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
23rd April 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.

Chapter 3

Design

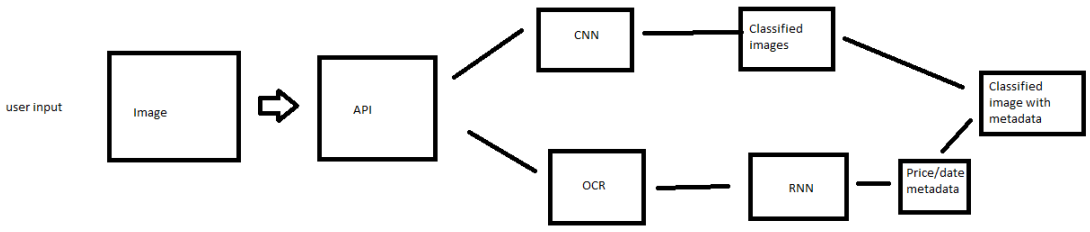

Figure 3.1: Graphic illustrating the project pipeline (PLACEHOLDER)

The project is designed to be able to take an input image of a receipt sent over http, and provide metadata to that image based on it's 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.

API GREIER HER

We have decided to split up the metadata extraction into two separate parts; a Convolutional Neural Network for image classification and a Recurrent Neural Network for natural language processing.

The CNN will take an image as input, decide which company issued that receipt based on previous training data.

The RNN will take text as input. In order to extract the text from the images, an open-source OCR is used.

Finally, the outputs of the CNN and RNN and combined and provided back to the user. This pipeline is illustrated in figure 3.1

3.1 API

3.2 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.

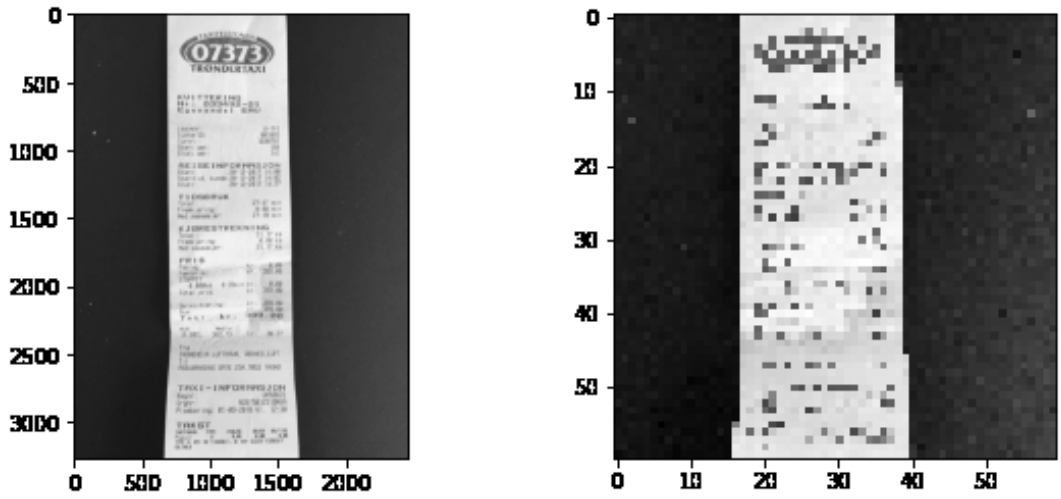

Figure 3.2: Before and after image rescaling

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.

3.3 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.

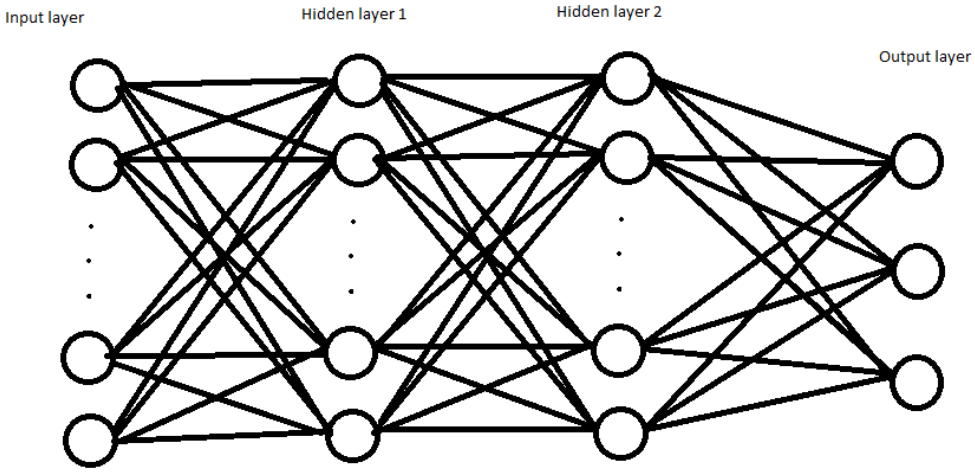

Figure 3.3: Graphic illustrating the Convolutional Neural Network (PLACEHOLDER)

3.4 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.

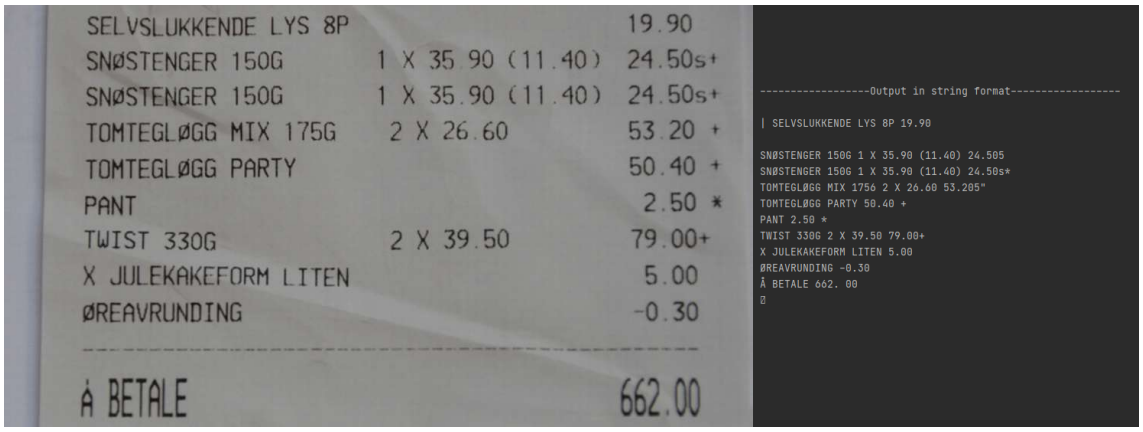

Figure 3.4: Example output of pytesseract with a test image

3.5 Recurrent Neural Network

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 exposes several endpoints that allows for saving an image and a receipt to a database. The API is also connected to the ML and OCR module.

4.2 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.

4.3 Dataset

The dataset provided by Simployer consists of 1194 unlabeled images and pdfs. However, the variance of the types of receipts in the dataset is very large. Because of this, we end up with a very small amount of images of each type after labeling them. The dataset also includes receipts in several languages, mainly norwegian, english and polish. We filtered the dataset to only include norwegian receipts and removing images of bad quality. 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.

4.4 Pre-processing

4.4.1 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.

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

4.5 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 thats called a max operation. We combine the results to get a 2x2 max pool output, down from a 4x4 format. As illustrated below.

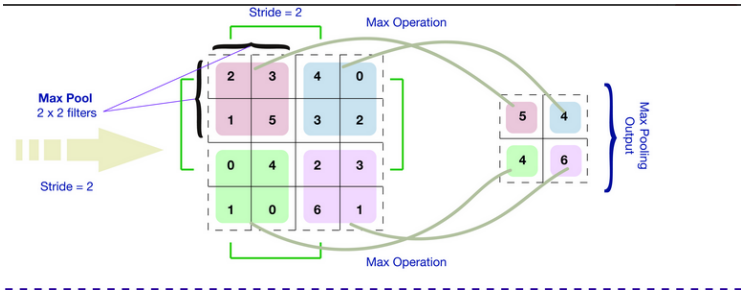

Figure 4.1: Overview for max pooling (Image from https://www.quora.com/What-is-max-pooling-in-convolutional-neural-networks)

We have also used batchnormalixation 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 trainingsteps, 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 4.2: 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 4.3: CNN model compile

Chapter 5

Evaluation

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

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.