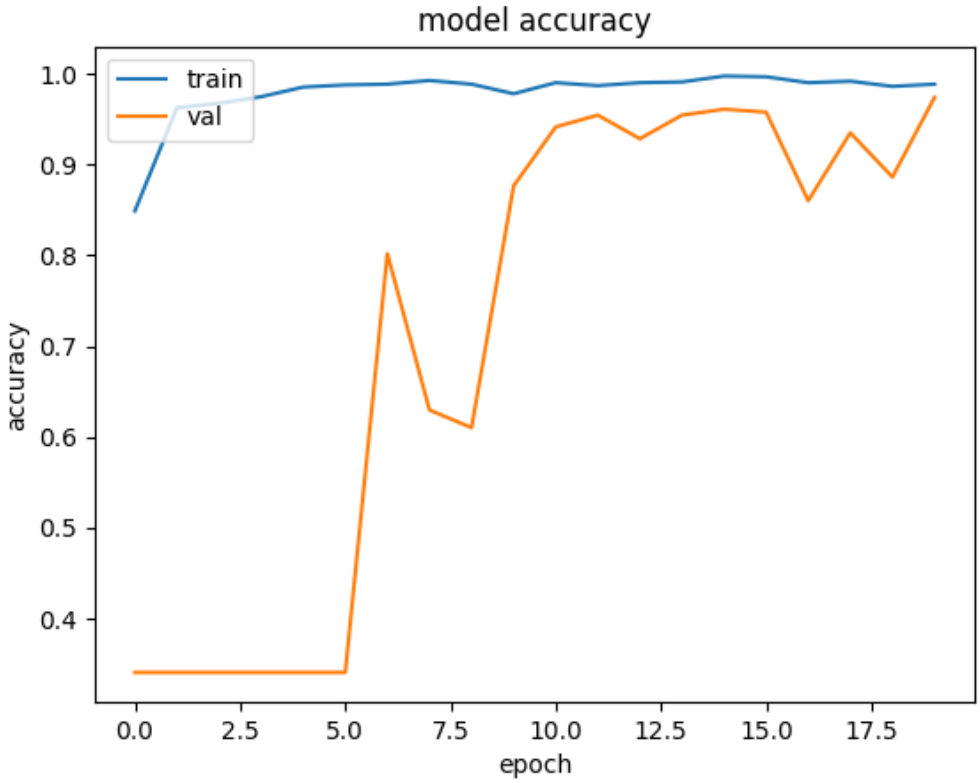

Figure 5.1: Training and validation accuracy for a model trained on 200x200 images

5.3 OCR results

Since our OCR implementation uses pre-defined templates to de-skew images, the output of our CNN plays a big part in whether the OCR will be able to extract text from the images. When the CNN does not accurately predict the correct category for an image, the OCR de-skew will use the wrong image template, and the result of this de-skew becomes an image that is unusable. This can be seen in 5.4 and 5.5

Because of this, the OCR module does not give any text output when the predicted category is not correct. When isolating the OCR module from the other modules, or when the CNN predicts the correct category, the text output from our OCR module is mostly accurate. However, small mistakes in the text extraction can lead to the wrong piece of data being extracted or not found at all.

Figure 5.6 shows the input image sent to the OCR on the left and the image after going through pre-processing on the right. Since the CNN made the correct prediction on what category of receipt this is, the de-skewing algorithm successfully straightened the image. The figure also illustrates the binarization of the image, making the text easier to distinguish from the background. An attempt to extract the text from the right-hand image is then being done.

Looking at 5.7, the text seems to have been reasonably accurately extracted. A date is then found using our regular expression for commonly used date formats.

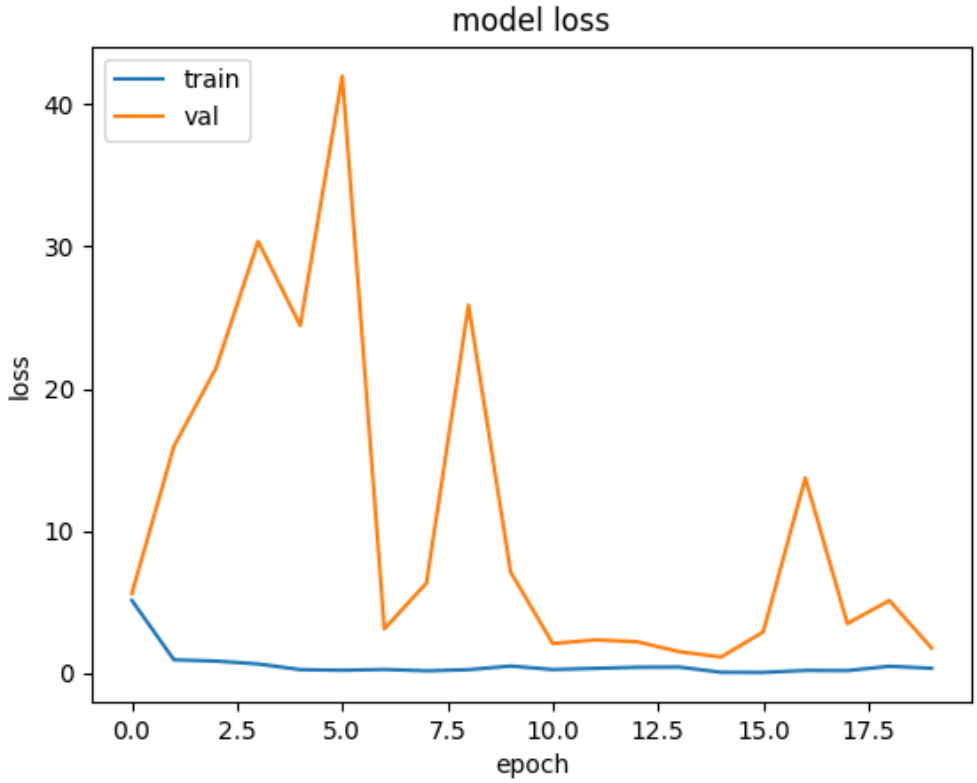

Figure 5.2: Training and validation loss for a model trained on 200x200 images

On further inspection, we can see that the date found in Figure 5.8 does not match the date in the image. The date extracted from the OCR is from 2019, while the actual date should be 2018. This means that the date that is given back to the API and back to the user, is not correct. This example in particular highlights some issues with our implementation and will be discussed more in chapter 6.

The above example is a good indicator for what happens when the CNN correctly predicts the proper category, and when the image is readable by the OCR. One of our categories of images were never readable by the OCR however.

Figure 5.9 shows the image as it is being sent from the user into our pipeline, and after the OCR pre-processing. The CNN made the correct prediction, as can be seen by the image not being rendered unusable by the de-skewing.

Looking at the extracted text in figure Figure 5.10, it is apparent that the OCR did not extract much useful information from the image. Text is being found incorrectly along the edge of the receipt and in empty spaces in the receipt. This is a recurring problem for receipts of this type, and will be discussed more in chapter 6.

5.4 NER results

5.5 Overall results

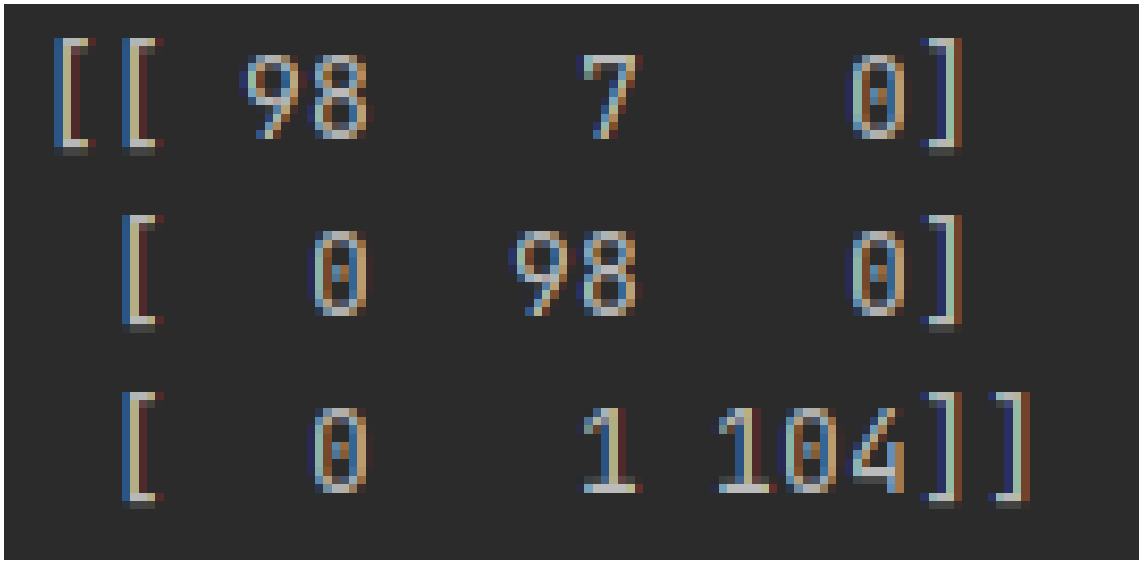

Figure 5.3: Confusion matrix for a model trained on 200x200 images

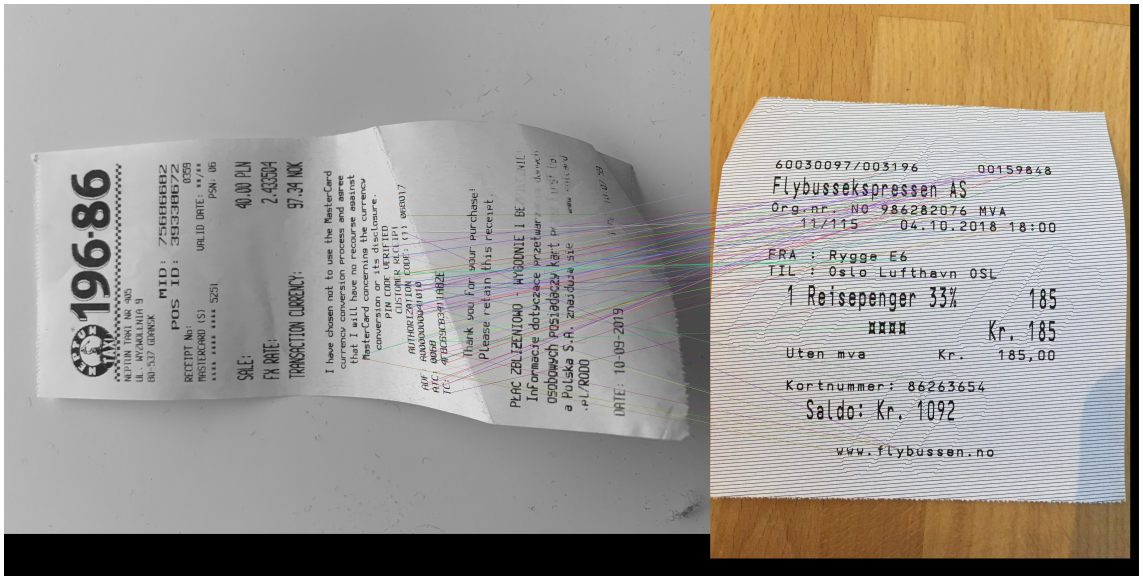

Figure 5.4: Input image (left) trying to match its content to the wrong image template (right)

Figure 5.5: Result of de-skewing when using the wrong image template

Figure 5.6: Before and after pre-processing

Figure 5.7: Extracted text in red, appearing over the place it has been extracted from

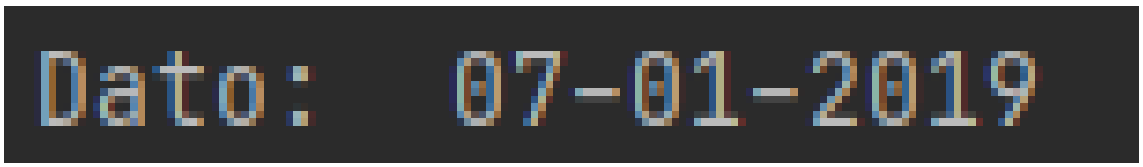

Figure 5.8: The date extracted from OCR output

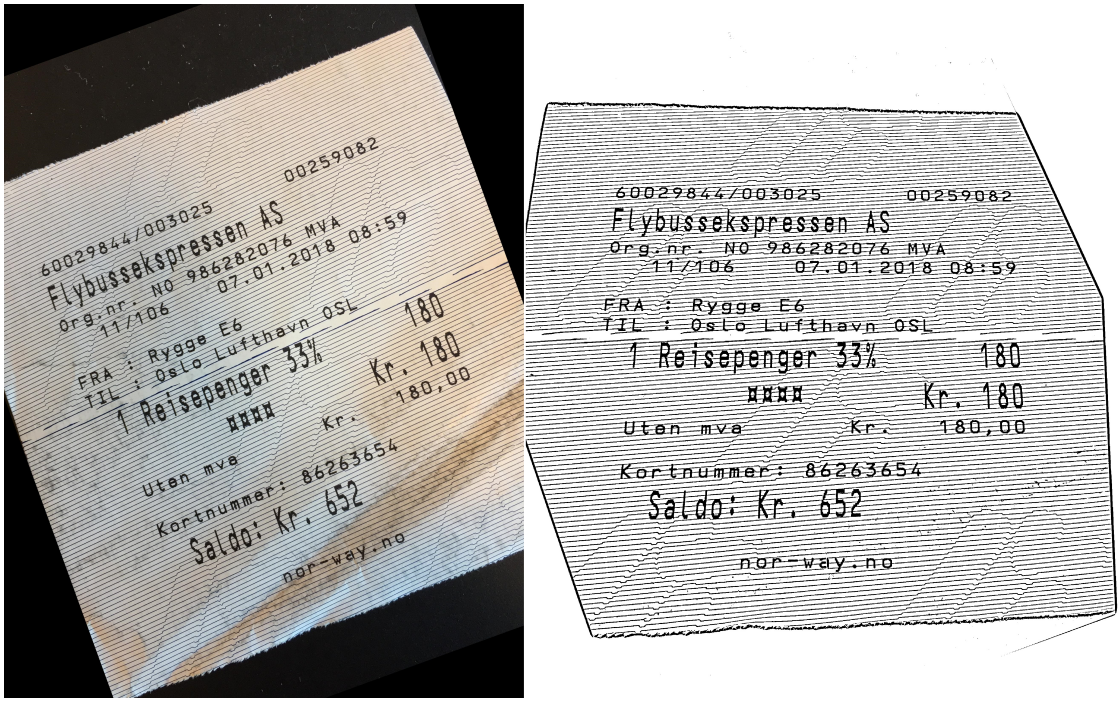

Figure 5.9: Before and after pre-processing

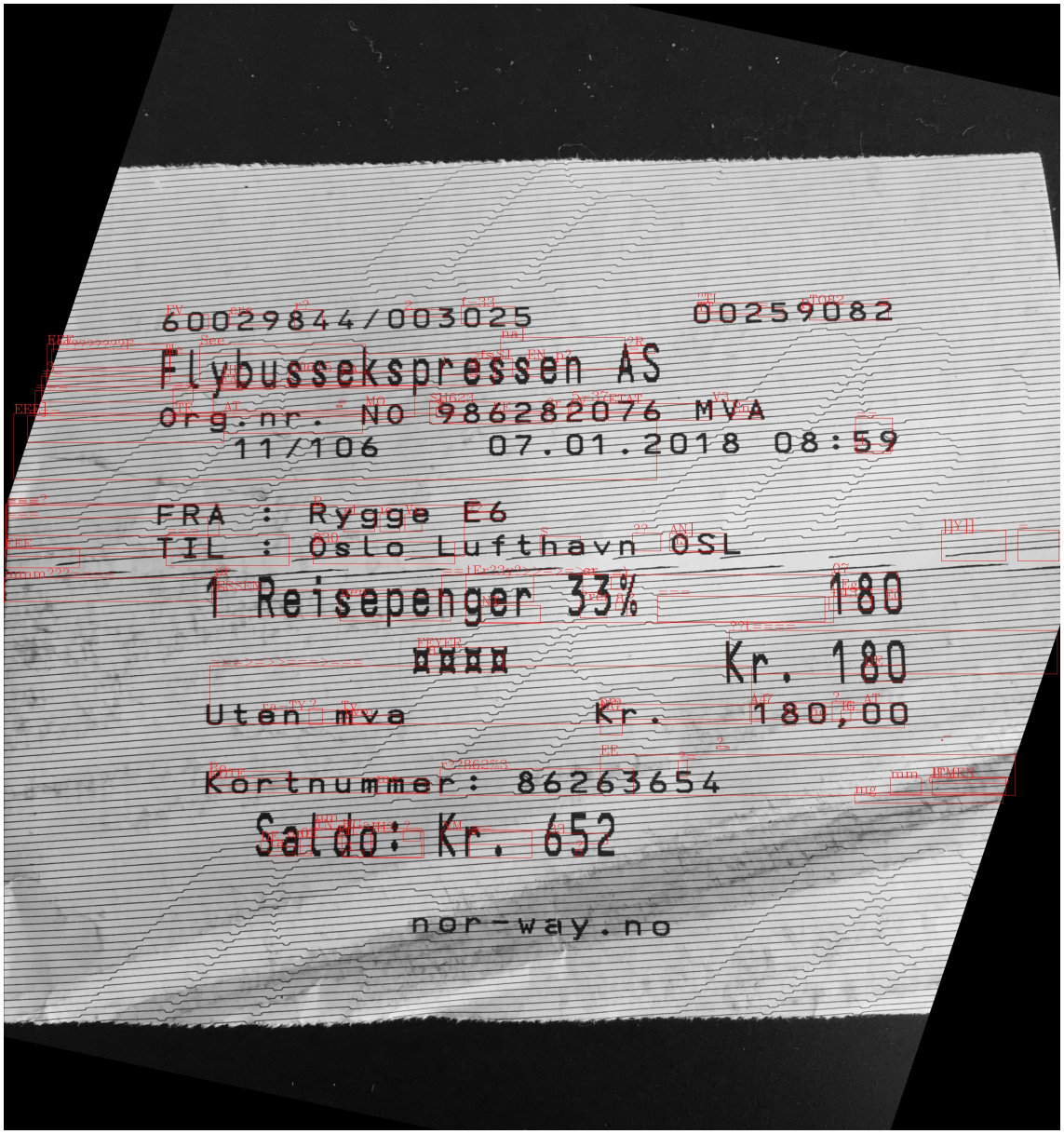

Figure 5.10: Extracted text in red, appearing over the place it has been extracted from

Chapter 6

Discussion

Chapter 6 discusses the results discovered in chapter 5. This includes how the results differ from what was expected, how the findings might be relevant to the thesis' field of study, etc.

Chapter 7

Conclusion

Chapter 7 concludes and summarizes the thesis in a manner that can be read by someone who did not read the entire report.

Glossary

API Application Programing Interface.

AWS Amazon Web Services.

CNN Convolutional neural network.

CTO Chief technology officer.

HIOF Østfold University College.

HRM Human Resource Management.

ML Machine Learning.

OCR Optical Character Recognition.

REST API Representational State Transfer Application Programing Interface.

RNN Recurrent neural network.