
CARPAIR

AALBORG UNIVERSITY
STUDENT REPORT

Information Technology
Aalborg University
6th semester

Title:
Centralization and enrichment of data from online car marketplaces.

Theme:
Programming and Problem-solving

Project Period:
February 2021 - May 2021

ECTS:
15 ECTS

Group:
b606f21

Participants:
Johan Christian Hansen
Simon Eliasen
Thomas Kristian Margon Vinther
Victor Simon Lumholtz

Supervisor:
Xinle Wu

Characters: 110053
Pages: 66

Abstract:

This paper seeks to investigate the possibility of centralizing and enriching existing data spread across online marketplaces, in order to improve the process of buying used cars online. This is done by collecting additional data from Motorstyrelsen.dk, to create universally enriched information on each car listing, on which the user can browse through. The paper presents Carpair, an application that utilizes web scraping and established methods within the field of artificial intelligence, with the *automatic number plate recognition* system, running with an approximate 76% accuracy. The application is a minimal viable product that can be scaled towards an application that centralizes the majority of all online car sales in Denmark. The paper showcases promising results in regard to the real world applications of neural networks.

Preface

The paper is written by 6^{th} semester students studying Information Technology(BaIT) at Aalborg University.

The goal of the semester was for the project group to demonstrate knowledge in specific techniques and concepts within the scope of computer science. The project group should then be able to convey this knowledge through the analysis and problem-solving of the research problem.

Acknowledgment

The project group would like to thank our supervisor, Xinle Wu, for the feedback and knowledge that was needed to produce the project.

Johan Christian Hansen
jcha18@student.aau.dk

Victor Simon Lumholtz
vlumho18@student.aau.dk

Simon Eliasen
selias18@student.aau.dk

Thomas Vinther
tvinth18@student.aau.dk

Table of Contents

1 Introduction

The car is for many an essential and necessary tool to make their daily lives function. It is required to get to work, social events or daily errands. The number of registered cars in Denmark are increasing (Statestik [2021]) and likewise is the rate of which cars are being purchased [Statestik, 2019].

As the market of cars expands, so does the market for used cars, which additionally has seen a record growth of 7,6 % in the period from 2019 to 2020. meanwhile, the sale of new cars had dropped by 12,2 % in the same period [Steensen, 2020]. These trends are assumed to be contributed by a series of different factors, but largely attributed to COVID-19. Because of the pandemic, the availability of public transportation had been limited which has resulted in people having to find other means of transportation. Many car dealerships has had low stock of available cars due to limitation in production and distribution, which resulted in more people buying used cars [Steensen, 2020].

Although used cars are more popular than ever, used car sales still comes with some issues. One issue could being that when dealing with a private seller online, it can be difficult to get all the specific details linked to a car. A private seller could choose to omit important information in their car listings such as when the next service is required or the weight tax of the car. An example of such a listing can be seen in figure 1.1 displaying a listing from the online marketplace "Den Blå Avis" shortened to DBA throughout the paper. This information is vague and offers no concrete information on when the car needs to be serviced.

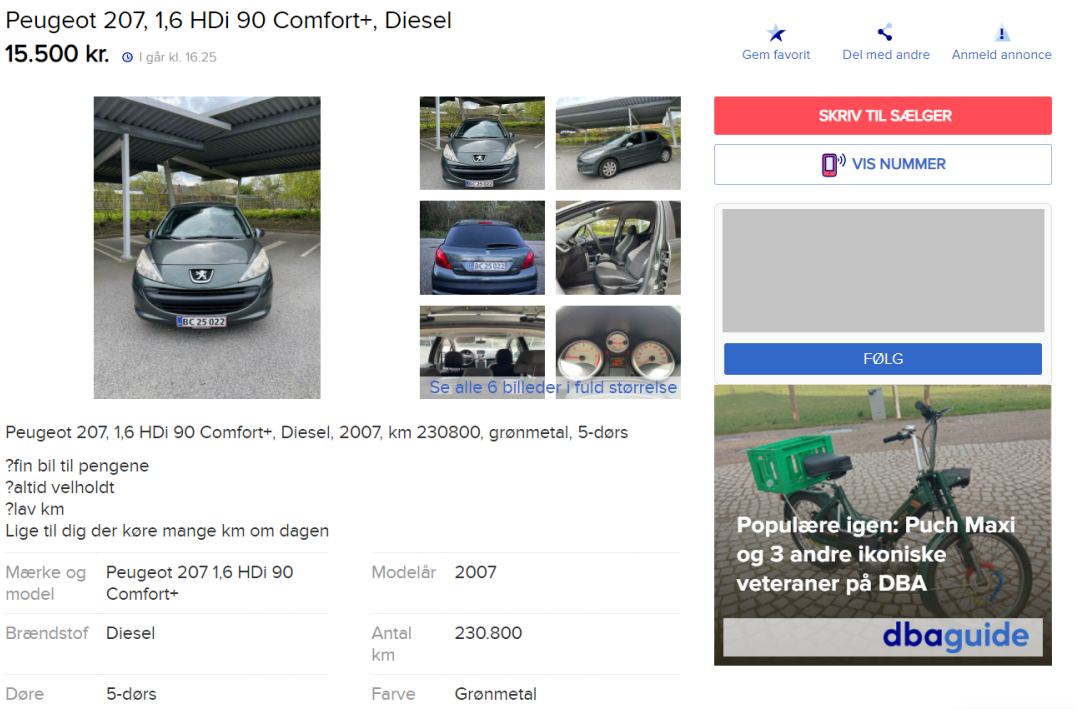

Figure 1.1. Example of a Car listing on DBA

The problem with missing information on cars can be solved by looking on other websites dedicated to finding information on cars, by inputting the number plate. An example of this is the official site for the Danish Motor Vehicle Agency which throughout the paper will be refereed to as MVA [Skat.dk, 2021a]. It can be time-consuming for the user as they must go back and forth between different websites each time they wish to look for a car to ensure that the information is in order. Each website selling used cars have different car listings which requires the user to look through multiple online marketplaces to find a suitable car.

Because of the increased interest and demand in used cars as well as the previously described issues related to the online marketplaces selling used cars, the question of whether it was possible to create a software solution to improve the process of buying used cars was raised. The initial premise for the software solution was to develop improvements to the aforementioned issues. There currently exists various websites that each offer different listings of used cars, such as DBA, and websites that provides data on most registered cars using their number plate, such as MVA. This indicates that some interest or need exists for the inquiry of such data related to cars in Denmark. None of the sites for checking information on cars based on number plates, which the project group had inspected, were tightly integrated to an online marketplace, these being "NummerpladeTjek.dk", "TjekBil.dk" and the MVA site "motorregister.skat.dk". Neither did any of the major online marketplaces link to registers with information pertaining to the number plate.

The software solution should therefore solve the issue of scattered information by encapsulating

all relevant information about each car on one page. This would make it easier to find all the important information for any car without having to access multiple sites. It should likewise be presented in such a way that it is easy to browse through a large selection of cars.

To facilitate easier browsing of the posted cars the system should employ a set of filter options that allows the user to specify their search rather than simply looking through a long unorganized list of cars. Some examples of potentially relevant parameters could be the acceptable price range, kilometers per liter, the brand of the car and the weight tax. The site could potentially contain thousands of different cars, so it would take an unreasonable amount of time to browse through if no options to filter were provided.

1.1 Research Question

The focus of the project is to develop a software solution that combines the requirements presented in the previous sections into one system. The Research question for the project could therefore be defined as the following:

Can the lack and scattering of information in online car marketplaces be solved by providing a centralized platform that enriches the car details.

1.2 System outline

To better define what the application should contain in order to fulfill the research question and address the underlying issues in regard to lack and scatter of information in car sales, an outline for the application is constructed. For the sake of simplicity the project will be developed as a minimal viable product (MVP), both in regard to establishing a realistic timeline but also to make it more adaptable to changes, without having to maintain large and complex data sets.

The application should be able to collect data from the online marketplace DBA and utilize the data to get further information through the external service the MVA. the data is then saved to a database, which the website should be able to retrieve data from and then display. Each listing of a car should be displayed with a picture and the data attributed to it as a card which also links back to the original listing on DBA. The website should support a set of filter options that can limit which cars will be displayed. It should be sufficiently fast and contain no major flaws or errors critical to the system. In addition to the previously mentioned features a recommendation system could also optionally be implemented in order to increase the accessibility of the application.

The initial outline for the system can be seen in figure 1.2

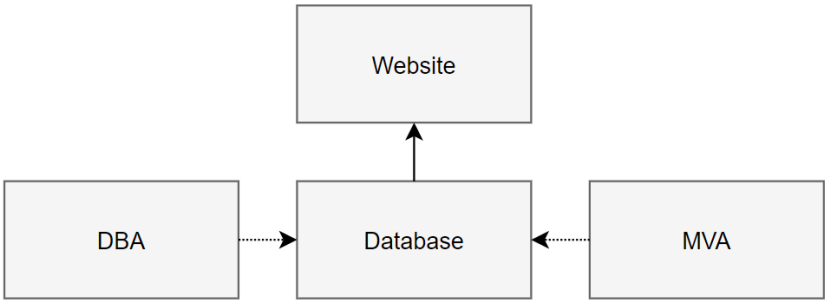

Figure 1.2. General outline of the system

These components will be elaborated upon in chapter 3 "Application design".

Planning the project 2

The project follows the Scrum methodology as described by Ken Schwaber [2020] in *The Scrum Guide* as a method to better plan out and manage projects. Scrum introduces the use of sprints, in an effort to organize and segment work into manageable bits. Each sprint will contain a planning process, where the project team define the purpose of the sprint, what should be achieved within the sprint and how long it should span for. When a sprint has been concluded, the project team evaluates the given sprint to investigate if the intended goals were reached and then plan the following sprint, thus repeating the process.

2.1 Scrum

The Scrum methodology was chosen because of its flexible, transparent and iterative framework used for project design. It enables an agile approach to application development as well as a larger focus on collaboration within the project team. The team is a central point when working agile with Scrum. The members of the Scrum team are closely working together and communication is therefore a key component to ensures that the project becomes much more transparent for those involved. A concept such as pair programming, where developers in the Scrum team works together on developing code, can be used to improve communication, create a better collective understanding of the work and lessen the hindrance of code-ownership [Böckeler, 2021]. By working together iteratively, it becomes easier to adapt to changing goals and sudden challenges throughout the project as the collective work is reviewed in every sprint allowing for the project team to correct or make changes in the planning for the next sprint. It is likewise much easier to resolve software and code related issues with the project team working closely in collaboration.

An alternative to the Scrum method could be to work with the waterfall approach that works much more linearly. In waterfall, each step of the project is extensively planned and mapped out at the beginning of the project and only after the project design is completed the development work initializes. This approach offers less inherent flexibility in its approach. The work on each component is also more fragmented.

In regard to testing and ensuring that the intended functionality of the application is correct, the agile approach offers some advantages for the project. As an example each sprint will involve some amount of testing to ensure, not only that the goals of the sprint is reached but also that

the collaborated work is compatible with the work of the other members of the Scrum team. Opposed to this, the waterfall approach focuses on testing towards the end of project once the components are finished.

It was evaluated that the optimal approach to the project would be the agile Scrum approach, both because of its strength as outlined for facilitating the project where some components and the required software solutions were foreign to the project team. This meant that a more flexible and exploratory approach was desired.

2.2 Implementation of Scrum

The agile approach of Scrum works by implementing sprints into the project in question as well as defining a Scrum team and a project backlog. A Scrum team normally consists of a project owner, a Scrum master and a development team. The development team incorporates different types of talent, making the Scrum teams cross-functional.

The Scrum master needs to ensure the teams' effectiveness by improving the conditions for working within the Scrum practices. The Scrum master is therefore required to have a firm understanding of the Scrum theory and practice to be able to help the other members of the team. For this project no direct Scrum leader was chosen as no member of the project team had any special prior experience working as a Scrum master. Instead, the Scrum team as a whole was tasked with familiarizing themselves with how to apply and follow the Scrum theory and help each other throughout the project to achieve the best possible result.

It was essential to have a product owner that was tasked with maximizing the value of the work produced by the Scrum team. The product owner was selected among the members of the project team and was tasked with ensuring that the project backlog was maintained and properly communicated with the rest of the team, making the product owner function as a form of intermediary between the backlog and Scrum team. The product owner for this project was also tasked with keeping an overview of the road map for the project. This meant that the product owner undertook an extensive amount of the project planning. The Scrum team as a whole contributed to maintaining the plan which was facilitated by the agile focus on communication between the different members of the team.

The product backlog is an ordered list of problems or items that the Scrum team must solve or complete to satisfy the requirements for the product as a whole. The items within the product backlog often undergoes refinements such that each item can be divided into smaller and more precisely defined items, which then can be designated to each sprint making the goal of the sprint more clear.

The product backlog operate as a list of the ongoing and upcoming work for the project. The Scrum team would dissect the work outlined in the research question into smaller problems

that each had to be solved to complete the application. These were then assembled into the product backlog for the project containing a complete list of the required work that had to be completed. The Scrum team would therefore have a rough idea of how to structure each sprint as some problems would have a logical sequence in which they would have to be solved. The case being that each component had to be completed first or at least be functional to a point where they could solve the required task. Although some broader outline was planned, the Scrum team would by each sprint investigate the items on the product backlog and analyze how to convert them into more precise and tangible tasks and then plan the sprint to complete the tasks deemed most necessary.

2.2.1 Sprints

Sprints are the reoccurring phases that are essential in the Scrum methodology. Each sprint will have a duration of approximately 30 days and each sprint will start with an investigation into what the focus for the individual sprint should be. Once the sprint investigation is finished, the intended outcome of the sprint will be discussed based on the product backlog to determine what was achieved and if the goal was reached. Lastly the acquired knowledge and experiences from the sprint will be used to determine how the following sprint should be planned and what new goals should be reached, in accordance to the outlined product backlog.

For this project, the product backlog mainly consists of the different aspects of designing, developing and testing the application outlined in chapter 3 "Application design" of this paper. The Product backlog would divide the components of the application into different items on the product backlog such that each sprint could focus on different stages of each component. The first sprint of the project would therefore mainly be concerned with converting the broad issues raised by the research question into more actionable and defined solutions that can be translated into specific tasks. This means that the initial sprint would focus on the core components of the application to create fundamental functionality that could then be built upon. The following sprints would focus on expanding upon and optimize the core components as well as moving towards a coherent system with the component connected. The following sprints would then be where the system would be polished and other less important aspects of the system would be finished, such as user interface. The last sprint of the project would be dedicated to finishing the work on the project paper and any remaining formalities related to the project.

Application Design 3

This chapter will deal with the construction of the application and its individual components as well as the purpose of each component and how each are designed and connected. The chapter expands upon the system outline presented in chapter 1.

3.1 Application components

The system will be composed of:

- A web scraper that collects car data from DBA
- The automatic number plate recognition that is responsible for identifying the number plate from cars based on a picture of them
- A web scraper that collects car data from MVA based on their number plates
- A web application showcasing the available cars.
- A database that stores all the collected car data

These components each serve a specific purpose in the overall system and have dependencies to each other to make the system operate successfully such as the database that stores information that is necessary for the other components to function. The dependencies of the different components can be seen visualized in figure 3.1, where a direct dependency is denoted by a pointed arrow.

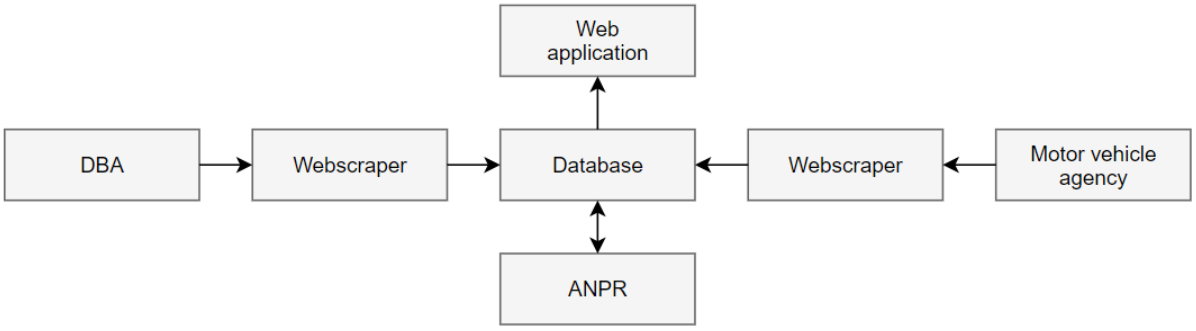

Figure 3.1. Application components and their respective dependencies

3.2 Individual components

This section will describe each component and their underlying design. Each component will be explained in regard to how they function together as shown in figure 3.1.

3.2.1 Web scraping DBA

The web scraper collects data from DBA, this includes the images, price, link to original listing and zip-code describing where the car is located. The images are particularly important as they are used by the ANPR to extract the car's number plate, which in turn will be used to extract additional data by the MVA web scraper. It is required that the system should be able to pull multiple images of a car listing, in case that the number plate is not contained within the first image in the listing. The DBA web scraper is further elaborated upon in section .

3.2.2 Automatic Number Plate Recognition

In order to populate the web-application with enriched data, the Automatic Number plate Recognition (ANPR) pipeline utilizes the data gathered from the DBA web scraper. The image URLs are processed by an object detection model to successfully identify number plates in the images. In continuation of this operation, a series of processing steps are applied to detect characters in the image. The individual characters from the number plate are then passed on to the Optical Character Recognition model (OCR) which translates the characters into a text-string which then can be passed to the MVA web scraper to gather detailed information on the car, that may not have been provided in its original car listing on DBA. A more detailed description of the elements contained within the ANPR component will be provided in chapter 5.

3.2.3 Web scraping MVA

This web scraper functions similarly to the one used for DBA, but instead collects data from the Motor Vehicle agency. The web scraper gets a string value from the ANPR containing a number plate that it uses to navigate and get the specific vehicle information tied to the given number plate on the MVA website. The Web scraper will gather the kilometer status, km/l, weight tax and time for next checkup on the cars and inserting it into the database which is then later displayed onto the website. Which is further elaborated in section 4.2.

3.2.4 Website

The website will be built with a combination of Hypertext Markup Language (HTML), for building the structure of the website, Cascading Style Sheets (CSS), for styling the visual representation of the website and JavaScript to support dynamic functionality of the website. Furthermore, Node.js is used as the primary tool for server-side operations in the application.

Node.js allows for the back-end to be written in Javascript, which makes the application more maintainable and simpler to develop as the front-end and back-end uses the same programming language. The back-end is mainly used to retrieve the correct data from the database in order for it to be used by the Javascript on the front-end of the application. This is described further in section 6.

Filter Options

The filter options will be created using a HTML form with a series of input fields where the user can adjust different metrics which will be used to create a query to the database based on their choices. The query will then only retrieve cars that fulfil the filter limitations, which then will be displayed on a result page containing the car listings that fit the users input which is further elaborated on in section 6.3.

Recommendation system

The website should contain a recommendation list of cars based on their economic appeal. The three most economically appealing cars will be shown to the user as the recommend cars to look at. This will be described in section 6.4.

Visual Design

The website of the application will be minimalistic in its form, with an emphasis on conveying the most important information to the user, thus increasing the accessibility. As a result of the focus on accessibility the colors are also toned down, to reduce distractions that removes the users' attention from the functionality of the application. To further enhance the quality of the website, the framework Bootstrap is used, in an effort to provide a responsive design that should convey its message successfully on all devices and browsers.

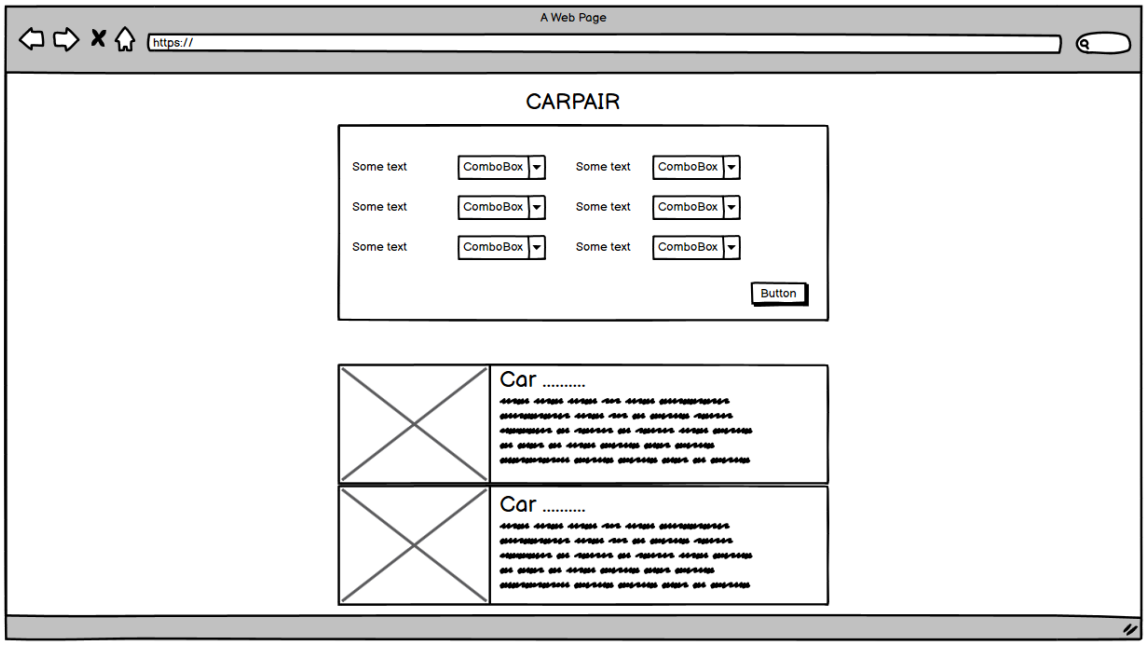

Figure 3.2. Wireframe of the websites homepage

Figure 3.2 showcases a low fidelity wireframe that displays the core elements of the website. The figure reflects a focus on displaying the most necessary information upfront, and restraining from including moving elements and complex structures that draws the users' attention away from the functionality of the application. The visual design draws inspiration from existing marketplaces that also sell cars.

3.2.5 Database model

The database will contain all the gathered information from the ANPR and the web scrapers. The data that is collected in the database is shown in figure 3.3. This information is displayed on the website to provide the content. The car data will eventually be erased when the cars are sold or are no longer available, which is done by the web scraper for DBA. The reason that the information is erased is to avoid overcrowding the database with unavailable cars. So that users will not see cars that are no longer for sale. The database schema is shown in figure 3.3.

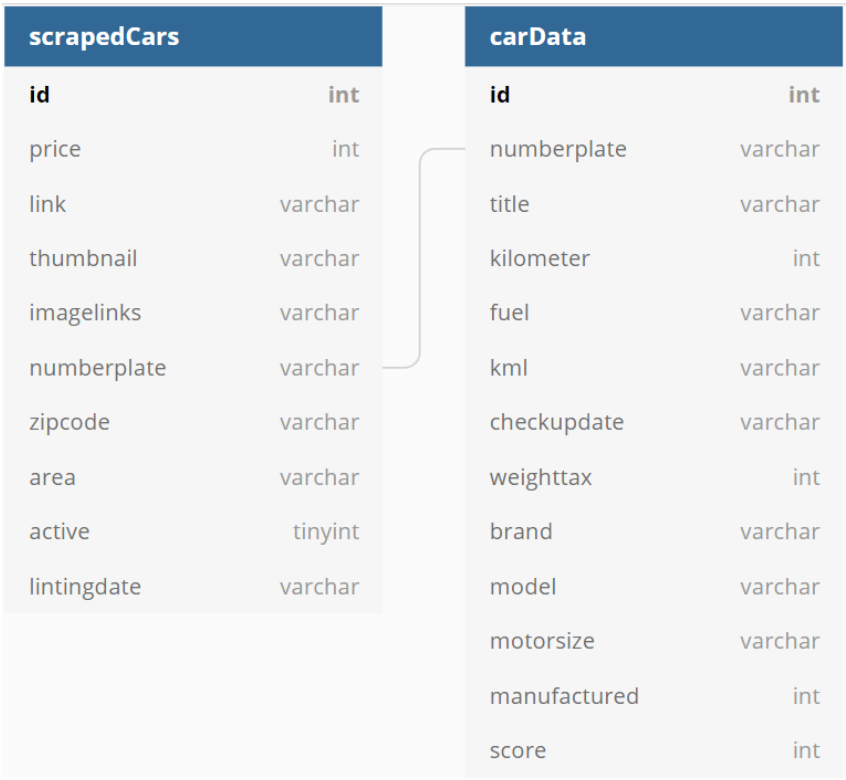

Figure 3.3. Database schema

As figure 3.3 shows, the database only contains two tables. *scrapedCars* is used for the scraped cars from DBA and *carData* is used for the data that is scraped from MVA. The cars are then shown on the website using an *inner join* on *numberplate* that combines the information of both tables on a given car. Most of the table-columns are evident, but some of them requires further clarification. The *zipcode* column in *scrapedCars* is a *varchar* because the zip area is included in the text, for example: "9000, Aalborg". The *area* column in *scrapedCars* contains the region in the country, e.g. "Nordjylland". The *manufactured* column in carData is an integer because the application only displays the year it was manufactured. The *active* column is used by the DBA web scraper to determine which cars will be deleted. When the web scraper starts, each cars' *active* status is set to 0. When a car shows up in the results, its status is set to 1. At the end, all the cars that remains with an active status of 0, will be deleted. The reason for including this in the database instead of simply implementing it in the DBA script itself, is that this approach makes the data gathering more scalable. It allows for parallel scraping of DBA, so that multiple devices can be scraping the website simultaneously. This is important, because there are a lot of cars for sale on DBA, so each web scraping is very time-consuming as it has to loop over all the results. The run time would be drastically reduced by web scraping on multiple devices simultaneously. Furthermore, this approach saves memory on the web scraping devices because they do not have to store all the car links and statuses themselves. This is very important, because if the platform should be published, then the web scraping would probably run on small economical devices such as Raspberry Pi's that have a very limited memory.

Data gathering 4

The required information for the system is based on data from DBA.dk and MVA. In order to gather this data, a web scraper is used. Web scraping is a way to automatically gather a lot of data from a website [Kaur, 2020]. The web scraper extracts the desired data from the unstructured HTML code and then transforms it into structured data in a format such as an Excel sheet or a database.

4.1 Webscraping dba.dk

The web scraper needs to gather information from a large amount of cars. In this paper the project group will work with cars from Aalborg (approximately 80 cars), as it makes the data more malleable because the scraping duration otherwise would be very time-consuming. However, if this platform should be available and useful for the public, it should include all the cars on dba.dk. In this regard the project is not meant as a finished product, but simply as a demonstration. The project group wants to scrape cars that include number plates, so only privately sold cars are scraped.

Approach

The first step for the web scraper is to go to *https://www.dba.dk/biler/reg-aalborg/?fra=privat* in order to show all the privately sold cars on dba.dk. This page is shown in the figure below.

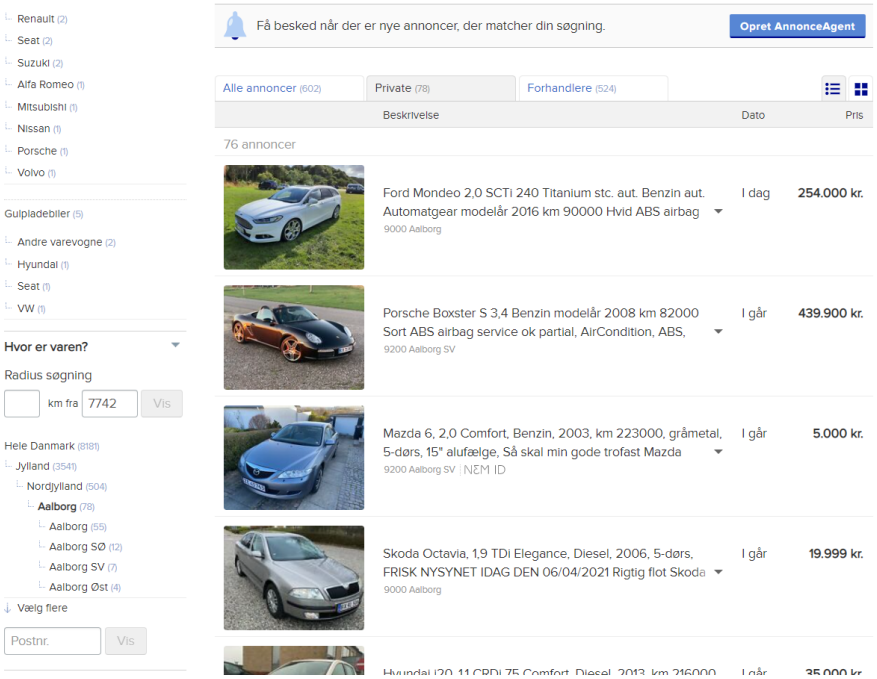

Figure 4.1. Cars on dba.dk

Next the scraper should loop through all the car listings. For each of the listings, that have not already been scraped, the web scraper should gather the price, image links, and zip code for the car listing. This data is available on the specific car page as visualized in the red boxes in the figure below.

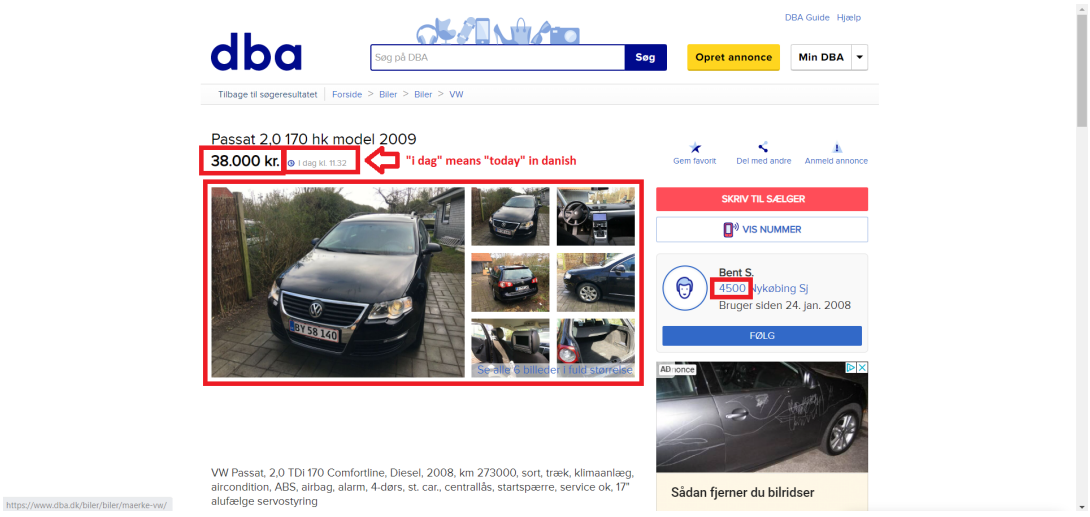

Figure 4.2. Car details on dba.dk

Then the scraper should identify this data, and insert it into the database.

For all the results that already have been scraped, the web scraper should check if the price is changed. If that is the case, then the car price should be updated in the database. This is achieved by comparing the price on dba.dk with the price in the database.

This script should then be executed with a specified time interval in order to keep the data updated. The project group aims to run it twice a day, but as a finished product, it would be better to run it more frequently.

4.1.1 Removing cars

In order to keep the platform updated, the system needs to know which cars are still for sale. So each time the scraper is executed it needs to check all the cars that are still for sale and compare them with the cars currently in the database. This is implemented by giving each of the cars an "active" status in the database. At the beginning of the script all the cars statuses are set to inactive. When a car is found by the web scraper, then the car status is set to active. At the end of the script the software then removes all the cars from the database that still have a status of inactive. This is visualized in figure 4.3.

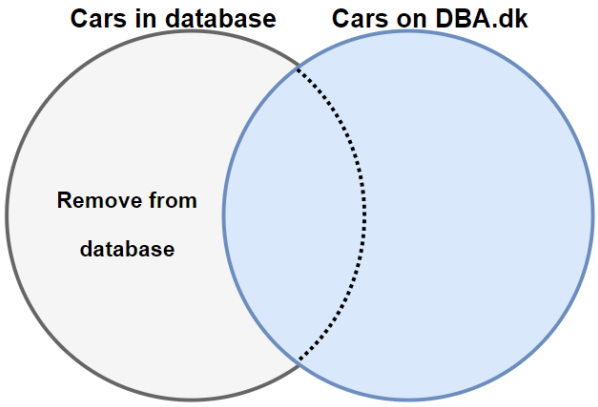

Figure 4.3. Cars that should be deleted in the database

4.1.2 Technologies used

The web scraper is written in python. The web browser automation is done with a python module called selenium which enables us to select and click on specific elements on webpages. Then SQL is used to insert the data into the database. The script is then run on a computer.

4.1.3 Implementation

The implementation of the web scraper in the form of a python script. The main structure of the script is finding the needed information on the dba.dk page and inserting it into the database. The most important method for the script is shown below:

```
1     def scanResults(self):
2         linksTmp = self.driver.find_elements_by_css_selector(".expandable-box a.listingLink")
3
4         for i in range(len(linksTmp)):
5             links = self.driver.find_elements_by_css_selector(".expandable-box a.listingLink")
6             try:
7                 linkUrl = links[i].get_attribute("href")
8             except:
9                 continue
10
11            if linkUrl in self.oldCarUrls:
12                print("Already taken")
13                self.dbSetActive(linkUrl)
14                continue
15            self.driver.get(linkUrl)
16            sleep(15)
17            self.scanCarPage(linkUrl)
18            #sleep(5)
19            self.driver.back()
20
21        self.goToNextPage()
```

The method loops through all the car listings on a DBA page. For each listing the script checks if the car is already scraped and therefore already in the database (lines 11-14 in the code above). If this is not the case, then the script open the details page for the car and then imports the required data into the database. After the script have run through all the results on a single page, then it simply repeats the process on the next page (line 21).

The method for inserting data into the database is shown below.

```
1     def importData(self, price, link, zip, area, dateStr, imageLinks):
2         mycursor = self.mydb.cursor()
3
4         try:
5             thumbnail = imageLinks[0]
6         except:
7             thumbnail = " "
8
9         imagesStr = ""
10
11        for img in imageLinks:
12            imagesStr += "" + img + ", "
13
14        sql = "INSERT INTO scrapedCars (price, link, thumbnail, imagelinks, zipcode, area, active,
              listingdate) VALUES (%s, %s, %s, %s, %s, %s, %s, %s)"
15        val = (price, link, thumbnail, imagesStr, zip, area, 1, dateStr)
16        mycursor.execute(sql, val)
17        self.mydb.commit()
18        print(mycursor.rowcount, "record inserted.")
```

This method inserts the price, link, zip code, area, date and images into the database.

4.2 Webscraping MVA

The script for web scraping MVA is very similar to the one for DBA. The primary method is shown below:

```
1     def openPage(self, url, numberplate, price):
2         self.driver.get(url)
3         self.driver.find_element_by_id("regnr").click()
4         self.driver.find_element_by_id("soegeord").send_keys(numberplate)
5         currentUrl = self.driver.current_url
6         self.driver.find_element_by_id("fremsoegKtBtn").click()
7
8         self.waitForPageLoad(currentUrl)
9         results = self.scanPage(price)
10        self.updateDB(numberplate, results)
```

This method is run for every number plate in the database that have not yet been used to scrape data on MVA. The approach of the method is to open the MVA website and search for the number plate and then show the results for the car (lines 2-6). The is seen in figure 4.4.

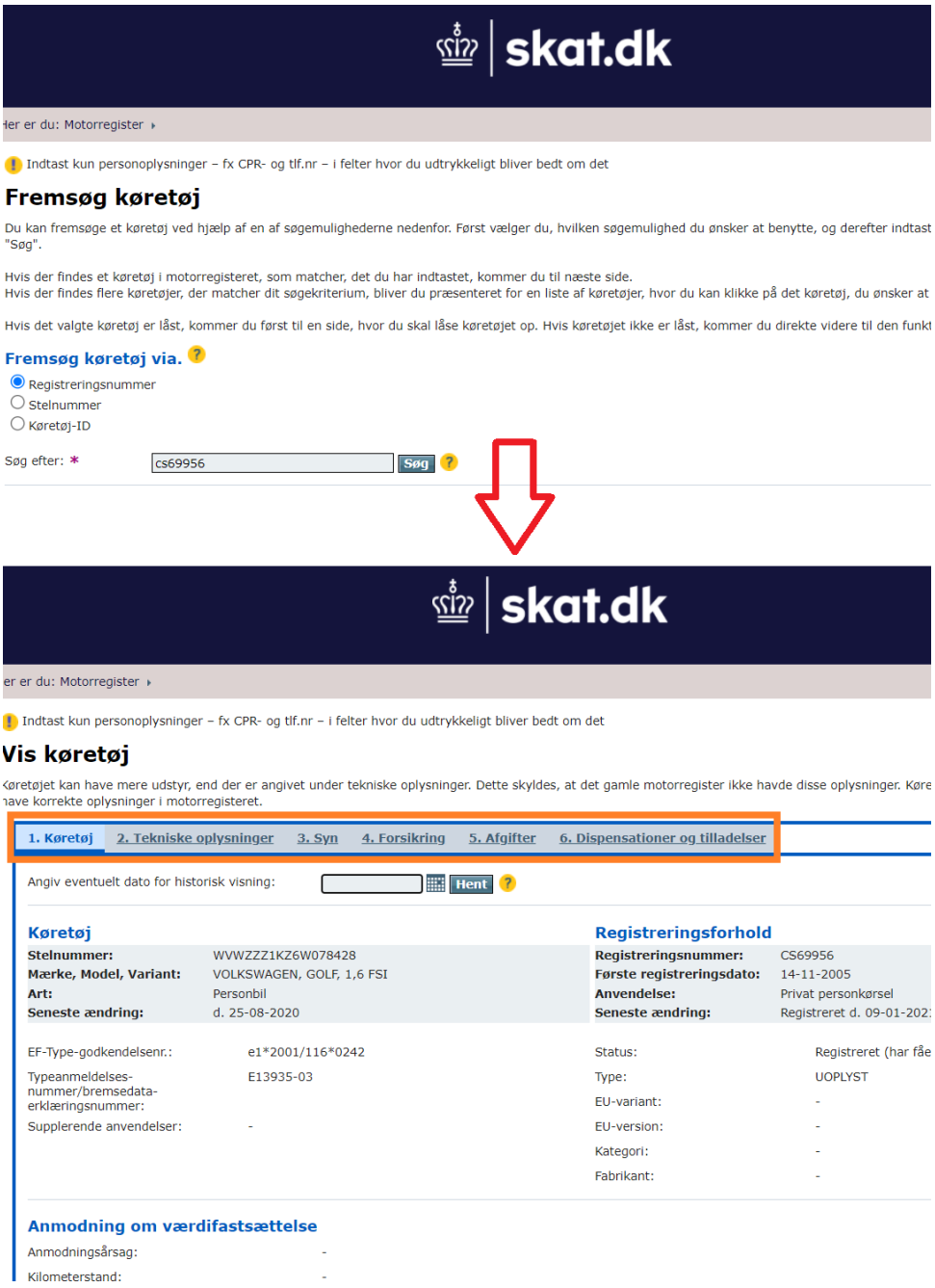

Figure 4.4. The MVA web scraper

Afterwards, the needed results will be collected and inserted into the database (lines 9-10). This is achieved by clicking though each of the required tabs shown in the orange box in figure 4.4, and then identifying each data point and inserting them into the database.

The script also calculates a score for each of the cars, which will be used to recommend the most economical cars on the platform. This will be elaborated on in section 6.4. .

Automatic number plate recognition 5

To extract data from the listed cars' number plates on DBA, a query through all the privately listed cars needs to be made, to extract their respective number plates from the listing photos. In order to obtain the number plates automatic number plate recognition(ANPR) is utilized. The system will be divided into three sub-problems:

1. **Object detection:** Identify if, or where a number plate is placed in the image.
2. **Image processing:** Identify all characters in the number plate
3. **Optical Character Recognition(OCR)** Read the characters and convert them into a text string.

This chapter will showcase the methodological approach of creating an end-to-end ANPR system that reads and extracts characters from images containing danish number plates.

Existing tools

In effort to create an ANPR, multiple existing tools were investigated, such as openALPR [OpenALPR, 2021], but were quickly deemed unfavorable due to the high cost of processing images in large quantities both for testing and running the actual application. This choice is also deemed favorable in regard to a lack of dependency on third party solutions, which can not necessarily guarantee their up-time, continued support and updates which might change or break the full application without proper maintenance. In addition, the lack of customization also played a role in the decision to develop a new model and furthermore the lack of focus on images taken in unconstrained scenarios.

Challenges

Creating an ANPR-system that successfully reads number plates can be very challenging especially when there is no control over the images which are being sent to the model. This means that instructing or favoring the placement of the number plate in the image is not a possibility. This sets up a large variation of challenges that can occur in an image, which the model should correct for. These challenges can be seen listed below.

- Bad lighting

- Bad image quality
- Image is skewed unfavorably
- Image is taken far away from the number plate
- Number plate is damaged or obscured
- Multiple number plates in one image
- No number plate or only a partial number plate in the image

In addition to these issues that might arise in a singular image, datasets of number plates are not widely available. Either a model with pre-trained weights from another country have to be used, or mining and labeling of number plates needs to be done manually. To minimize the large variation of challenges, the system will only focus on the two most broadly used number plates in Denmark as displayed in figure 5.1.

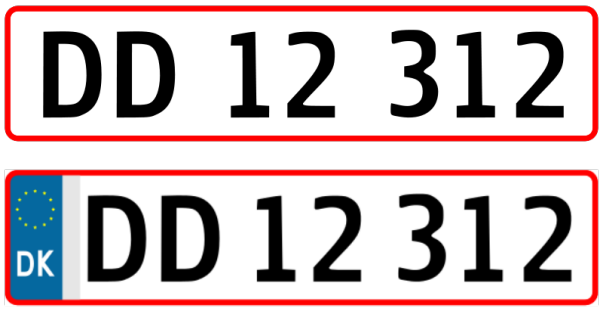

Figure 5.1. Most common number plates for privately owned cars in Denmark

Development approach

The methodology of development in regard to the ANPR will be centered around utilizing a brute-force approach and then further optimizing the result based on challenges identified in the crude approach, such that a wider spectrum of knowledge continuously is gathered on the subject, with the intention of producing a better implementation. The initial focus will be on identifying number plates from images and then later using processing on the images, to then use OCR to transcribe the text. The success rate of each iteration will be determined on successful identifications on a small dataset gathered from DBA containing 100 cars with a varying degree of skew in the image, image quality, distance from camera and multiple number plates in the images. The success of each iteration, will be determined based on the result returned by running the ANPR on the test dataset.

5.1 Object detection

Object detection is related to the field of computer vision and concerns itself with identifying instances of a specified object in images or video. The preferred approach to object detection is the utilization of neural networks.

5.1.1 Neural networks

Neural networks are classified as a series of neurons or nodes interconnected by edges or synapses presented in a graph structure, that can be loosely thought of as an imitation of the human brain. Each node in the graph has an input (x), weight (w) and output (y) assigned to it, where the weight represents the relative importance of the connection between each node. The sum of all weighed inputs is then passed through an activation function, which produce the output of the singular node, as depicted below [Meng et al., 2020].

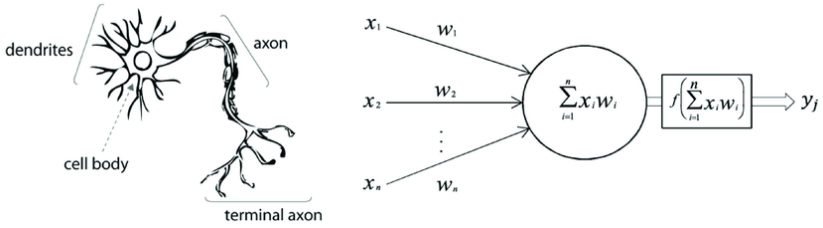

Figure 5.2. Neuron and node analogy [Meng et al., 2020]

A simplification of such a network can be represented by an input layer, hidden layer and an output layer. In this context the hidden layer runs a series of computations, with the intention of serving an output that matches the desired outcome. This is achieved by training the neural network on correctly labeled data, so that it through continuous iterations of training on the data can adjust its internal weights, to provide the correct output when the model is turned towards non-labeled data [Meng et al., 2020].

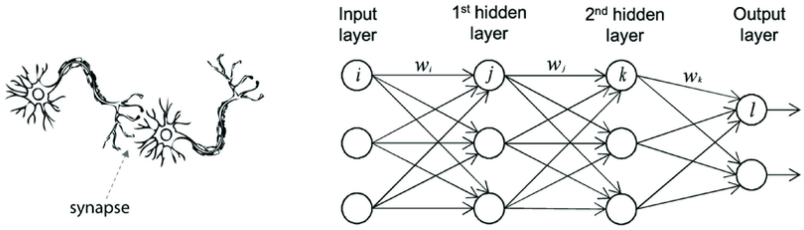

Figure 5.3. Human brain and neural network analogy [Meng et al., 2020]

5.1.2 Convolutional Neural Networks

Neural networks can be applied in a wide array of applications, which in turn has resulted in specialized neural networks to complete a specific set of challenges. Convolutional neural networks (CNN) (depicted in figure 5.4) specializes in identification of features in images, that can be used to recognize objects within them.

The input of the network is represented as a 3-dimensional matrix (Width x Height x Depth) where the depth represents the red, green and blue(RGB) color channel in an image. Considering

the amount of inputs in a neural network with a 1280*720*3 pixel image is 2764800. Additional techniques need to be applied, to be able to process such a large input.

A CNN typically consists of 4 distinct layers, the *convolutional layer*, *the pooling layer*, the *normalization layer* and the *fully connected layer*.

The convolutional layer takes an image and repeatedly applies kernels to it, which can be thought of as feature detectors. The kernel is a matrix, that has been defined when training the model. The results of these operations are the identified features in the image, also called feature maps

The pooling layer effectively shrinks its input by creating a small matrix e.g. 2x2 pixels and move it over the image with a stride of e.g. 2 px and take the maximum value of the image contained within the 2x2 matrix, shrinking the input to half its size in this case. This works in the recognitions favor, as it neglects small rotations or misplacement of features in images. After the pooling layer is applied it end up with a stack of smaller images than was put into it.

The Normalization layer applies rectified linear units (RELUS) in effort to keep the math of the network from breaking, by taking all negative values and replacing them with a 0.

The fully connected layer takes a highly reduced list of features and flattens them into a list. This list can then be processed as a neural network, where each input-field has a corresponding weight, determining the importance of each in relation to the object that wises to be identified. The fully connected layer can then be processed as a neural network as described in section 5.1.1 [Saha, 2021]

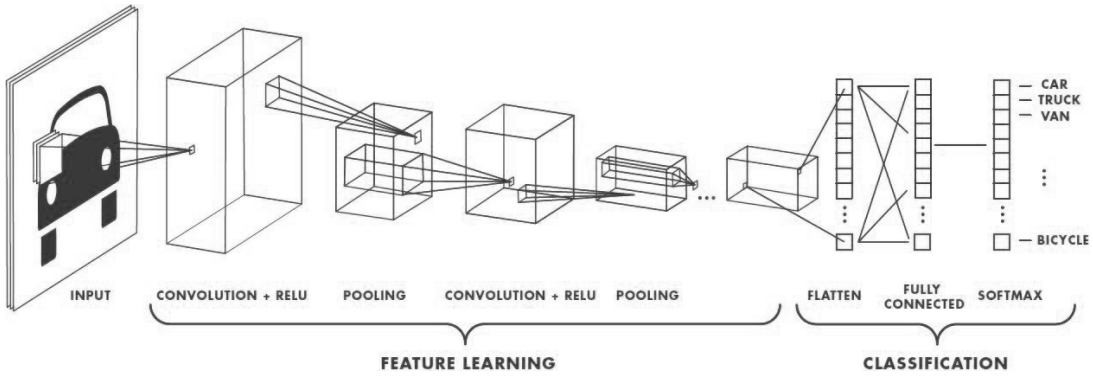

Figure 5.4. A Convolutional Neural Network [Saha, 2021]

5.1.3 You Only Look Once

This project is utilizing the fourth iteration of You only look once (YoloV4 [Bochkovskiy et al., 2020]). YOLO is a state-of-the-art object detection system, and by utilizing this thoroughly tested system for object detection the development time can be shortened considerably by not having to develop a neural network from scratch, that targets the specific use of the application in question. YOLO adds additional tools to the CNN, which are beyond the scope of this project,

but subsequently increases the speed and accuracy of the model comparatively to other models and the earlier iterations of the YOLO-models. This is visualized in figure 5.5, displaying an average precision(AP) of 43,5% which describes the correct localization and identification of objects, as well as the precision of the bounding boxes surrounding the objects identified. The figure also displays 65 frames pr. second (fps) at the given AP, which reflects the speed of the model, which in short indicates that the model operates with a high accuracy without sacrificing speed. YOLOv4 and the listed models were all tested using a Tesla V100 GPU on the MS COCO dataset [MsCOCO, 2021].

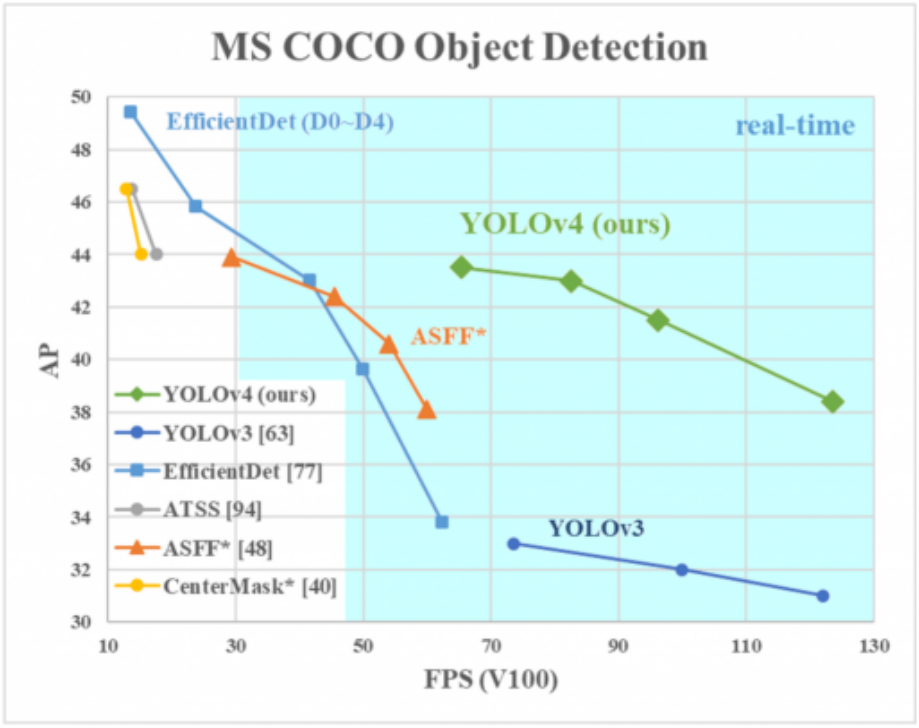

Figure 5.5. YOLOV4 Performance

In addition, YOLOv4 is highly accessible in conjuncture with python, which aligns with additional tools such as OpenCV that also will be used for extracting the license plates from the images, which will be elaborated upon later. For the YOLO-model it will be attempted to use pre-trained weights, which are made available online, thus saving the time of manually labelling a huge array of data, if the results from the pre-trained weights are found sufficiently accurate on danish number plates.

Upon setting up YOLOV4 with the pre-trained weights the results were very impressive, considering the large amount of challenges linked to unregulated inputs. In a smaller dataset of 100 cars with number plates, it correctly identified the location of the number plate in 99% of the cases. Thus showcasing a large potential for the accessibility of information in regard to developing or using CNN applications.

5.2 Image Processing

Following a successful identification of a number plate a box surrounding the object is left. From this a series of processing techniques needs to be applied in order to combat the challenges identified with number plates from a source, where the input format can not be controlled. Thus, it is attempted to remove as much noise that could hinder the optical character recognitions' efficiency, which happens in extension of this activity. The python library OpenCV [openCV.com, 2021] is used for image processing, which allows for a vast amount of built-in image operations to be used. OpenCV can often be seen used in relation to image processing and more specifically in relation to object detection models like YOLO.

Convert to grayscale

The first step in processing the image is to convert it to grayscale, this is done in the effort to get rid of the RGB channels, which in this endeavor is unnecessary values. In extension to this, previous academic research has suggested that it also can provide a higher classification accuracy as a bonus [Bui et al., 2016]. This action is performed by using openCV's cv2.cvtColor() as shown below.

```
1    gray = cv2.cvtColor(image, cv2.COLOR_RGB2GRAY)
```

Resize image

To increase the readability for the OCR, the image is resized to thrice its size. The resize operation can be seen displayed below utilizing the cv2.resize() method [OpenCV, 2021f].

```
1    resize = cv2.resize(gray, None, fx = 3, fy = 3, interpolation = cv2.INTER_CUBIC)
```

Gaussian blur

After the resizing a filter can be applied to decrease the noise in the image, which was further enhanced by the previous resize operation. A Gaussian blur is added to the previous image to decrease the Gaussian noise, which can be described as a TV-static white noise. This is done to improve the result of the upcoming edge-detection of the image [OpenCV, 2021d].

```
1        blur = cv2.GaussianBlur(resize, (5,5), 0)
```

Thresholding

At this point a large somewhat blurred number-plate is left. In this step thresholding will be applied to the image, which will give either a 0 or 255 value to each pixel, thus presenting us with a binary image. Instead of setting a global variable to decide what pixels should be white

or black, Otsu's binarization is utilized which determines it automatically [OpenCV, 2021b]. Otsu's method operates by looking at a gray-scale image, which can be presented as a histogram, displaying the occurrence of each pixel value. Otsu's method finds the optimal cut, to in this case separate the characters from the background. A display of this operation can be seen in figure 5.6.

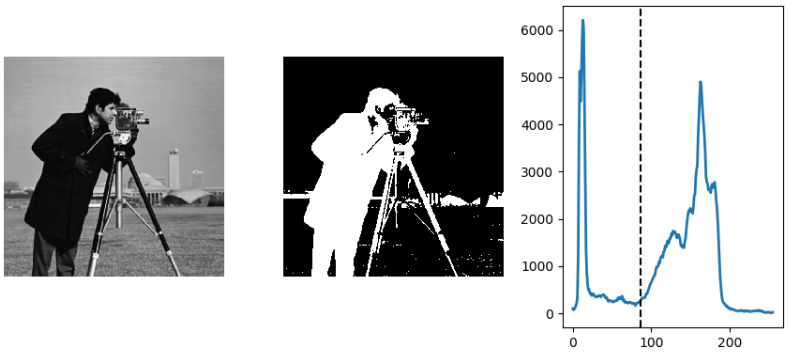

Figure 5.6. Visual representation of Otsu cut[Notes, 2021]

This method is built into openCV, only requiring the flag cv2.TRESH_OTSU to be passed, and it will generate the image in figure 5.7, using the cv2.treshold() method showcased below.

```
1        ret, thresh = cv2.threshold(gray, 0, 255, cv2.THRESH_OTSU | cv2.THRESH_BINARY_INV)
```

Figure 5.7. OTSU tresholding applied

Dilation

Dilation is a morphological operator which seeks to manipulate existing shapes in images, in this case improving the clarity of characters for the OCR[OpenCV, 2021c]. This is done by providing a kernel, which is a small matrix that moves over the image altering the pixels depending on its content. The kernel is applied to the dilation method, which emphasizes the pixels that have a 5x5 fill as seen on figure 5.8, as the result of the code below.

```
1        rect_kern = cv2.getStructuringElement(cv2.MORPH_RECT, (5,5))
2        dilation = cv2.dilate(thresh, rect_kern, iterations = 1)
```

Figure 5.8. Dilation applied

Identify contours

Using the dilated image the findcontour() method can be applied as displayed in the code below, which will apply contours to the image using the input parameter RETR_TREE which serves the contours in a hierarchy ordered tree and CHAIN_APPROX_SIMPLE as a contour approximation method that removes redundant points, such as (x, y) values spanning across the lines in the showcased contours in figure 5.9, thus saving the memory used [OpenCV, 2021a].

```
1        contours, hierarchy = cv2.findContours(dilation, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
```

Figure 5.9. Contours identified

Remove unwanted contours

In this step a general rule-set will be enforced to discount non-viable contours, that does not correspond to the sizing of characters in relation to the number plate. Chronologically going through the code below, the height and width of the characters in relation to the number plate are defined as well as checking if the total area of the contour is above 100 pixels. As seen in figure 5.10, the smaller contours are removed from the image.

```
1        height, width = image.shape
2        # if height of box is not tall enough relative to total height then skip
3        if height / float(h) > 6: continue
4        ratio = h / float(w)
5        # if height to width ratio is less than 1.5 skip
6        if ratio < 1.5: continue
7        # if width is not wide enough relative to total width then skip
8        if width / float(w) > 15: continue
9        area = h * w
10       # if area is less than 100 pixels skip
11       if area < 100: continue
```

Figure 5.10. Unwanted contours removed

Isolate characters

In the final step of image processing a bitwise mask is applied to isolate the remaining contours, to increase readability for OCR. These characters can then be used by the optical character recognition system, to extract the number plate as a text string.

```
1         # grab character region of image
2         roi = thresh[y-5:y+h+5, x-5:x+w+5]
3         # perfrom bitwise not to flip image to black text on white background
4         roi = cv2.bitwise_not(roi)
```

Figure 5.11. Characters isolated

5.3 Warped images

As previously presented, the cars scraped from DBA are unconstrained, which results in images that are not optimal for performing optical character recognition on. In a majority of published literature, number plate recognition is performed in a controlled environment, either having a fixed location in which the image is taken from (surveillance camera) or capturing the image with the intent of performing ANPR. The latter could be user-applications with a phone, where multiple frames are gathered until the phone serves a prompt when a sufficient read has been gathered.

5.3.1 Houghline

Houghline transform can be used, to identify the dominant vertical and horizontal lines in an image, from which x, y coordinates of the 4 corners containing the number plate can be identified as seen in figure 5.12[OpenCV, 2021e]. After the identification of the four corners a series of transformation operations can then be applied to correct for the distortion of the warped image.

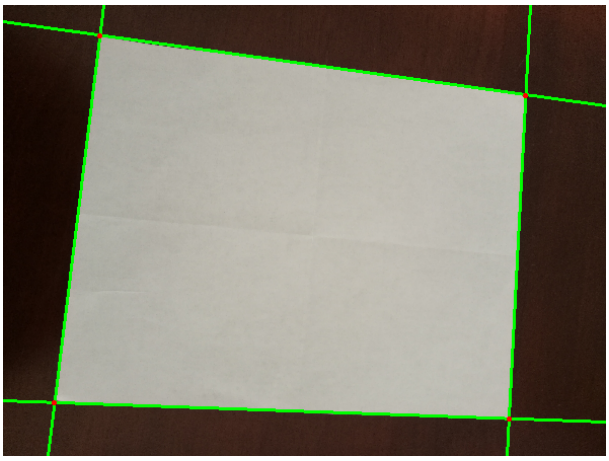

Figure 5.12. Houghline visualized on a piece of paper [Chavan, 2021]

After testing the results of the ANPR using houghline to correct for warping, it was found to be unreliable in the context of Carpair. When adding dynamic metrics for identification of dominant lines, it led to unfavorable detection and correction of the number plates at given times (see figure 5.13). This ultimately hurt the overall performance of the ANPR and was deemed non-usable at the given time, as a general rule-set for its application could not be identified.

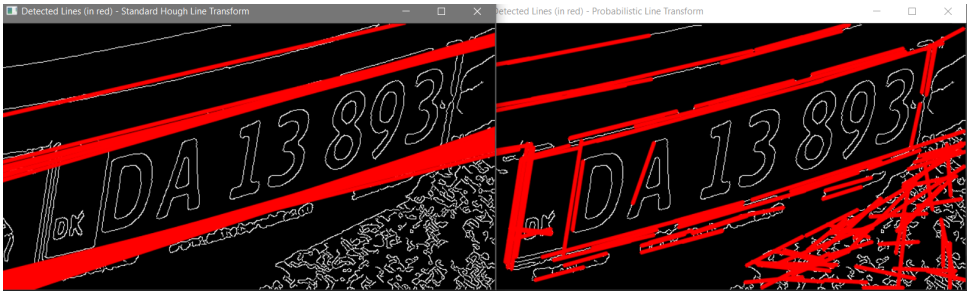

Figure 5.13. Houghline applied

5.3.2 Warped Planar Object Detection Network (WPOD-NET)

After a series of unsatisfactory results, experimenting with a range of different image processing techniques. The paper *License Plate Detection and Recognition in Unconstrained Scenarios*[Silva og Jung, 2018] was discovered, which showcase the use of *spatial transformers* to identify more precise bounding boxes of the number plates, which take oblique camera views into consideration. In the figures below the difference between a YOLOV4 and WPOD-NET bounding box can be seen.

Figure 5.14. YOLOv4 bounding box.

Figure 5.15. WPOD-NET bounding box.

A spatial transformer module is used to account for spatial invariants in the input data, and further allows for spatial manipulation within the model, as seen depicted in figure 5.16 [Jaderberg et al., 2016].

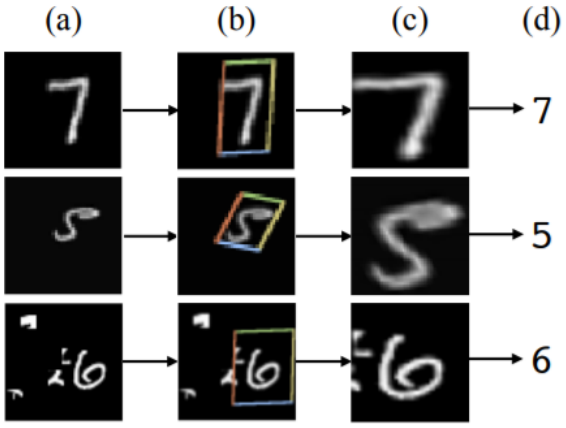

Figure 5.16. Spatial transformer operations [Jaderberg et al., 2016]

The module is added to the fully-connected network, which takes the input (a), which is distorted in relation to scale, rotation and clutter. the localization network of the spatial transformer (b) predicts the appropriate translation to do on the image. After a correct identification has been made, the operation is carried out(c), which then can be extracted by the CNN (d).

The WPOD-NET model is highly usable in the Carpair application, where a large majority of the input images are distorted. The model draws inspiration from the pre-established object detection model YOLO, which comes with a high accuracy in regard to object detection, with the addition of the aforementioned spatial transform module.

In the table below, the authors of the paper presents 5 databases of varying degree of oblique camera views. As seen in figure 5.17.

Table 1: Evaluation datasets.

Database (subset)	LP angle	Vehicle Dist.	#images	Region
OpenALPR [5] (EU)	mostly frontal	close	104	Europe
OpenALPR (BR)	mostly frontal	close	108	Brazil
SSIG (test-set)	mostly frontal	medium,far	804	Brazil
AOLP (Road Patrol)	frontal + oblique	close	611	Taiwan
Proposed (CD-HARD)	mostly oblique	close,medium,far	102	Various

Figure 5.17. Test databases.

In figure 5.18 the performance of their full ANPR-pipeline is displayed, showcasing results matching and exceeding some commercial and research systems alike.

Table 2: Full ALPR results for all 5 datasets.

	OpenALPR EU	OpenALPR BR	SSIG Test	AOLP RP	Proposed CD-HARD	Average
Ours	93.52%	91.23%	**88.56%**	**98.36%**	**75.00%**	**89.33%**
Ours (no artf.)	92.59%	88.60%	84.58%	93.29%	73.08%	86.43%
Ours (unrect.)	94.44%	90.35%	87.81%	84.61%	57.69%	82.98%
Commercial systems						
OpenALPR	**96.30%**	85.96%	87.44%	69.72%*	67.31%	81.35%
Sighthound	83.33%	**94.73%**	81.46%	83.47%	45.19%	77.64%
Amazon Rekog.	69.44%	83.33%	31.21%	68.25%	30.77%	56.60%
Literature						
Laroca et al. [17]	-	-	85.45%	-	-	-
Li et al. [18]	-	-	-	88.38%	-	-
Li et al. [19]	-	-	-	83.63%	-	-
Hsu et al. [10]	-	-	-	85.70%**	-	-

*OpenALPR struggled to understand the "Q" letter in Taiwanese LPs.

**In [10] the authors provided an estimative, and not the real evaluation.

Figure 5.18. Test results.

It is especially the results in the AOLP and Proposed CD-HARD database that can be applied in regard to the Carpair application. In those specific databases the model outperforms all the other tested systems, when focusing on distorted images presented in unconstrained input scenarios.

Based on the displayed research, the Carpair application will be utilizing WPOD-NET, to identify and unwarp images, to obtain a better performance in the full ANPR-solution. WPOD-NET will be used with a series of pre-trained weights, published by the authors of the paper.

5.4 Optical Character Recognition

In order to read the characters which was the output of the image processing MobileNet [Howard et al., 2017] is used, in unison with a set of pre-trained weights for character recognition. MobileNet is a Convolutional Neural Network, specifically aimed towards being used on mobile devices, which subsequently runs fast, without sacrificing much accuracy. This

is ideal considering the amount of calculations which needs to be made, when looping over all the individual characters identified in the image processing steps.

5.5 Number plate validation

After identifying a text string from the image containing a number plate, it is important to ensure that the string represents an actual number plate that can be linked to the correct listed car displayed on the image. If no restrictions were applied to the accepted string the system could falsely accept an invalid string. This could result in the database containing large amount of incorrect number plates for each of the car elements and some elements which in turn make the system unable to correctly use the number plate to retrieve the connected information from the MVA site which would render the system unable to function properly. This would be a substantial error and thus some measures to prevent this should be implemented. The system must therefore be able to correctly identify what a correctly number plate string is and only accept any string that follows the rules for a number plate.

The vast majority of danish number plates follows the pattern of two letters followed by 5 numbers making it a total of 7 characters. The system should therefore only accept strings identified from number plates that follow this exact pattern. Any accepted number plate can therefore be described as the string (LL-NNNNN) where L represents any letter from the Latin alphabet consisting A to Z except I and Q [Thorsen, 2021a]. Furthermore, the letter N represents any number in the range from 0 to 9 where duplicates can occur. Some exceptions to the aforementioned rules exist, such as custom number plates that could contain any combination of characters, and military number plates containing only five numbers [Thorsen, 2021b]. Although for the purpose of the system, these number plates will be ignored. This is because neither of these alternate number plates are relevant for the scope in which the system operates, being used privately owned cars listed on DBA. Both types of plates are sufficiently obscure and rare and will therefore not be expected to be relevant for the current scope of the system.

To better communicate and express the logic for accepting number plates, a visualization can be constructed. The project used the automata theory presented by Michael Sipser in *Introduction to the Theory of Computation* to express this logic [Sipser, 2021]. The format used for the logic in this section of the paper is the same as that of a push down automata in form and function (Sipser [2021] pp. 111-113). Unlike a normal push down automata which is used for implementing a context free grammar, the construction and logic behind it will only be used to describe how the application should verify number plates. It is important to note that it is only is the logic, form and functionality of the push down automata that is used in this chapter rather than the entire theory of computation. This was mainly done to visually express how the project team should understand and implement number plate validation into the developed ANPR system.

A push down automata, henceforth shortened to PDA, can formally be described by the following 7-tuple (Q, Σ , S, δ, q0, I, F) (Sipser [2021] p. 113) where:

- Q is the finite number of states
- Σ is the input alphabet
- S is stack symbols
- δ is the transition function: Q × (Σ ∪ {ϵ}) × S × Q × S*
- q0 is the initial state (q0 ∈ Q)
- I is the initial stack top symbol (I ∈ S)
- F is a set of accepting states (F ∈ Q)

The automaton used to describe the logic for selecting correct number plate strings should for any string "T" consisting of letters "L" and numbers "N", be able to identify a sub-string in T called "u" of the construction (LL-NNNNN) presuming such a string exists within string T. In the case where no such sub-string is present within string T, the automaton should recognize that no string can be accepted as a correct number plate.

The construction of a such automata can thus be seen in figure 5.19:

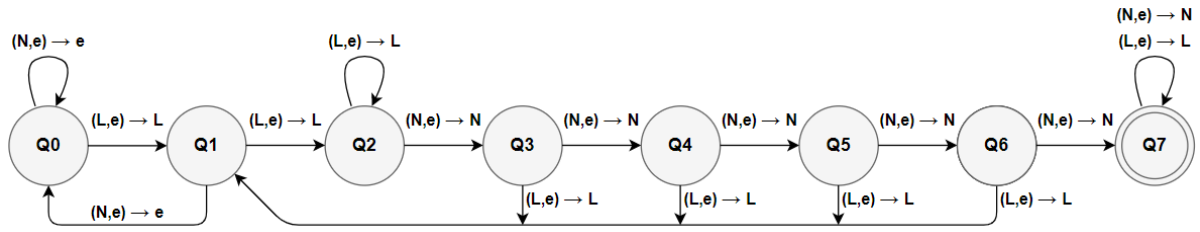

Figure 5.19. visualization of logic for number plate validation

This automation can formally be described as:

- The automaton contains 8 states
- The input alphabet for the automaton is L, N, ϵ Where L and N follows the former definition and represents an empty string.
- The stack symbols are also L, N, ϵ, where ϵ denotes nothing
- is the transition function: Q × (Σ ∪ {ϵ}) × S × Q × S* that allows for the automaton to move between states and follow the stack.
- The initial state is Q0
- The initial stack top symbol is (ϵ) meaning nothing is in the initial stack
- The set of accepting states is (Q7)

The automaton functions operates by moving through different states depending on the input string given. It always starts in the initial state, Q0 and the automaton is only accepted if it ends in the accepting state, Q7. The automaton uses an input string to transition between the

different states. For each state, the automaton will perform different actions depending on what input it reads. The automaton also incorporates a stack that can be modified when transitioning between the different states. The transitions are denoted by "(a, b) –> c" where "a" represents what input the automaton must read from the input string to perform the action, "b" is the required value the automaton must pop from the stack and "c" is what is pushed to the stack if the prior conditions are meet.

To verify that the automaton can correctly identify a number plate from any string, an example can be demonstrated of how the automaton processes a given string. For a string that is of the correct number plate construction (LL-NNNNN), for example (AB12345), the automaton would start in Q0 and would then read the letter "A". It would then pop nothing from the stack, push "A" to the stack and then move to Q1. Next it would read a "B" and proceed to Q2, popping nothing and pushing "B" onto the stack. It then reads a 1 which moves it from Q2 to Q3, again popping nothing from the stack and pushing 1. This process then repeats until Q7 where the stack will consist of the input string. While in state Q7, regardless of the rest of the input string, the automaton will remain in Q7, the accepted state, and neither push nor pop the stack. When the process is done, the correct number plate string can then be retrieved from the stack by removing the top seven elements of the stack and reversing the order of them. For the given example the stack would contain the seven elements read from top to bottom (5,4,3,2,1,B,A), and reversing it would result in the string "AB12345" which would be a legal construction for a number plate.

It is then relevant to display a case where the input string does not contain a number plate, for example the string 6CD78F90. The automaton would first read the number "6" and then loop in Q0 pushing and popping nothing on the stack. It would then read "C" move to Q1 and pushing C. It would then read and push "D" and move to Q2. it would like the previous example move on through Q3 and Q4 by reading the numbers "7" and "8". It would then read the letter "F" rather than a number which would pop nothing but push the onto the stack and move back to Q1. This allows the automaton to potentially identify a number plate hidden within a string. The stack is at this point (F,8,7,D,C) read from top to bottom. The automaton would then read the number "9" and since a number plate string has to start with two letters in a row and the two characters prior to the "9" only contains one letter, the automaton moves back to Q0. The final character "0" is then read, resulting in the automaton looping in Q0. Since the automaton failed to end in an accepting state, It can be concluded that the input string contained no correct number plate.

5.5.1 Number plate validation in code

The logic from the automaton can be directly translated into python in an effort to validate the output of the ANPR. In an effort to translate the steps the python library *re* can be used, which simplifies the logical operation of regex. The wish to only have 2 values with uppercase letters spanning from A-Z can be translated into [A-Z]2 which then can be followed by [0-9]5 that translates into 5 characters between 0-9. Combining these expressions they can be used to search for matches in the gathered data from the ANPR as showcased below.

```
1     regexVal = re.compile("[A-Z]{2}[0-9]{5}")
```

This expression will loop over the output from the ANPR, and end up with all the strings matching the regular expression. Afterwards the regex pattern can be extracted from the string, so the leading and following characters are ignored. The finalized method for validation can be seen below.

```
1  #Logic for validating carplates
2      validatedCarPlates = {}
3
4      #Two leading letters in capital + 5 numbers
5      regexVal = re.compile("[A-Z]{2}[0-9]{5}")
6
7      for carID, plateList in plateReadings.items():
8          for plate in plateList:
9              #Check min. length and if regex match is present
10             if regexVal.search(plate):
11                 #Store only the regex match and ignore leading and following charchters
12                 result = regexVal.search(plate).group(0)
13                 #Only add the first elemnent that matches the criteria (first pics in listing often has
                       better view)
14                 if carID not in validatedCarPlates:
15                     validatedCarPlates[carID] = result
```

In addition to extracting the regex pattern as seen on line 12, only the first identified regex pattern is saved when looking through multiple images from one car listing. This is done based on the logic that the number plates identified first from multiple car images is often the correct one as car listings on DBA often showcase more detailed exterior images first, thus ignoring the later readings which could be more prone to error.

5.6 Evaluation

In this section the overall functional performance of the ANPR will be discussed, in the context of the Carpair application. The test-set will be composed of all car listings in Aalborg on DBA, which will not suffice for providing exact performance metrics for the model itself. In contrast, it will provide indicative result of how a model only using pre-trained weights, not having undergone any additional training on danish number plates can perform in the context of the Carpair application.

The test-set does not include car listings containing either:

- No images.
- No number plates.
- Commercial number plates.
- Specialized number plates(custom or military).

After scraping all privately listed cars on DBA from Aalborg, 80 cars were returned but included: 3 listings with no images, 2 listings with commercial plates, 1 listing with a specialized number plate and 23 listings without number plates. This makes the refined test-set consist of 51 cars, which could allude to having picked a too narrow scope, and the speculative nature of the results gathered. This will be looked past in this section and hopefully still show the indicative potential of using the different methodologies listed.

The tests performed will utilize the validation techniques described in the previous section. Such that the ANPR, will look through multiple images in one car listing and then clean up the output using regular expressions. In the case of a number plate with 8 characters is identified containing AAB10293, the validation method correctly will identify the pattern LL-NNNNN, providing us with the correctly identified plate AB10293.

After running the ANPR pipeline on the described test-set it yielded a result of 76%, correctly identifying the number plate in 39 listings out of 51. While the result should not be taken as fact, due to the small test-set, it still is indicative of the performance. The results are considered satisfactory for the time being considering that only pre-trained weights have been used for training the models. This could allude to a greater performance if labeled datasets of danish number plates were used for training the models in future iterations.

Website implementation 6

In the following chapter a general overview of the system will be showcased, presenting the website in which the data collected by the back-end services are displayed and further processed. Firstly showing the architecture and handling of requests from the back-end. Afterwards the filtering of user inputs into queries that are sent to the back-end, serving data to the application. Lastly the recommendation system, showcasing cars which could be of specific interest to the user.

6.1 Architecture

The overall architecture of the implementation of the platform is based on the back-end sending data to the front-end that transform the data into HTML elements on the web page. The data is sent from the back-end as JSON strings and converted to JSON objects in the front-end. JSON is used as the data that the back-end exchanges with the front-end, can only be text [w3schools.com, 2021]. JSON is therefore benificial as this allows for objects to be represented as text. An example of a JSON string is shown below:

```
'{"name":"John", "age":31, "city":"New York"}'
```

The data is retrieved from the back-end in two ways: with the page requests or with XMLHttpRequests.

6.1.1 Page requests

The data can be sent to the front-end with the page request. An example of this can be seen in listing 6.1

```
// Display car listings dependendt on user input from form
app.get('/results/getFiltered', function (req, res) {
    [...]
    // Execute query and render the results on the page
    handleSql(query, "return lots", function (result) {
        var string = JSON.stringify(result);
        res.render(path + 'results/results.html', { results: string });
    });
});
```

Listing 6.1. Results endpoint from app.js

The function executes code that generates data in JSON format. The JSON object is then transformed to a simple string that is returned with the web page (lines 6-7 in listing 6.1). The data is then inserted into the HTML page, as seen in listing 6.2.

```
1 <script>loadCars('<%- results %>');</script>
```

Listing 6.2. Inserting back-end data in front-end (results.html)

As the listing shows, the JSON string is inserted as a parameter in the *loadCars* function. This function is shown in listing 6.3.

```
1 function loadCars(jsonStr) {
2     var result = JSON.parse(jsonStr);
3 
4                 for (var i = 0; i < result.length; i++) {
5                         var row = createDomRow(result, i);
6                         resultp.innerHTML += row;
7                 }
8 }
```

Listing 6.3. loadCars function from results.js

As the listing shows, the JSON string is transformed into a JSON object. For each element in the JSON object a new row is inserted into the website (lines 4-7). Figure 6.1 shows the web page when three rows are inserted.

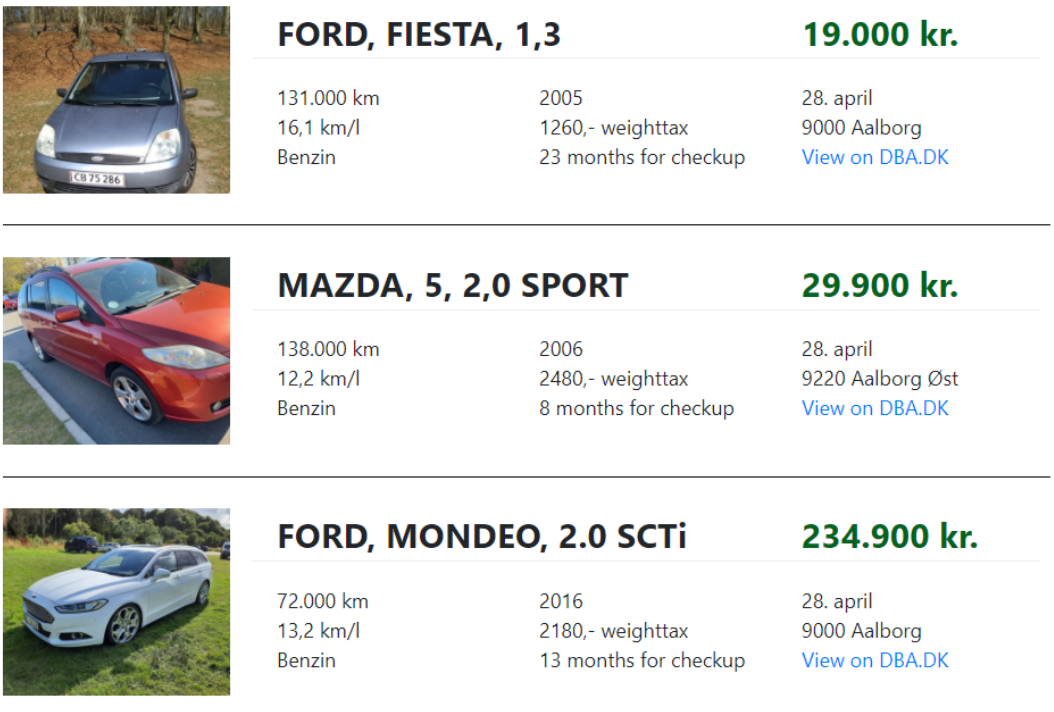

Figure 6.1. Three rows inserted from *loadCars*

The function that generates the rows is shown in listing 6.4.

```
1     function createDomRow(json, i) {
2         var bgColor = (i % 2) ? "#ffffff" : "#ffffff";
3         var row = `<div class="h150container" style="background-color: ${bgColor}">
4             [...]
5                 <div class="row details">
6                         <div class="col-sm-4">
7                                 ${json[i]["kilometer"]}.000 km
8                         </div>
9                         <div class="col-sm-4">
10                                ${json[i]["manufactured"]}
11                        </div>
12                        <div class="col-sm-4">
13                                ${json[i]["listingdate"]}
14                        </div>
15                </div>
16                <div class="row details">
17                        [...]
18                </div>
19                <div class="row details">
20                        [...]
21                </div>
22                </div>
23
24        </div></div>`;
25
26        return row;
27  }
```

Listing 6.4. *createDomRow* function in results.js

This function takes a JSON object and a counter as parameters and returns a div string with the selected JSON data inserted. Most of the data is inserted in the *child divs* with a class of *row details*. These rows store the kilometers, manufacturing age, listing date, km/l, weight tax, zip code, fuel, months until next check up and a link to the listing on dba.dk. This parent div (lines 3-24) is then appended to the inner HTML of the container for the car-rows and then transformed into an actual DIV element on the web page.

6.1.2 XMLhttpRequests

The other method that is used for connecting the front-end and back-end is XMLhttpRequests. The difference between this method and the previously described method is that this method is done from the front-end of the application, whereas the other was done from the back-end. The XMLhttpRequests enables the application to update the web page even after the page has loaded and without refreshing the web page. An example of this is seen in listing 6.5.

```
1  function loadCarsCount() {
2         var xmlhttp = new XMLHttpRequest();
3         xmlhttp.onreadystatechange = function () {
4                 if (this.readyState == 4 && this.status == 200) {
5                         var result = JSON.parse(this.response)[0]["count(price)"];
6                         createPageButtons(result);
7                 }
8         };
9         xmlhttp.open("GET", `getCarsCount`, true);
10        xmlhttp.send();
11 }
```

Listing 6.5. loadCarsCount from results.js

The function starts by creating a new XMLHttpRequest (line 2). When the data is received, the function then transforms a JSON string into a JSON object and passes that object as a parameter for the *createPageButtons* function (lines 3-8). Afterwards the function simply selects the endpoint that should be called and sends the request. The back-end then handles the requests and returns the specified data. The *createPageButtons* then insert page buttons on the web page, which enables the user to browse through all the cars in the database (lines 5-6).

6.2 Creating result pages

The database contains a significant amount of data, so it would be overwhelming to display all of it at once for the user. For this reason, the application is split into pages, where each page displays 10 cars. This makes the platform a lot simpler for the user. The data for the page buttons is retrieved as JSON strings as described in the prior section. The code for inserting the page buttons on the website is seen below:

```
1  function createPageButtons(carsCount) {
2         var div = document.getElementById("pages");
3         var buttons = Math.ceil(carsCount / 3); //3 because we want to show 3 cars per page.
4         var pagetmp = getParam("page");
5         var page = (pagetmp != null) ? pagetmp : 1;
6
7         for (var i = 0; i < buttons; i++) {
8                 var active = (page - 1 == i) ? "class='active' " : "";
9                 var button = `<button ${active}onclick='changePage(${i+1})' type='button'>Page
                        ${i+1}</button> `;
10                div.innerHTML += button;
11        }
12 }
```

Listing 6.6. createPageButtons from results.js

The number of buttons that are generated is the ceiling of *number of cars* divided by three (line 3). Then the HTML code for the buttons is created and appended to the chosen DIV element on the website (lines 7-11). The page button for the current page is given a class of *active*, as it should be displayed in different colors in order to enhance the visibility of the system status (line 8). When a button is clicked, the *page* parameter that is passed to the

web page, is changed in the URL. This is seen in the *onclick* attribute for the created buttons: *"onclick='changePage($i+1)'"*.

The cars that are shown are then calculated by the back-end with the page request based on the *page* URL parameter:

```
1     var page = req.query.page;
2     var offsetStr = "";
3     if (!isNaN(page)) {
4         var offset = (page - 1) * carsPerPage;
5         offsetStr = ` OFFSET ${offset}`;
6     }
```

Listing 6.7. page number and SQL offset (app.js)

The code generates the offset that is needed for the SQL query. The offset is calculated on line 4. CarsPerPage is set to 3, as three cars should be displayed per page. For example: If the page number is 4, then the offset is (4 - 1) * 3 = 9. The SQL query that retrieves the cars that is displayed on the web page would then end with "LIMIT 3 OFFSET 9;".

6.3 Car filtering

In order for the user to filter car queries, a HTML-form is presented on the front-page of the application, where users can select as many filters to query the data on as they desire. This front page can be seen displayed in figure 6.2.

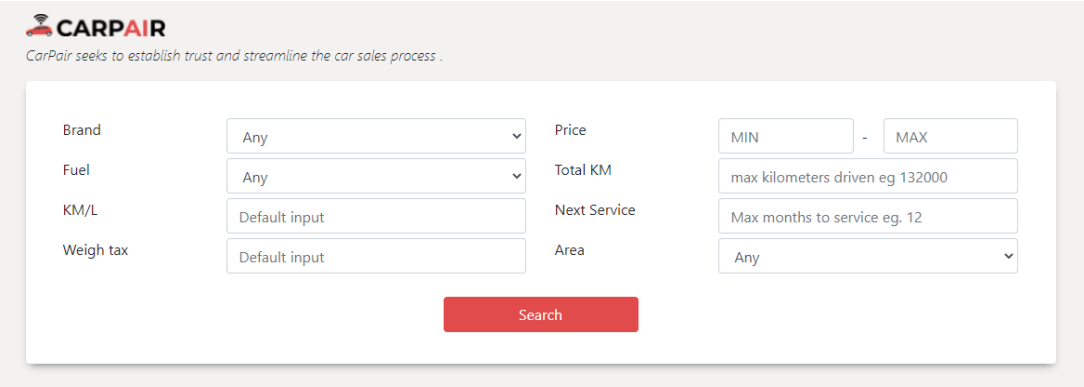

Figure 6.2. Html-form presented on the index page

The entered user-input is then handled with a GET request and routed with node.js to /getFiltered, where the user-input is stored as values in a Javascript dictionary, corresponding to the input-fields entered, as seen in the code below.

```
1     var carFormInput = {
2         weighttax: req.query.weighttax,
3         brand: req.query.brand,
4         fuel: req.query.fuel,
5         kml: req.query.kml,
6         kilometer: req.query.kilometer,
7         area: req.query.area,
8         priceMIN: req.query.priceMIN,
9         priceMAX: req.query.priceMAX,
10        area: req.query.area,
11        nextService: req.query.nextService,
12    };
```

The initial idea was to create a generic query, which just had a large SELECT WHERE statement, whereby looping through the dictionary, the element specifying WHERE would be the key of the dictionary and the clause would be the value of the key-value pair as described in the pseudo-code below. To also accept non-entries the value would be replaced with the SQL wildcard "LIKE % " To accept all possible values, if nothing was specified in the HTML forms input field.

```
1  "SELECT * FROM scrapedCars INNER JOIN carData ON scrapedCars.numberplate = carData.numberplate"
2  for i in carformINput:
3      "WHERE" + carFormInput.key + "=" carFormInput.value + ";"
```

Upon further inspection it was discovered that it almost never was desired to return the same exact value which was entered in the HTML-form annotated by "=" in the statement above, which meant that the query had to be dynamic in nature and should be formed differently depending on which field the input was gathered from.

To allow for a more dynamic query, the different natures of the inputs needed to be considered. In the first if statement seen below specific default values needed to be specified where "LIKE % " would not return a successful query, such as in priceMIN and priceMAX on line 5-8 where an arbitrary high and low value was entered instead. The else statement handles successful queries where different operators was used, such as weight tax on line 19-20 to show all car-listings where the number was higher than the one entered. The explanation for choosing the different logical operators can also be seen commented out in the code below.

```
1     for (var key in carFormInput) {
2         value = carFormInput[key];
3         // add LIKE % to non-entries and set specific values for priceMIN,MAX as they need a value not
              LIKE%
4         if (value === "" || value === undefined || value == "any") {
5             if (key === "priceMIN") {
6                 cleanFormData["priceMIN"] = 1;
7             } else if (key === "priceMAX") {
8                 cleanFormData["priceMAX"] = 90000000;
9             } else if (key === "nextService") {
10                cleanFormData["nextService"] = 1000;
11            } else {
12                cleanFormData[key] = " LIKE '%'";
13            }
14        }
15        else { // Add operators in certain statements
16            if (key === "kilometer") { // Show numbers lower than entered
17                value = (value) / 1000;
18                cleanFormData[key] = " < " + value;
19            } else if (key === "weighttax") { // Show numbers higher than entered
20                cleanFormData[key] = " < " + value;
21            } else if (key === "kml") { // Show numbers higher than entered
22                cleanFormData[key] = " > " + value;
23            } else {
24                cleanFormData[key] = "= " + value; // we want the exact number (selectables)
25            }
26        }
27    }
```

After the data has been cleaned to the desired specification, it is then interpolated as strings in the SQL statement shown below. The only addition to this being the inserted BETWEEN clause between priceMIN and priceMAX on line 8 and further calculating the months from the checkup date as seen on line 10.

```
1   // Insert SQL-esque dictionary into master query (could be dynamically generated)
2     var query =
3         'SELECT * FROM scrapedCars INNER JOIN carData ON scrapedCars.numberplate = carData.numberplate ' +
4         'WHERE carData.weighttax' + cleanFormData.weighttax +
5         ' AND carData.fuel' + cleanFormData.fuel +
6         ' AND carData.kml' + cleanFormData.kml +
7         ' AND carData.kilometer' + cleanFormData.kilometer +
8         ' AND scrapedCars.price' + " BETWEEN " + cleanFormData.priceMIN + " AND " + cleanFormData.priceMAX
              +
9         ' AND scrapedCars.area' + cleanFormData.area +
10        " AND STR_TO_DATE(carData.checkupdate, '%d-%m-%Y') BETWEEN CURDATE() AND DATE_ADD(CURDATE(),
              INTERVAL +" + cleanFormData.nextService + " MONTH)" +
11        ' AND carData.brand' + cleanFormData.brand + ` LIMIT ${carsPerPage}${offsetStr};`;
```

Following the creation of the query, the pre-written handleSQL() method as seen in the code below, can be utilized. This connects and handles SQL requests to the database, where the query is then sent and the results of the query is returned and sent to the result-page.

```
1       // Execute query and render the results on the page
2     handleSql(query, "return lots", function (result) {
3         var string = JSON.stringify(result);
4         res.render(path + 'results/results.html', { results: string });
5     });
```

The result of the query is showcased on the result-page as seen in figure 6.3, which displays the car-listings corresponding to the desired inputs, that the user entered in the HTML form.

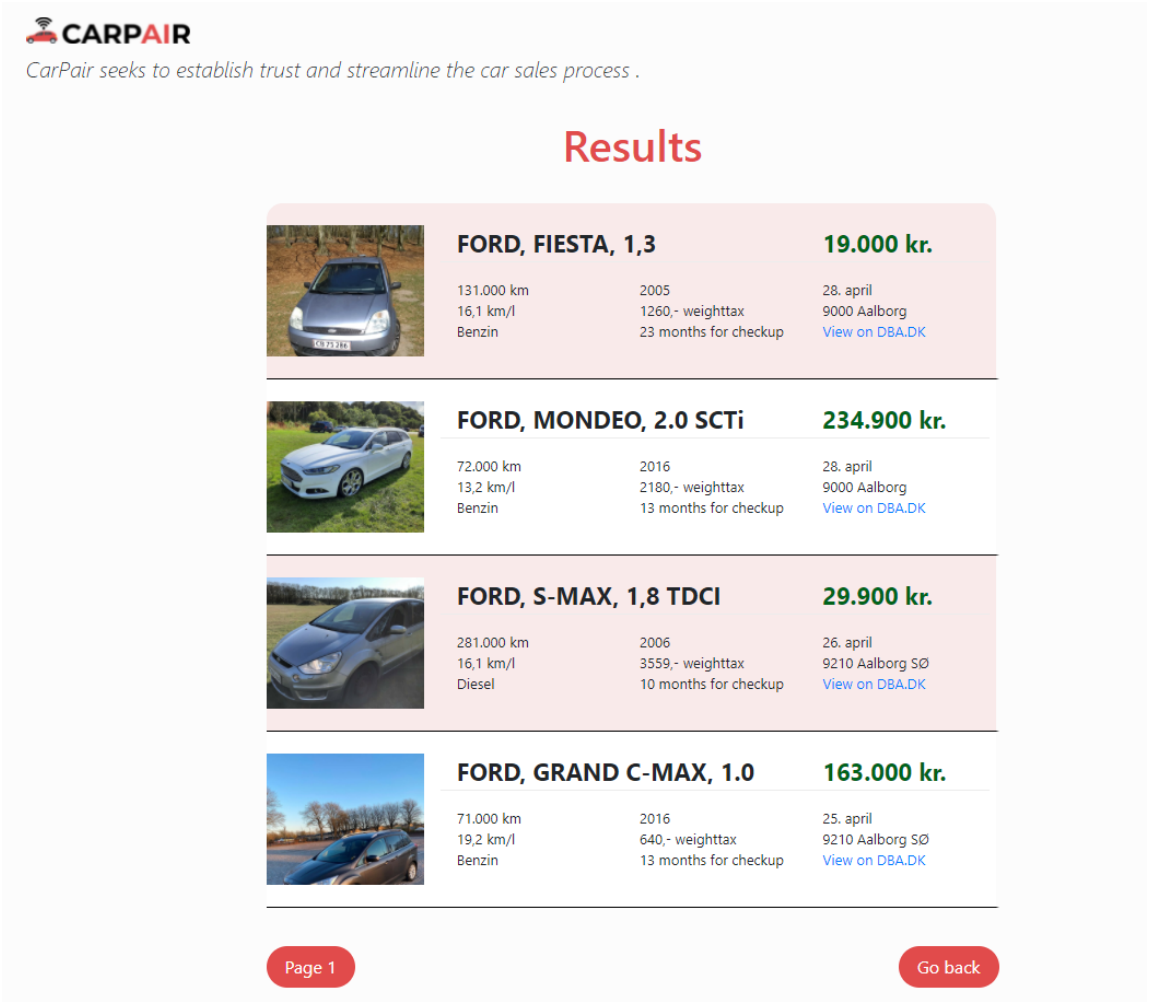

Figure 6.3. Result-page which displays car listings matching the user-input

6.4 Recommendation system

The front page of the website should show some recommended cars in addition to the filter-options. This would be a feature for the users, which would make it easier for users to find a desirable car, and it would provide more content for the front page. However, the desirability of a car is quite subjective as it depends on the users needs. The main advantage of using this website instead of dba.dk, is that this website always provides information and filters about economic data-points such as kilometers driven, fuel efficiency, taxing, and months until the next checkup. For this reason the advantage of this website boils down to making it easier to find an economical car. This should therefore be reflected in the recommended cars displayed to the user. So the recommended cars should display cars that are desirable from an economic standpoint.

6.4.1 Approach

It is quite difficult to find the most economical cars as it depends on a large number of factors, such as:

1. The specific user needs
    - How many kilometers does the user expect to drive per year?
    - Does the user have any technical knowledge? I.e. can the user do some maintenance themselves or do they require a mechanic to do everything?
    - How important is safety? Is ABS or ESP a requirement?
    - Are there any requirements for features such as cruise control or air conditioning?
2. The car model
    - Does the car model tend to last long?
    - Does the car model tend to have a lot of issues relative to other models?
    - Safety, appearance, power, and so on.
3. The specific car
    - What has been newly changed or repaired on the car? E.g. new timing belt, brakes, radiator, or motor and so on.
    - What does the car look like? Is it rusty? Has it had any undercarriage treatments? Does the paint still look good? Are there any scratches or dents?
    - Does the car run well? Has it lost power? Is the gear shift working smoothly? Has it lost fuel efficiency?
    - Is there anything leaking from the car?

Finding the most desirable car is therefore extremely difficult and could be the goal for an entirely separate scientific paper in itself. For this reason, the algorithm for finding the recommended cars will restrain from using complex implementations, reflected in state-of-the-art recommendation systems. The approach for the recommended-cars algorithm will be based on the data in the database that is generated by the previously described components: web scraping and ANPR. The data is thus generated by web scraping dba.dk for privately sold cars in Aalborg. Then using ANPR to find and extract the number plates. Then using another web scraper for MVA that gathers the number plate data.

The approach for generating the algorithm is to analyze data points for the cars and find possible correlations. To simplify the data, the data points will be split into three categories: Price, age and expected yearly expenses. The price and age category is easy to gather from the cars, but the expected yearly cost is more difficult and will be based on rough estimations.

The expected yearly expenses will be calculated by the sum of three kinds of expenses, namely fuel expenses, maintenance expenses, and tax expenses. The calculation for each expense is as follows:

1. Fuel expenses

- The average kilometers driven per year in Denmark is approximately 16.000 [Lemche, 2018][Lund, N/A][DTU, 2014].
- The average liter price for fuel is approximately 11 kr. per liter [FuelFinder, 2021].
- Therefore, the yearly fuel expenses is calculated as 16000/kml*11.

2. Maintenance expenses
   - The maintenance expenses are very difficult to estimate. For this reason the estimation will be based on a source from the development manager at AutoMester. The following table is based on his estimations [Foget, 2019]:

Kilometers driven	Estimated yearly maintenance expenses
<60.000	3500 kr.
<120.000	11000 kr.
>120.000	12.000 kr.

Table 6.1. Maintenance expense vs. kilometers driven

3. Tax expenses
   - The yearly tax expenses is intuitively calculated as the tax is paid twice per year. The yearly tax is therefore simply: tax*2.

Now there are three categories of car data: price, age, and yearly expenses. The next step is to find possible correlations for price/age and price/yearly expenses.

6.4.2 Score for price/age

The correlations are analyzed by importing the data in Microsoft Excel as seen in figure 6.4.

	A	B	C
1	Price	Age	EstYearlyExpense
2	7000	17	23291
3	10000	19	25814
4	13000	15	20033
5	13000	15	20033
6	13000	15	20033
7	13000	15	20033
8	15500	21	29058
9	16000	17	29160
10	17500	12	24087
11	19000	16	24458
12	22000	13	28527
13	29900	15	29056
14	29900	15	30075
15	32000	14	28086
16	33999	9	18217
17	35000	11	23087
18	38000	14	25368
19	54000	9	20743
20	59800	7	21589
21	78000	18	28763
22	83000	6	17501
23	127500	6	18383
24	163000	5	20613
25	199900	9	28860
26	234900	5	27481
27	347000	5	21294

Figure 6.4. Database data imported in Excel

Then the price and age are inserted into a scatter plot and a trend line is added. The result is seen in figure 6.5.

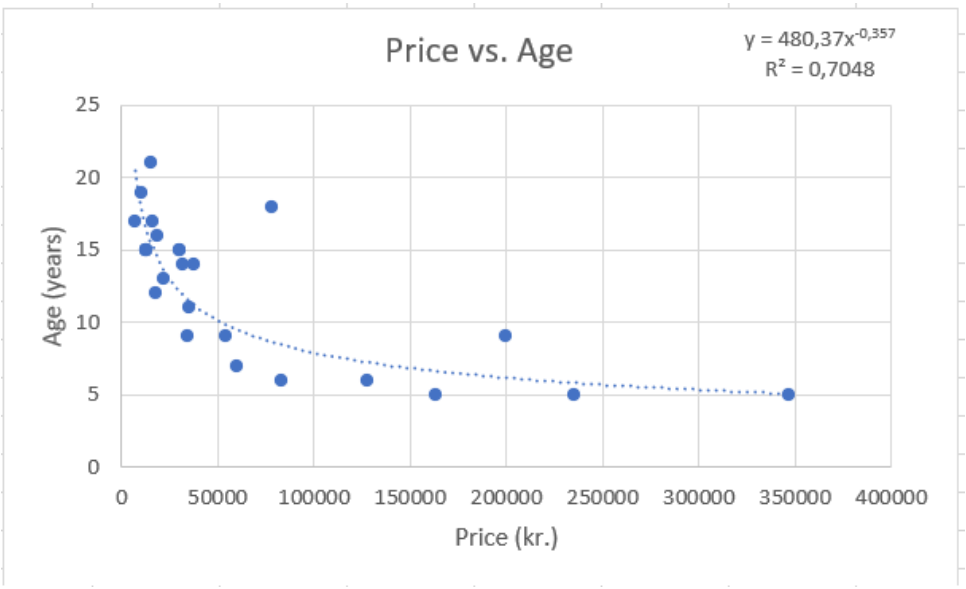

Figure 6.5. Price/age correlation

Figure 6.5 shows, that the correlation is moderately strong as the correlation coefficient (R^2) value is 0,7 [Brewer, 2001]. Excel generates a formula for the correlation which is seen in the top

left corner of figure 6.5. This formula can be used to calculate an approximation of the current age of a car based on the price of the car.

For example: If a car costs 17.000 kr. then the expected age is $480{,}37*17.000^{-0{,}357} = 14{,}8$ years. Then an index number is calculated by the age of the car divided by the expected age times 100.

For example: If the car is 16 years old then the index will be: $16/14{,}8*100 = 108$. This index is a score for the age of the car, where a score of 100 is neutral. A low score is good, and a high score is bad. This score will be used later, but first a score for price/expected yearly expenses is needed.

6.4.3 Score for price/expenses

The correlation between price and expected yearly expenses can be seen in figure 6.6.

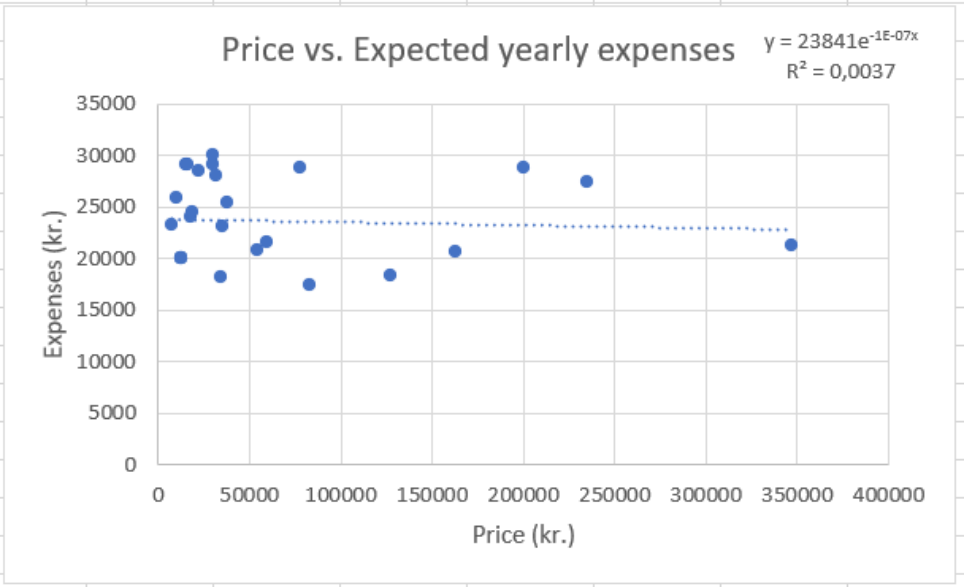

Figure 6.6. Price/expense correlation

The trend line and correlation coefficient in figure 6.6 shows that there is almost no correlation between price and expected yearly expenses. The correlation is expected, because expensive cars can also have a high expected yearly expense; the km/l is mostly determined by the motor size of the car [Fraser, 2020]. A lower motor size is more fuel efficient, and the weight tax is based on the kilometers per liter [skat.dk, 2021].

When the correlation coefficient is almost 0, the formula is not valid. Therefore, another option is needed. The chosen approach for this is to calculate the estimated expenses divided by the average estimated expenses for the cars, which is about 24.000. For example: If a car have a yearly average expense of 30.000 then the index will be $30.000/24.000*100 = 125$. A low score is good, and a high score is bad. A score of 100 is still neutral.

6.4.4 Overall score

Now the index for age/price and expenses/price can be calculated. Lastly the score will be an average of the two. So if a car has an age score of 110 and an expense score of 100, then the overall score will be 105.

But this formula does not particularly account for the actual price of the car, so the expensive cars are predispositioned to have a better score. This is accommodated by implementing a price-handicap in the formula. The handicap is calculated by dividing the price with 500.000, in order to achieve a fair balance for the handicap. We chose the number of 500.000 in order to achieve a fair balance between score and price. Then the final score is achieved by the average score times one plus the handicap. It is multiplied by one plus the handicap as the handicap should correlate the score and not change it completely. For example: If the average score is 105 for a car and the price is 17.000 then the handicap will be: 17.000/500000 = 0,034. And the overall score is 105 * (1 + 0,034) = 109.

The overall score is calculated and stored in the database when MVA is scraped. This makes it easy to show the 3 most economical cars, as it can be done with a simple SQL query:

```
1        SELECT  * FROM scrapedCars INNER JOIN carData ON scrapedCars.numberplate = carData.numberplate
             ORDER BY score LIMIT 3;
```

The result of this formula is that the smaller economical cars are prioritized. This can also be seen in the actual implementation on the website in figure 6.7.

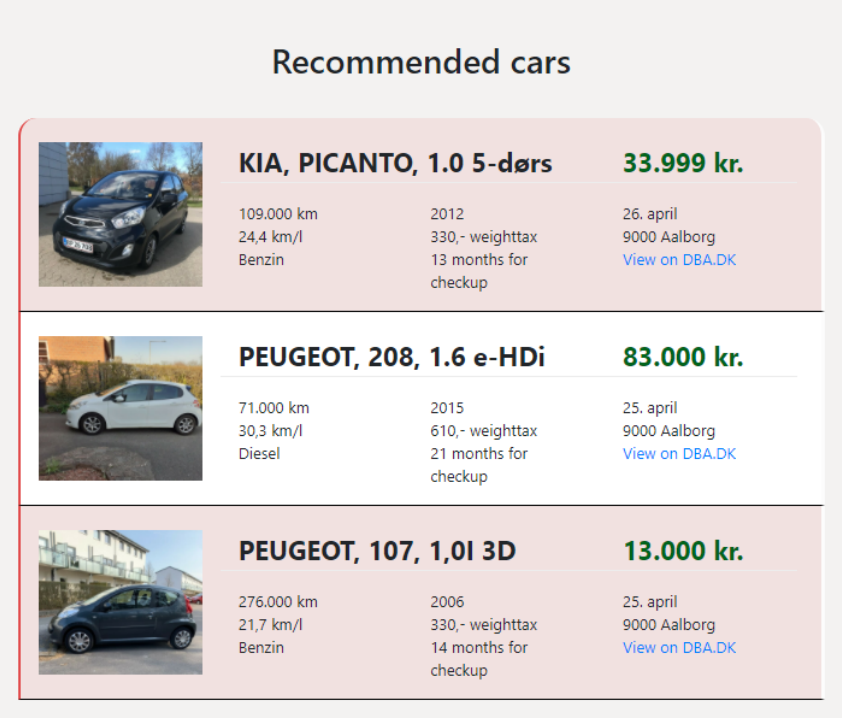

Figure 6.7. Recommended cars

The final algorithm is shown in figure 6.8.

$$\frac{\left(\left(\frac{\text{carAge}}{480{,}37 * \text{carPrice}^{-0{,}357}} * 100\right) + \left(\frac{carExpense}{24.000} * 100\right)\right)}{2 * (1 + (\frac{carPrice}{500.000}))}$$

Figure 6.8. Score algorithm

An example with a car that is 12 years old, costs 18.000 per year in expenses and have a price of 21.000 is shown in figure 6.9.

$$\frac{\left(\left(\frac{12}{480{,}37 * 21.000^{-0{,}357}} * 100\right) + \left(\frac{18.000}{24.000} * 100\right)\right)}{2 * (1 + (\frac{21.000}{500.000}))} = 78$$

Figure 6.9. Score algorithm example

Sprint Summary 7

This chapter will document the outcome of each of the sprints conducted throughout the project. For each sprint the goal will be presented as well as whether the goal was reached and what the project group learned from the sprint when going into the planning for the next sprint.

7.1 The first sprint

The first sprint had the goal of making the individual components function as outlined in the application design. One area of interest was to investigate how and if the ideas were reachable.

One of the items in the first sprint was to create the web scraper. to collect specific data from the car listings on dba.dk within Aalborg. The initial goal was for the web scraper to collect the correct data and correctly put it into the database of the system. one concern was the efficiency of the web scraper which could run slower than what was desired, the improvements on the component would be a continuous focus throughout future sprints.

Another item in the backlog was the creation of a component to recognize number plates from an image. The ambition was to make the component able to pull a number plate string from an image, without any regard to how accurate it was. Presuming that the component was able to work with some images, the accuracy would be improved further upon in later sprints.

The last item of the first sprint was to plan out the initial design of the Website, deciding how the content on the page should be presented and what information would be contained on the page.

Sprint outcome

The outcome of the first sprint is described below, showcasing images and text that shows the progress and future work planned in regard to each component.

The web scraper correctly collected the prices, thumbnail links and the links of the cars from the DBA listing as seen in figure 4.2 in the aforementioned chapter. This is described in greater detail in the previous chapter 4 Data gathering. This result was a success as it was able to do what was intended. The efficiency of the ANPR in the first sprint, was around 17%, which will be sought to be improved upon in the following sprints aiming for at least a 50% efficiency in the following sprints, ensuring that it could work as a proof of concept.

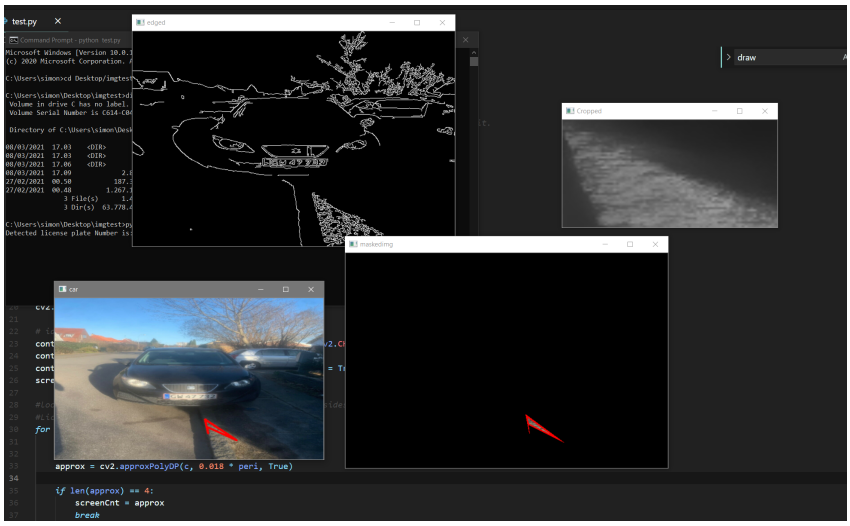

Figure 7.1. First results of the ANPR

The first draft of the website was primarily created to get an insight into the basic functionality and composition of the website, see figure 7.2. JavaScript was not used in the first iteration, as the primary focus on setting up the was not on the intricate interactions between the elements but creating the overarching structure and layout of the application.

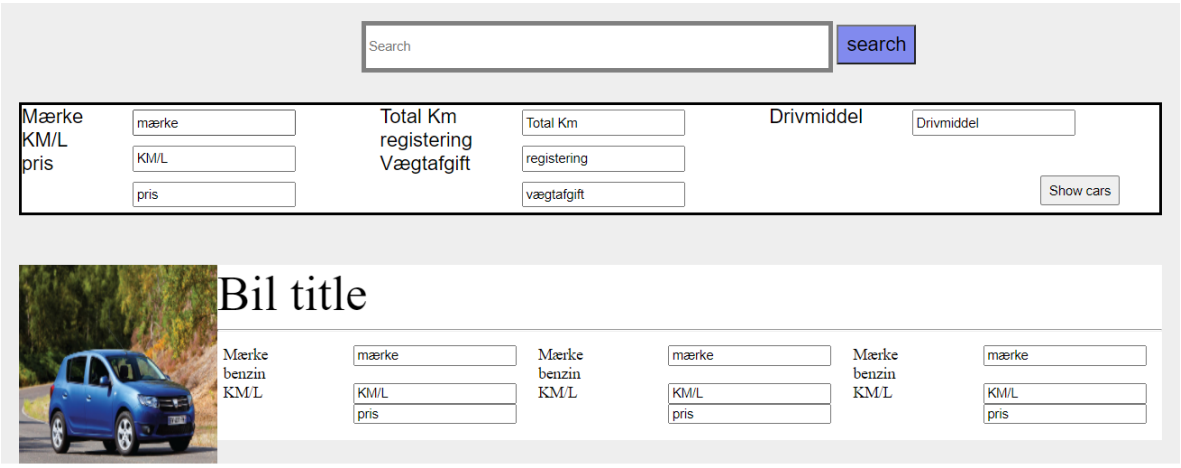

Figure 7.2. Early draft for the website design

The project group evaluated that the first sprint was an overall success, the goals which were set was sufficiently reached. A series of improvements would have to be made both to the accuracy of the ANPR and the efficiency of the web scraper. It was decided that the second sprint should continue to improve the current components to a state where they could be combined to work together with the central database.

7.2 The second sprint

The focus of the second sprint was to ensure that the components correctly worked together. Additional improvements to the components were still a continuous focus for this sprint as well. The goal was to connect the website, ANPR and web scraper, so they were able to transfer data between each other. A solution to the problem of gathering the necessary data from the MVA was also an item that would have to be solved in this sprint.

Sprint outcome

The results from the second sprint showed great promise. The intended functionality of the website was finalized, making the website correctly navigate to the results page where the cars were displayed, see figure 7.3.

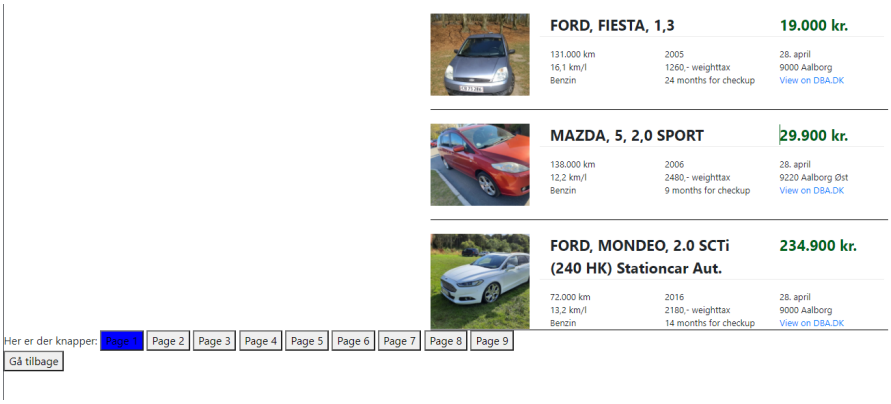

Figure 7.3. Example of results page on website

The visual design were likewise updated to be cleaner and more sharp looking, displaying the filter options more clearly, see figure 7.4. The placement of the recommendation system was also made within this sprint, although the functionality was not finished. A decision was made to push this item to the next sprint as the recommendation system was less crucial for the overall functionality of the application.

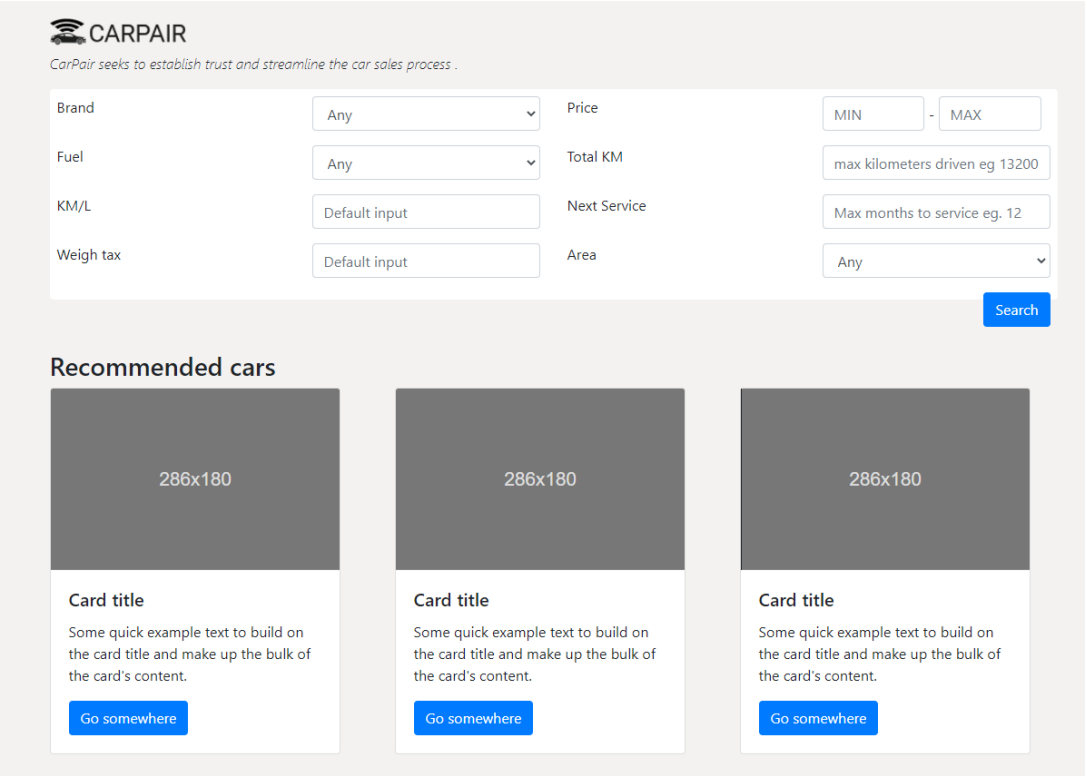

Figure 7.4. Updated Visual design for website

An issue that was encountered in this sprint was how to access the data from the MVA website and get it into the database for the website to display. The intention was to use an API that was available from MVA, but due to complications regarding gaining access to the API and time limitations, the project group decided to find another solution, which ended up being to use another web scraper. The issue with this solution was the amount of time the application would need to use to run both web scrapers. The primary focus at this stage was to find a way to make sure that the application worked, regardless of efficiency. Improvement of the efficiency of the web scraper was something that would be carried through to later sprints.

At the end of the second sprint, the application reached a functional state where the components were connected, as was the goal of the sprint. Some new challenges were encountered, mainly the issue with the MVA API, but this would be taken into consideration in the next sprint, which would primarily focus on finishing the system functionality, website and the performance of the web scrapers. The recommendation system was another important item for the next sprint that should be completed.

7.3 The third sprint

The goal of the third sprint was to finish the remaining items related to the application including the completion of the recommendation system. The tasks in this sprint was finishing the website, implementing the recommendation system and continuing to improve on the components, especially the accuracy of the ANPR.

Sprint outcome

The first step in the development of the recommendation system was to decide how the system should decide what to show. The project group ended on a solution where each car in the database would get a score based on the economic value of the car. This was based on the price of the car and the yearly expenditure as well as age.

The recommendation system could be designed to take other factors into account, but the project group decided that the economic factors were the most important, as that is essential for buying used cars. The recommendation system was placed on the front page of the website displaying the top three cars as can be seen in figure 7.5.

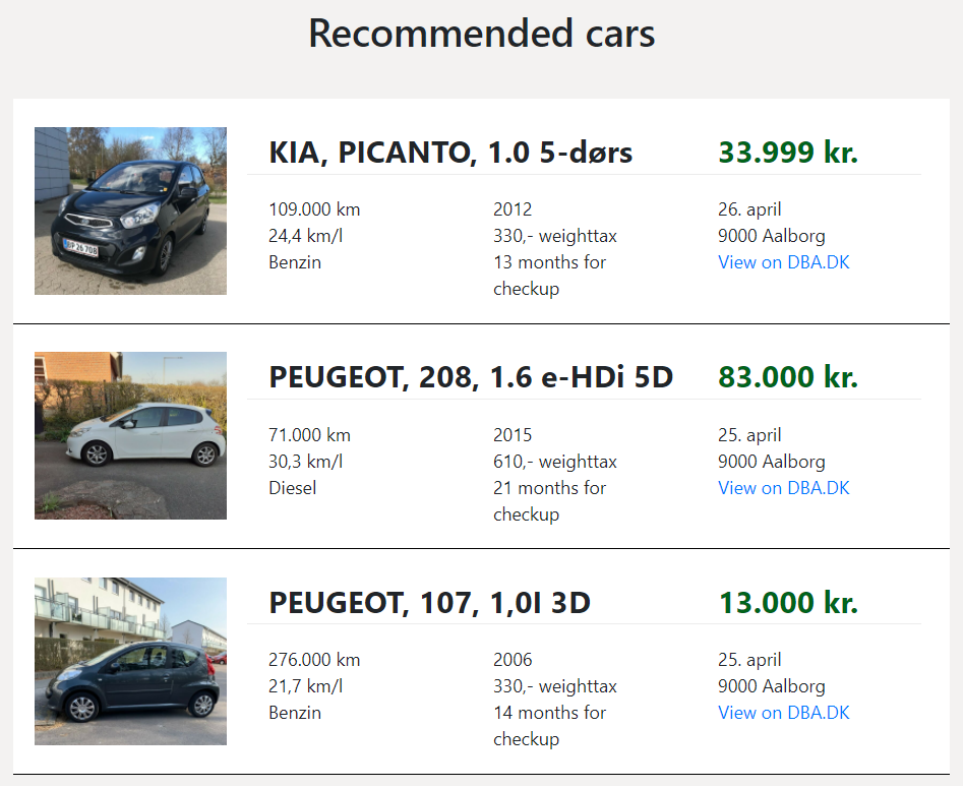

Figure 7.5. Recommended cars

Regarding the run time issue of using the web scrapers in the application, the project group found a solution where the web scrapers would only be called twice every 24 hours at a set time. This solution is less elegant as a user could end up looking at a listing that could have been

changed or even removed from DBA which would waste the users time, going against the very problem that the project seeks to improve upon. The decision was made purely for the purpose of making the application functional. As the application is limited to being a minimum viable product the solution was sufficient but would be something the project group would seek to improve upon if the project should be further developed upon at a later date.

With the third sprint concluded, the application as a whole was completed. The recommendation system worked rather well and functioned as intended. The design of the website remains somewhat simple, but it functions as it was planned to.

7.4 The fourth sprint

The fourth sprint was conducted after the work with the application was fully or mostly completed. The work within this sprint was mainly focused towards finishing the project paper that slowly had been worked on throughout the project. In the previous sprints the project had taken notes on how each component was designed which made it easier to document the respective chapters of the paper.

Sprint outcome The Project team divided the work between the different member to work on different chapters. The team would then collectively review and discuss what had been written thus far in order to give constructive feedback on the progress. This iterative approach would then continue until a chapter was finished. The evaluation and conclusion was written as a collective effort by the entire team, so all members could give input contributing to the outcome of the project. The last part of the fourth sprint was the proofreading of the project paper which was splint into two steps where each member would first read specified sections of the project paper leaving comments to what needed to be adjusted. Next the project group would go through each of these comments together deciding if and how each comment should be resolved. This process would then be repeated for the entire project paper. Following this approach, each member of the project team was given an opportunity to present their input both in regard to what and how changes should be introduced.

Discussion 8

This chapter will reflect on the methods used in the project work as well as how some aspects of the project potentially could have been approached differently.

8.1 Reflections on Scrum

The project team put time and effort in incorporating the Scrum methodology into the project. The intent was to work according to the agile methodology with the intended outcome of having a better and more flexible development process. The decision to use Scrum was ultimately positive as it enabled the group to work more iteratively and explore multiple options when approaching problems throughout the project. A concern that was raised collectively within the project team was the time and energy required to maintain the Scrum guidelines and practices. The concern was whether the time invested was equally weighted as the actual outcome of following the Scrum method. The development process would likely still be successful in the case of Scrum not being used.

A reason for not choosing the Scrum method could be caused by the size of the project team and the scope of the problem that the project dealt with. The project team consisted of 4 members which meant that close and optimal communication could most likely be facilitated without Scrum and the distribution of team roles would have been much more effective in a larger team. The size of the work which was required to solve the problems in the project was also not extreme. The use of product backlogs and sprints would have been more effective if the system in question had a vastly larger scope. In summary, the Scrum and agile methodology would have been more useful if the project had been much larger in scope.

8.2 API usage

To acquire the information connected to the number plate of each car from the MVA, the project group used a web scraper that could input the number plate gathered from the ANPR and then extract the necessary data into the database. Although this solution worked it was not the best solution to solve this problem, as the MVA had an API that, if used, would improve the current process as the web scraper would be able to operate much faster. The reason for not utilizing the API was chosen out of necessity as the API was rather difficult to gain access to. It required

four steps to be completed to gain access to it Skat.dk [2021b].

One requirement was to file an application to the MVA. Furthermore, you needed to be a registered business to do so, which removed it as an option for the project group. Additionally, you were required to sign a connection agreement from the MVA which would only be sent once your application was accepted which could take quite some time. The final requirements were related to setting up employee access and system-to-system access to the MVA system. With this knowledge, the project group concluded that the usage of the API was a non-viable solution, which only could be implemented if future work was to occur.

The requirement for being a registered company was unable to be fulfilled, and the required time spent in communicating with the MVA would be too time taxing. The project therefore sought to find another solution to the problem, which was to use another web scraper.

8.3 Scope of testing

Upon testing the ANPR-system of the application, it was found that the scope of the testing might have been too narrow; only 51 cars was possible to take into consideration when making an overall estimation of the accuracy of the ANPR-system. This hurts the overall credibility of the test results, showcasing a 76% accuracy on the small test-set. The first step in future development, should be to highly prioritize this issue, so that the accuracy can be proven with a much higher certainty and not just be indicative.

8.4 User Involvement

An important aspect that was omitted from the project was the inclusion of users in the application evaluation. It could have been useful to have conducted more tests where potential users or other potential stakeholders was included. an example of such a test could have been a usability test in the attempt to improve the user interface and user experience when using the application. This could have yielded valuable insight into how different people would use the platform and what needs that they may have. It could also give insight into what information users would look for and whether something was missing that the project group had not thought of. It might also be the case that some functional component was missing from the system such as the ability to log in to the platform in order to save certain listings or perhaps there would be a need for a share function that could link a listing to social media platforms. User involvement would have been a good way to validate the system, but due to a strong focus on the technical aspects of the application, the project group omitted user involvement at the current state of the application. A usability test would have required an extensive amount of time to plan, execute and analyze. This time was instead prioritized towards development of the application. Effort was put into verifying the system to ensure that it was developed in accordance with the

outlined application design and that it functioned as intended. In the case where the project would be expanded or revisited, usability testing would be highly prioritized.

8.5 Issues with data ownership

A rather large problem which the project could have a risk of encountering is the issue of using data that is contained within another website, this being the car listing from DBA. Acquiring the information using a web scraper could be seen as a legal gray area as DBA might not be interested in their data being used elsewhere. The issue lies in the potential value that DBA could lose in revenue in regard to the advertisements displayed on their page. Without access to the data from DBA the application would be unable to function. It is important to note that the application is a proof of concept rather than a commercialized product. The research question that the project group sought to answer was whether it was possible to create an application that was able to improve the process of buying used cars online by making the information of each car more accessible. The application developed in the project would be a proposal of how the current state of selling cars could be improved. It would not be unlikely that DBA or similar websites could be improved by including this or a similar solution, combining their car listing with an external database with additional information. The concern regarding data ownership is still valid, but not to an extent where it will have impact on the currently formulated project.

Conclusion 9

In conclusion of the project, we will attempt to answer the initial research question as presented below:

Can the lack and scattering of information in online car marketplaces be solved by providing a centralized platform that enriches the car details.

Considering the issues presented in regard to finding uniform and enriched data on existing online car marketplaces as presented in the initial introduction, the project group developed the Carpair application. Carpair tackles both questions in the research question, tackling both the scattering of information across online car marketplaces and the lack of information on the specific platforms.

The issue regarding the lack of information on online car marketplaces is partly solved by Carpair. Carpair enriches the car details by utilizing ANPR for extracting the number plate from the car listings. The number plate is then used by a web scraper to extract further information gathering from the MVA website. However, the problem is only partly solved, due to the ANPR only being able to extract the number plate in approximately 75 % of the car listings. If the issue should be fully solved then the extracting-percentage should be higher. The results furthermore showcases the potential of using pre-trained weights in established neural networks, to use in potential commercial applications.

The issue of scattered information is partly solved by utilizing web scrapers that takes data from dba.dk as well as the MVA website and centralizes it in the Carpair application. However, if the scattering problem should be fully solved, then all the other car marketplaces should be web scraped (e.g. guloggratis.dk and bilbasen.dk). A problem arising is the gray area presented by web scraping online car marketplaces. Web scraping is legal in some circumstances and illegal in others. If the platform should be published, then the legal rule-set should be clarified.

In summarization, the Carpair application serves as a Minimal Viable product, that showcases the potential of utilizing existing data sources to solve the lack and scattering of details in online car marketplaces. However, the platform only partly solves the problems, as the car listings are only scraped from dba.dk and the ANPR only extracts the number plate in approximately 75 % of car listings.

Future Work 10

Looking at the fruition of the work of the project, it shows promise in regard to utilizing what can be considered to be complex practices such as machine learning to develop applications that serves a relatively niche solution. In this regard the use of publicly available pre-established state-of-the-art object detection models, showcase the accessibility of the technology. In junction with using pre-trained weights for the model available online, the accessibility of utilizing neural networks in applications is seen to continue in the future.

10.1 Model optimization

A challenge in relation to the Carpair application could be the successful extraction of number plates. This does not lie in the fault of the object detection which has a 99 percent success rate, but instead in trying to balance irregular images, pre-processing and optical character recognition in extension to that. In the future this could be optimized, with the possibility of reaching a success rate in the high 90s, looking at the existing applications. To further add upon the ANPR success rate, unsuccessful scans containing number plates can be sent to developers and identified successfully, and route the results back in to the model, thus optimizing it. This could also be done by users as a CAPTCHA when utilizing the site to a specific regard.

10.2 Efficiency

Looking at the existing bottlenecks of the application, the web scraping is functional but can be prone to errors if changes occur on the website that is being scraped and also the gray area presented by using it in commercial applications. In the case of future development the data-gathering should be performed utilizing official APIs from online marketplaces as well as the motor vehicle agency, which in turn should improve the time of which data can be gathered and ultimately save processing power.

In addition to the optimization of web scraping, the recommendation system can also be upgraded for higher perceived value for the users. This could be done by caching the users' activity in the application and then serve recommended cars to the user which share similar traits to a normalization of the common characteristics of the users interactions in the application.

10.3 Usability

Regarding the current state of usability, it is needed to expand the search-radius in regard to which cars are considered for the application, thus expanding from Aalborg to the entirety of Denmark on DBA, from where it can be desired to expand to other existing car sales site, effectively creating a platform that contains an overview of all car listings made where a number-plates is visible in the images posted, as well as having the largest depth of information for the user to query upon.

This projects main purpose and focus have not lied in considering the commercial aspect of the application, but instead the functional and technical structure of it as a minimal viable product. In addition to this, for a potential commercial roll-out then extensive usability testing should be conducted both in regard to discovering elements that should be added or removed, but also in relation to the actual use of the application, which should be aimed towards being intuitive.

10.4 Applications

Looking ahead, the deep data that is gathered on a large percentage of car sales in Denmark could serve as a dataset for developing future applications in the field of data science and machine learning. The ANPR model could also be utilized in other businesses such as car resellers, parking companies or the police. The application itself could be expanded in regard to the geography covered on individual online marketplaces, but also the total amount of online marketplaces covered in the application.

Bibliography

Bochkovskiy et al., 2020. Alexey Bochkovskiy, Chien-Yao Wang og H. Liao. *YOLOv4: Optimal Speed and Accuracy of Object Detection*. ArXiv, abs/2004.10934, 2020.

Brewer, 2001. Devon D. Brewer. *Regression and Correlation*, 2001. URL `http://faculty.washington.edu/ddbrewer/s231/s231regr.htm`. Accessed: 03/05/2021.

Hieu Minh Bui, Margaret Lech, Eva Cheng, Katrina Neville og Ian S. Burnett, 2016. Hieu Minh Bui, Margaret Lech, Eva Cheng, Katrina Neville og Ian S. Burnett. Using grayscale images for object recognition with convolutional-recursive neural network. In *2016 IEEE Sixth International Conference on Communications and Electronics (ICCE)*, pages 321–325, 2016. doi: 10.1109/CCE.2016.7562656.

Böckeler, 2021. Birgitta Böckeler. *MARKEDET FOR BRUGTE BILER HAVDE SIT BEDSTE ÅR NOGENSINDE I 2020*, 2021. URL `https://martinfowler.com/articles/on-pair-programming.html`. Accessed: 13/04/2021.

Chavan, 2021. Akshay Chavan. *Process the output of cv2.HoughLines*, 2021. URL `https://arccoder.medium.com/process-the-output-of-cv2-houghlines-f43c7546deae`.

DTU, 2014. DTU. *Faktaark om biltrafik i Danmark*, 2014. URL `https://www.cta.man.dtu.dk/-/media/Centre/Modelcenter/modeller-og-publikationer/Faktaark/2013-Faktaark_biltrafik.ashx`. Accessed: 03/05/2021.

Foget, 2019. Emil Foget. *Hvad koster det at holde bil?*, 2019. URL `https://santanderconsumer.dk/magasinet/bil-og-mc/hvad-koster-det-at-vedligeholde-en-bil/`. Accessed: 03/05/2021.

Fraser, 2020. Tom Fraser. *What is engine size, and why does it matter?*, 2020. URL `https://www.whichcar.com.au/car-advice/what-is-engine-size-and-why-does-it-matter`. Accessed: 03/05/2021.

FuelFinder, 2021. FuelFinder. *Nyeste listepriser på brændstof for danske selskaber*, 2021. URL `https://www.fuelfinder.dk/listprices.php`. Accessed: 03/05/2021.

Howard et al., 2017. Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto og Hartwig Adam. *MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications*, 2017.

Jaderberg et al., **2016**. Max Jaderberg, Karen Simonyan, Andrew Zisserman og Koray Kavukcuoglu. *Spatial Transformer Networks*, 2016.

Kaur, **2020**. Harkiran Kaur. *What is Web Scraping and How to Use It?*, 2020. URL https://www.geeksforgeeks.org/what-is-web-scraping-and-how-to-use-it/. Accessed: 14/05/2021.

Ken Schwaber, **2020**. Jeff Sutherland Ken Schwaber. *The Scrum Guide*. 2020.

Lemche, **2018**. Karsten Meyland Lemche. *GRØNNE BILER ER (LIDT) BILLIGERE*, 2018. URL https://fdm.dk/sites/default/files/inline-files/Beregning%20Motor-08.pdf. Accessed: 03/05/2021.

Lund, **N/A**. Arne Lund. *Trafikpolitik*, N/A. URL https://www.leksikon.org/art.php?n=3563. Accessed: 03/05/2021.

Meng et al., **02 2020**. Zhenzhu Meng, Yating Hu og Christophe Ancey. *Using a Data Driven Approach to Predict Waves Generated by Gravity Driven Mass Flows*. Water, 12, 2020. doi: 10.3390/w12020600.

MsCOCO, **2021**. MsCOCO. *MsCOCO*, 2021. URL https://arxiv.org/pdf/1405.0312.pdf. Accessed: 10/05/2021.

Notes, **2021**. Scipy Lecture Notes. *Otsu thresholding*, 2021. URL http://scipy-lectures.org/packages/scikit-image/auto_examples/plot_threshold.html. Accessed: 10/05/2021.

OpenALPR, **2021**. OpenALPR. *OpenALPR*, 2021. URL https://www.openalpr.com/. Accessed: 22/04/2021.

OpenCV, **2021a**. OpenCV. *Contours : Getting Started*, 2021. URL https://docs.opencv.org/master/d4/d73/tutorial_py_contours_begin.html. Accessed: 10/05/2021.

OpenCV, **2021b**. OpenCV. *Image Tresholding*, 2021. URL https://docs.opencv.org/master/d7/d4d/tutorial_py_thresholding.html. Accessed: 10/05/2021.

OpenCV, **2021c**. OpenCV. *Eroding and Dilating*, 2021. URL https://docs.opencv.org/3.4/db/df6/tutorial_erosion_dilatation.html.

OpenCV, **2021d**. OpenCV. *Smoothing Images*, 2021. URL https://docs.opencv.org/master/d4/d13/tutorial_py_filtering.html.

OpenCV, **2021e**. OpenCV. *Hough Line Transform*, 2021. URL https://docs.opencv.org/3.4/d9/db0/tutorial_hough_lines.html.

OpenCV, **2021f**. OpenCV. *Geometric Transformations of Images*, 2021. URL https://docs.opencv.org/master/da/d6e/tutorial_py_geometric_transformations.html.

openCV.com, **2021**. openCV.com. *openCV*, 2021. URL https://opencv.org/. Accessed: 10/05/2021.

Saha, **2021**. Sumit Saha. *A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way*, 2021. URL https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53. Accessed: 12/05/2021.

S. M. Silva og C. R. Jung, Sep 2018. S. M. Silva og C. R. Jung. License Plate Detection and Recognition in Unconstrained Scenarios. In *2018 European Conference on Computer Vision (ECCV)*, pages 580–596, Sep 2018. doi: 10.1007/978-3-030-01258-8_36.

Sipser, **2021**. Michael Sipser. *ntroduction to the Theory of Computation*. Cengage Learning, Inc, 2021.

Skat.dk, **2021a**. Skat.dk. *Motor Vehicle Agency*, 2021. URL https://www.motorst.dk/english/the-motor-vehicle-agency-motorstyrelsen/. Accessed: 20/03/2021.

Skat.dk, **2021b**. Skat.dk. *Adgang til Motorregistret*, 2021. URL https://skat.dk/SKAT.aspx?oid=2111621. Accessed: 16/04/2021.

skat.dk, **2021**. skat.dk. *Beregn vægtafgift og grøn ejerafgift*, 2021. URL https://skat.dk/skat.aspx?oid=4052. Accessed: 03/05/2021.

Statestik, **2019**. Danmarks Statestik. *Motorparken – bestanden af personbiler 1. januar 2019*, 2019. URL https://www.dst.dk/Site/Dst/Udgivelser/nyt/GetPdf.aspx?cid=28467. Accessed: 18/03/2021.

Statestik, **2021**. Danmarks Statestik. *Bestanden af personbiler pr. 1. januar efter egenvægt, drivmiddel og tid*, 2021. URL https://www.statistikbanken.dk/BIL10. Accessed: 17/03/2021.

Steensen, **2020**. Nina Steensen, Emma Siessegger. *On Pair Programming*, 2020. URL https://bilmagasinet.dk/bil-nyheder/konklusion-brugtbilsmarkedet-2020-stak-af. Accessed: 19/03/2021.

Thorsen, **2021a**. Thomas Thorsen. *Numberplate Combinations*, 2021. URL http://www.nrpl.dk/comb-1966.php. Accessed: 05/04/2021.

Thorsen, **2021b**. Thomas Thorsen. *Military numberplates*, 2021. URL http://www.nrpl.dk/def-eme.php. Accessed: 06/04/2021.

w3schools.com, **2021**. w3schools.com. *JSON introduction*, 2021. URL `https://www.w3schools.com/js/js_json_intro.asp`. Accessed: 15/05/2021.