NOVA IMS Information Management School

MAA

Mestrado em Métodos Analíticos Avançados
Master Program in Advanced Analytics

Mixed-input second-hand car price estimation model based on scraped data

Matteo Fiorani

Dissertation presented as partial requirement for obtaining the Master's degree in Advanced Analytics

NOVA Information Management School
Instituto Superior de Estatística e Gestão de Informação
Universidade Nova de Lisboa

NOVA Information Management School

Instituto Superior de Estatística e Gestão de Informação

Universidade Nova de Lisboa

MIXED-INPUT SECOND-HAND CAR PRICE ESTIMATION MODEL BASED ON SCRAPED DATA

by

Matteo Fiorani

Dissertation presented as partial requirement for obtaining the Master's degree in Advanced Analytics

Advisor: Prof Doutor Mauro Castelli

September 2021

DEDICATION

I want to dedicate this achievement to my family, especially my mother, who never stopped supporting me, not even for a minute. To my master colleagues and dear friends, Davide Montali and Umberto Tammaro who helped me transition back to school and gave me the strength to get through the master program and the motivation to work on this thesis while in a full-time job.

ABSTRACT

The number of second-hand cars is growing year by year. More and more people prefer to buy a second-hand car rather than a new one due to the increasing cost of new cars and their fast devaluation in price. Consequently, there has also been an increase in online marketplaces for peer-to-peer (P2P) second-hand cars trades. A robust price estimation is needed for both dealers, to have a good idea on how to price their cars, and buyers, to understand whether a listing is overpriced or not. Price estimation for second-hand cars has been, to my knowledge, so far only explored with numerical and categorical features such as mileage driven, brand or production year. An approach that also uses image data has yet to be developed.

This work aims to investigate the use of a multi-input price estimation model for second-hand cars taking advantage of a convolutional neural network (CNN), to extract features from car images, combined with an artificial neural network (ANN), dealing with the categorical-numerical features, and assess whether this method improves accuracy in price estimation over more traditional single-input methods.

To train and evaluate the model, a dataset of second-hand car images and textual features is scraped from a marketplace and curated such that more than 700 images can be used for the training.

KEYWORDS

Data Science; Artificial Intelligence; Machine Learning; Deep Learning; Convolutional Neural network;

Scraping, Used Cars;

INDEX

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS AND ACRONYMS

P2P Peer-to-Peer

CNN Convolutional Neural Network

ANN Artificial Neural Network

ML Machine Learning

API Application Programming Interface

WWW World Wide Web

HTTP Hypertext Transfer Protocol

DNS Domain Name System

HTML Hypertext Markup Language

CSS Cascading style sheet

JS Javascript

DOM Document Object manipulation

OLS Ordinary Least Squares

SSE Sum of Squared Errors

KNN K-Nearest Neighbors

CART Classification and Regression Trees

SSR Sum of Squared Residuals

AI Artificial Intelligence

MLP Multilayer Perceptron

SGD Stochastic Gradient Descent

IQR Interquartile range

SSR Sum of Squared Residuals

AI Artificial Intelligence

MSE Mean squared error

MAE Mean Absolute error

RMSE Root Mean Square Error

MAPE Mean Absolute Percentage Error

GRAD-Cam Gradient-weighted Class Activation Mapping

RAM Regression Activation Maps

Hybrid-DNNs Hybrid Deep Neural Networks

SVR Support Vector Regressor

SVC Support Vector Classifier

1. INTRODUCTION

1.1. OBJECTIVES

This work aims to develop a model for estimating second-hand cars prices, and more precisely, investigate whether the addition of image input data alongside numerical and categorical data yields a better accuracy in this domain compared to more traditional approaches where image data is neglected. Firstly, traditional machine learning (ML) methods are applied to numerical and categorical features to establish a baseline. Secondly, Deep learning (DL) methods are developed to address separately numeric and image data. For numerical and categorical features an ANN is developed, whereas to address the image data, a CNN is developed and the results analyzed. Finally, a mixed-input model is developed and the results compared to single-input models.

1.2. CONSTRAINTS AND LIMITATIONS

Machine learning and Deep Learning (DL) methods require a large amount of data. In particular, CNN's need a large number of pictures to make the model able to extract important features from the training images and therefore being able to generalize and perform well on unseen pictures. The dataset used in this work was created by scraping a P2P trading platform of second-hand cars. The amount of time required to build such a large dataset manually is not trivial. As explored in the next chapter, it is possible to automate the scraping and processing completely when it comes to textual data but not entirely for the image data. To be useful, the images automatically scraped have to be one by one carefully processed through different steps, from labeling to cropping. In other words, scraping a large number of images can be done automatically once the scraping program is developed, but these images can not be directly used for training as they come and require manual human supervision to process them. Due to the limited time available to develop this thesis, a total amount of 3000 images belonging to 750 cars is processed and used to train CNN's.

2. LITERATURE REVIEW

Price estimation models for second-hand cars have been extensively explored in multiple studies. In the first papers, researchers explored the application of traditional machine learning algorithms. In 2009, Listiani applied a Support Vector Regressor (SVR) on leased cars to estimate their price and compared the results with a Linear Regression (LR) method. The dataset was high-dimensional and comprised of around 124386 samples and 180 columns. The error metric chosen was Root Mean Squared Error (RMSE). Different experiments were run, and the results showed how both with, and without features selection, SVR outperformed LR. The best results yielded an RMSE of 5938. (Listiani, 2009). Subsequently, the first papers appeared where deep learning was also applied (Peerun et al,2015). In this study, the authors collected a dataset of 200 second-hand cars in Mauritius, where in the previous years the second-hand car sales surpassed new car sales. The objective of this work was to apply a Neural Network to predict car prices and comparing the results with traditional machine learning methods. More specifically Linear Regression (LR), K-nearest regressor (KNN), and Support Vector Regressor. The authors found that, overall, the results were similar between ML and DL approaches, but that SVR performed slightly better than a simple Neural Network with 1 hidden layer and 2 nodes and that a simple Linear Regression was only slightly worse than the Neural Network approach. The error metric used for this study was the mean absolute error (MAE). It is important to point out that the dataset collected was relatively small for ML approaches. It is reasonable therefore that a simple Linear Regression would almost outperform a Neural Network. Finally, studies explored the use of ensemble machine learning models to improve the performance of singular models (Gegic et al, 2019). In this work, the author collected information of more than 1000 cars by web scraping an online second-hand car trading platform in Bosnia and Herzegovina. After data processing, the final dataset comprised 739 cars. The objective of this work was to apply ensemble learning to second-hand car price prediction by combining the predictive power of different estimators to improve prediction accuracy. The authors transformed the problem from a regression to an ordinal classification problem by binning the price into 3 different categories based on price range. Cheap cars with a price range from 0 to 6000 euros, moderate from 6000 to 12000 and expensive when the price exceeded 12000 euros. The models used for the ensemble classification model were an Artificial Neural Network (ANN), a Random Forest classifier (RF), and a Support Vector Classifier (SVC). The application of singular approaches only yielded accuracies up to 50% whereas the ensembles approach improved the accuracy to a remarkable 90%. These results confirmed how combining different machine learning algorithms can improve accuracy for second-hand car price prediction.

Some of these works achieved good values in prediction accuracy but they all fail to include image data, excluding many important features that could improve model performance. A hybrid machine learning model to second-hand car price estimation has, therefore, up to the time of this writing and to my knowledge, not yet been explored. Hybrid models have nevertheless been applied to other domains. For instance, in house price estimation (Eman Ahmed et al, 2016) where pictures of houses are combined to their listing data in real estate web pages. In this paper, the authors were able to develop a mixed input price estimation model with an MLP and CNN combined that outperformed a baseline model only based on textual features. Pictures of different parts of the houses coming from online listings were imputed together into the CNN branch of the mixed-data model. The number of sample houses used for this study was 535, 4 pictures per house were prepared and fed to CNN, totaling 2140 pictures. The metric of choice to evaluate the performance of the model was the root mean square error (RMSE). Also, other studies have been conducted in Geoscience, especially in the realm of commodities such as oil production, where the features that make a Hybrid neural network perform better than a singular MLP and CNN have been analyzed (Zhenyu Yuan et al, 2020). In this study the author explains how geological data can be very diverse, including images, sequences, numerical and categorical data, and mixed input models are praised for how they can improve generalization and prediction accuracy. Also, it is indicated that their flexibility is very important in the world of big data where machine learning practitioners have to work with datasets that are always increasing. The importance of data preparation and preprocessing is also highlighted as mixed data comes in different shapes. The proposed architecture uses techniques like batch normalization and drop out to enhance the model's training time and performance. Rectified linear unit (ReLU) is used as an activation function to include nonlinearity in the model. More recently, in the biomedical field, mixed-input models were applied to predict whether a patient was affected by covid-19 in the early stages to prevent the virus from spreading. In this study, numerical and categorical features about the patient's health history were combined with chest X-ray images. Furthermore, the authors experimented with multiple optimizations algorithms including adaptive learning rate optimization algorithm, stochastic gradient descent, and root mean square propagation. The experiments suggested how a model trained with adaptive learning rate (Adam) yielded better prediction accuracy than the rest. Hybrid machine learning models have been applied successfully to both classification and regression tasks.

3. THEORETICAL FRAMEWORK

3.1. WEB SCRAPING

The data analyzed in this work was collected and scraped from an online marketplace for P2P second-hand cars trades. Scraping is the process of automating data download, processing, and organization from an online source. (S. vanden Broucke et al,2018, p. 3)

It therefore replaces the laborious process of a human navigating a website and clicking around to download and collect data by, for instance, copy and pasting manually into a spreadsheet (S. Vanden Broucke et al, p 3)

This process would be too cumbersome and time-consuming when collecting a dataset for machine learning purposes that, by nature, can include hundreds, if not thousands, of samples.

Many websites do offer an application programming interface (API) to scrape their content, but it is usually only the most frequented websites with more resources such as Facebook or Twitter that can develop such products. Also, many APIs only let the user scrape certain kinds of data. This means that if a website offers an API, the developer/data scientist might still need to use other scraping methods to collect the data of interest. Also, some APIs are not free to use, whereas automated scraping is. (S. vanden Broucke et al, 2018, p 5).

To understand how scraping works, a basic understanding of how the world wide web (WWW) works is necessary. In this chapter, some basic concepts such as HTTP, HTML, CSS, Javascript are explored.

3.1.1. HyperText Transfer Protocol

One of the main building blocks of the internet is the Hypertext Transfer Protocol (HTTP). It is a fairly simple client-server protocol as it is text-based and allows good interpretability by the user (S. Vanden Broucke et al, 2018, p29). In simple words, HTTP is a protocol that allows and facilitates the exchange of information online. An HTTP client, for example, a browser, or in the case of this work, a python web scraping program, sends a request to a web server and waits for an HTTP response. The web server replies with a response, for example, an HTML page, and stops the connection.

For this thesis, the python library *requests* is be used to take care of HTTP. An example of its functionality can be seen in the picture below.

```python
import requests

url = 'http://www.google.com/'
request= requests.get(url)

print(request.text)
```

Figure 3.3.1 - Example of request library

After importing the library a variable *URL* containing the web server Domain Name System (DNS) is stored and in line 4 a request is fired and the response stored in the variable *request.* In line 6 the response is printed after calling the method *.text* which returns the text content of the response. A connection between the client and the server is now correctly established.

3.1.2. HyperText Markup Language

Hypertext Markup Language (HTML) is the standard markup language for creating web pages by specifying their structure and format (S. Vanden Broucke et al, p 50). HTML achieves this with the use of tags. As can be seen in the picture below, tags come in pairs with a start tag and a closing tag containing a specific element.

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <h1>this is a heading</h1>
    <p>this is a paragraph</p>
</body>
</html>
```

Figure 3.3.2 - Example of HTML document

It would be possible to simply create a request by knowing the website's name and obtain the HTML back in response to start investigating the page structure and building a web scraper. While possible, this would be a very cumbersome process and require a lot of trial and error to know exactly which HTML element corresponds to which part of the webpage. To make this process more flexible, many web browsers provide tools to inspect and view the source code behind a webpage in a dynamic manner. This means that by simply hovering on the element of interest in the rendered page, the corresponding part of the underlying code will be highlighted.

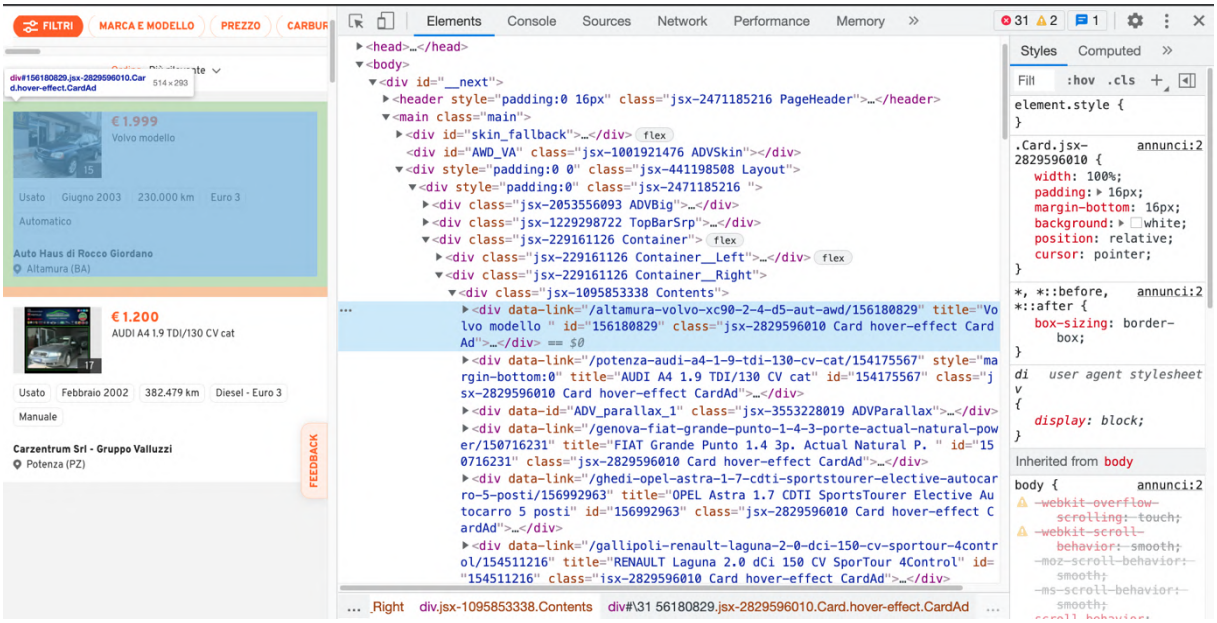

Figure 3.3.3 - Browser inspector in Google Chrome

3.1.3. Cascading style sheet

Cascading style sheet (CSS) is a style sheet language and it works together with HTML to stylize a website. (S. vanden Broucke et al,2018, p. 50). The main HTML attributes that play along with CSS are *Id* and *Class*. An id identifies a unique HTML element, whereas a class identifies a collection of elements that belong to the same family. CSS is then written to stylize these parts of the HTML page. This is important to understand because HTML and CSS force a webpage and its rendered elements to be more or less well structured for the web page to look pleasant, and this will make the process of recognizing and scraping the right elements in the webpage easier. (S. vanden Broucke et al, 2018, p 50).

```
1  body {
2    background-color: white;
3  }
4  h1 {
5    color: blue;
6  }
7  p {
8    color: red;
9  }
```

Figure 3.3.4 - Example of CSS

In figure 2.4 a typical structure of a CSS document is shown. Usually, CSS is stored in a different file than HTML, and the style that needs to be attributed to a particular tag is stored between curly brackets.

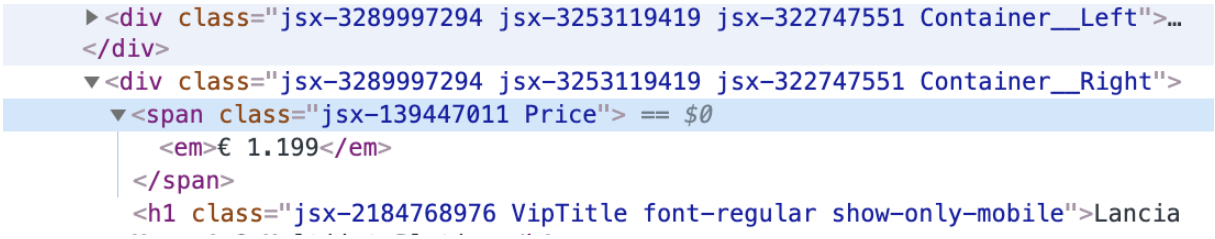

Figure 3.3.5 - Example of HTML class method

In figure 2.5 the highlighted class is containing the price for a car listing on a trading platform. As the style applied to text is the same throughout the page, the tag can be used to automatically identify and download the price of the listed cars.

3.1.4. Beautiful Soup

Beautiful Soup is one of the main libraries for retrieving data from markup languages such as HTML. It includes different types of parsers, allowing the user to scrape from pages that have poorly written HTML. (Jay M. Patel,2020,p.37). Beautiful soup works well along with the request library, which is introduced in 2.2.1. After requesting the webpage source code, Beautiful Soup is then used to retrieve the sought information. There are many different methods to extract information, but the most used ones are find() and find_all(). The method find_all() allows the user to pass as argument an HTML tag to fetch all the content in the page wrapped by that tag. On the other hand, find() only retrieves the first matching tag. (Jay M. Patel,2020,p.37)

```
def download_data(car,carid):

    r = requests.get(car).text
    soup = BeautifulSoup(r, 'lxml')

    price = soup.find('span',{'class': 'jsx-139447011 Price'})
    restfeatures = soup.find_all('span',{'class': 'jsx-3587327592 font-base font-bold'})

    F=[]
    try:
        price = price.text.replace('€', '').replace(',', '').replace('.', '').strip()
    except:
        price ='not_found'
    try:
        for i in restfeatures:
            t= i.text.strip()
            F.append(t)
    except:
        pass
```

Figure 3.3.6 - Example of Beautiful Soup methods

In figure 2.6 a chunk of the function download data developed to scrape the numerical and categorical feature can be observed. In this case, the function argument *car* is the link to a car listing, whereas

carid is the id of the car. In line 3 the HTML of the car listing page is requested and in line 4 the code is stored in a variable called soup. Now the Beautiful Soup method find() is used to find the right tag for the price of the car, in this case, a *.* While price has its own class, other car features all share the same one. This means that the rest of the features need to be accessed with the method find_all() and stored in a variable for further processing.

3.1.5. Scraping with JavaScript

Issues can occur in scraping when a web page is using scripts to enhance the website functionality. An example of language used for this purpose is Javascript (JS). JS is one of the main technologies used on the web and most websites currently use it extensively. (Jay M. Patel,2020, p.68). Scripts used to be placed on top of a webpage, in the head part of the HTML, and all the content was loaded right away when opening a website. This means that even content that a user was not interested in, would load and increase computation time. The advantage of this procedure was that while inspecting a website, the HTML displayed was exactly what a user would expect it to be, making scraping content easier. In practice, the developer would need to wait a few seconds before sending a request to the website, so that the browsers had enough time to execute the javascript present on the webpage. The more the content became sophisticated and the user attention span lower, web pages had to become more dynamic and fast. Nowadays, scripts are placed everywhere on the web page and the content of interest for scraping might not appear right away and not even after letting the website execute the scripts present at the top of the page. (Jay M. Patel,2020, p.68).

```
<script>
    function changeColor() {
        var msg;
        p = document.getElementById("firstParagraph");
        p.style.color = "blue";
    }
</script>
```

Figure 3.3.7 - Example of JavaScript in HTML page

Javascript works on the concept of Document Object manipulation (DOM), which means it can dynamically change the structure of the website while interacting with it. For instance, a piece of information might be “hidden” and only available when the user clicks a button. Such data cannot be retrieved only with traditional scraping techniques, for example with the use of BeautifulSoup. It will be necessary to use more advanced techniques that recreate the behavior of a user and implement it into code. In practice, a python program that is automatically looking for a button or another element by class or id name and clicks it to trigger the script and load the information needed. At this point, the

content will be displayed and can be retrieved as expected. An important library for such a task is Selenium.

3.1.6. Selenium

Selenium is a library to automate web browsing. It was originally intended for web testing but it has developed into a fundamental tool in scraping. (Jay M. Patel,2020, p.76). As mentioned above, Selenium lets the user uncover information that is concealed by JavaScript but it also comes with a cost, mainly time and memory consumption. (Chapagain, 2019,p.393). If the objective is to scrape a lot of data, automating the way a user interacts with increases exponentially scraping time because every single step to reach the wanted information has to be repeated n times, where n is the total number of instances we want to scrape. The main Selenium component that is interesting for scraping is the selenium Webdriver. Webdriver is an object-oriented API and can be implemented with any browser and allows to automate user behavior on a site and also locate elements of interest (Chapagain, 2019,p.396). Selenium comes with many locators, similar to find and find_all in Beautiful Soup. Some important locators are: *find_element_by_id(), find_element_by_name(), find_element_by_tag_name(), find_element_by_class_name()* and *find_element_by_xpath().*

```
def download_images(car,carid):

    driver = webdriver.Chrome(os.getcwd() + "/chromedriver")
    driver.get(car)
    time.sleep(5)
    next_button =  driver.find_element_by_xpath('//button[@class="jsx-3065026281 Navigation Navigation--next"]')

    for _ in range(1,10):
        try:
            next_button.click()
            time.sleep(3)
        except:
            driver.find_element_by_xpath('//div[@id="onetrust-policy"]').click()

    driver_source = driver.page_source

    driver.close()
    driver.quit()

    soup = BeautifulSoup(driver_source, 'lxml')
    images = soup.find_all('img',  {'class': 'jsx-2129646631','data-type':'slide'})

    for image in images:
        try:
            print(image['src'])
            imglink= image["src"]
            with open(SAVE_FOLDER + '/' + imglink.replace(' ','-').replace('/','')+ '.jpg', 'wb') as f:
                img = requests.get(imglink)
                if img.status_code == 200:
                    f.write(img.content)
                    print('writing: ', carid)
        except:
             pass
```

Figure 3.3.8 - Example of Selenium and Beautiful Soup in use to scrape car listing images

In figure 2.8 we can see a function that is developed to scrape the car listings images. Every single image is stored in a slideshow of images and to access them the *next* button needs to be clicked for the javascript to load and the image link to appear in the HTML. In line 3 a *webdriver* object is created and in line 4 the method *get* is called with the argument car. When the method is called Webdriver automatically opens a new Chrome window with the passed URL. In line 5 the python method sleep is called that does nothing but wait 5 seconds until executing the next lines of code. This time is necessary for the entire page to load and for the following commands to work. In line 6 the locator *find_element_by_xpath()* is called to find the button next in the webpage. Once the button is found the next button is clicked n times thanks to a for loop. Every time the next button is clicked the scripts are triggered and the link to the images reveal. Now that all the needed HTML is loaded and the source code is stored in an object (line 15), the WebDriver window is firstly closed with the *close()* method (line 17), and then the session is suppressed with the *quit()* (line 18) method to save memory. Now Beautiful soup can be used to find the link to the images (lines 20-21) and store them automatically in a folder (lines 23 to 31).

3.2. MACHINE LEARNING

Machine learning is a branch of computer science whose main focus is to build algorithms that demand the collection of data on a specific topic. This data can be collected from nature, curated by humans, or gathered from the output of another algorithm (Burkov, 2019, p. 3). Machine learning algorithms can be supervised and unsupervised.

3.2.1. Supervised Learning

Supervised learning implies that the data collected for training is labeled. In other terms, every element of the dataset belongs to one class. The algorithm takes as input feature data belonging to a target variable. (Burkov, 2019, p. 3). The target feature is what the model will try to predict whereas the feature data is what the model will base its decision on. The goal is to then predict unseen feature data for which a target variable is not available or not being labeled yet by a human.

Supervised learning can be further divided into regression and classification. Whenever a model is trying to predict a numerical value then we talk about a regression machine learning model. An example of this would be predicting the price of a house or car. On the other hand, when a model is trying to predict a categorical value then we talk about a classification machine learning model. An example of classification would be trying to predict whether the input data fed into the model belongs to a cat or a dog.

3.2.2. Unsupervised Learning

Unsupervised learning implies that the data collected for training is not labeled. That means that there is no target feature corresponding to a feature vector. (Burkov, 2019, p.4). The goal of unsupervised learning is to understand the relationship between the feature data and divide it into groups with similar characteristics. An example of unsupervised learning would be customer segmentation, namely clustering customers into different groups and building marketing and business strategies around them. Another example is anomaly detection.

3.2.3. Linear Regression

Linear regression, also known as *ordinary least squares* (OLS), is the simplest and best-known method for regression. Linear regression allows to model mathematically the relationship between one or more variables and an output. Linear Regression, and linear models in general, are easy to apply, to interpret, are computationally inexpensive, and can even outperform nonlinear models when the training data is small. (T. Hastie et al, 2009, p.43). The goal of linear regression is to minimize the sum of squared errors (SSE), namely the squared difference between the estimated and the actual values. Linear regression is a parametric algorithm, as it assumes that the relationship between the inputs and outputs is linear. Hence, it does not do well with non-linear data (Harrington, 2012, p. 154).

3.2.4. K-Nearest Neighbors

Another easily interpretable algorithm is K-nearest Neighbors (KNN). It is a non-parametric algorithm, as it does not make heavy assumptions about the data and it is slower to train. It can be used for both classification and regression problems.

A k number of data points is selected in the training data based on how similar they are to the input value, namely, the value to predict. These selected points are also called the neighborhood.

To find the most similar data point in the training data, a distance metric is needed. There are many distance metrics such as Manhattan distance and Minkowski distance, but most commonly Euclidean distance is used. As the results will depend on this distance, it is fundamental to scale the data prior to the training phase, to avoid the larger values to skew the prediction. (Kuhn,2018, p. 159).

In a regression problem, the prediction for the test case is simply the average of all the k values in the neighborhood. (Kuhn, 2018, p. 160). It is important to point out that k is a tuning parameter and as such it has to be chosen cautiously. With a k number that is too big, the model risks underfitting the data. Whereas with a small number of k the model will overfit the data.

The KNN algorithm can be easily applied using the sci-kit learn library.

3.2.5. Classification and Regression Trees (CART)

Classification and Regression Trees (CART) are a type of machine learning algorithm that can be used for both regression and classification. What differentiates classification from regression trees is that the former have either boolean values or categorical values in their leaves, whereas in regression trees each tree represents a numeric value. Decision trees are based on binary if-then decisions that are used to split the data in a divide and conquer manner (Kuhn, 2018, p 173). The top of a decision tree is called root, going down three internal nodes can be found, whereas at the bottom we have the leaves. In a Regression tree the first split, namely the root node, is found by taking the linear division of all the features in the training data for which the sum of square residuals (SSR) is smaller. The feature with the smallest SSR becomes the root node. The same process is done recursively for the whole dataset. The prediction is then the average number in any cluster of observations. Decision trees handle missing values and outliers very well and because of the way they are built they don't need any kind of preprocessing. (Kuhn, 2018, p 174). On the other hand, there is a risk of overfitting if the tree goes too deep and splits the data too perfectly. To avoid overfitting, a parameter such as minimum number of observations *n* is applied so that if in a node, after a split, there are less than n observations, that node becomes a leaf and the splitting stops. If the data cannot be homogeneously split into square regions, then the prediction error can be bigger than in other models. (Kuhn, 2018, p 174). To overcome this weakness, other important machine learning algorithms such as random forest and boosted decision trees have been developed on top of decision trees.

3.2.6. Random Forest

Random forest is an ensemble machine learning algorithm based on decision trees. The random forest algorithm works by first bootstrapping the training set, namely by selecting random training samples and allowing duplicate entries. After that, a tree is built only on a subset of features. This process is done multiple times and an *n* number of trees is created. Each tree is then used to predict a new sample and the predictions are then averaged to yield the random forest's prediction. This process of bootstrapping and aggregating different predictions is called bagging. This variety of trees is what makes random forests have great predicting power and reduced variance. Moreover, because of trees being independent from one another, random forest is robust to noisy data (Kuhn,2018, p 205).

3.3. DEEP LEARNING

Deep learning is a type of machine learning that takes inspiration from the structure of the human brain (Goodfellow et al., 2016, p.169). In the artificial intelligence (AI) world, this structure is called an artificial neural network. Neural networks take in the input data and learn the relationship between them and the outputs, the target features (Goodfellow et al., 2016,p.167). The way that neural networks learn is not directly interpretable, for this reason, they are also seen as a "black box". Artificial neural networks have been used successfully for a variety of tasks from classification to regression and have proven to be very successful for speech recognition and image recognition. (Manning,2018, p.11)

3.3.1. Artificial Neural Networks

The simplest form of an ANN is a single-layer network, also called *perceptron.* (Aggarwal,2018,p.5). A perceptron is made of a single input layer and output layer. A Multilayer perceptron (MLP) is a network composed of multiple layers of perceptrons. An MLP adds computational layers between the input and output, these are also called hidden layers (Aggarwal,2018, p.17).

A neural network can be relatively shallow, when there is only one hidden layer, or can go very deep, with multiple hidden layers, when the problem to solve is more complex (Manning,2018, p.8). Each hidden layer contains a certain number of nodes, also called neurons.

When information goes through the network, weights and biases are applied to the inputs, these are called learnable parameters. During the learning phase, the network gets the input values, multiplies them by the weight, and then adds a bias. At this point, an activation function comes in and computes the final value for that layer (Goodfellow et al., 2016, p.171). The activation is an important step because it introduces non-linearity allowing the network to learn far more complex problems. The choice of activation function is based on the nature of the data. (Christopher M. Bishop, p. 245).

The definition of a loss function is necessary for the network to calculate the error between the output target value and the predicted value. The goal could either be to maximize or minimize the function based on the problem at hand. Now the error from the loss function is backpropagated so that the network can adjust the weights accordingly. This process is called backpropagation. The stochastic gradient descent (SGD) algorithm is used to adjust the weights and biases. How fast these weights and biases are updated is based on the learning rate. Smaller learning rates require the network to train for a longer time but provide better accuracy. On the other hand, bigger learning rates make the

network converge fast to a solution. The learning rate is the most important hyperparameter in a neural network (Goodfellow et al., 2016, p.429).

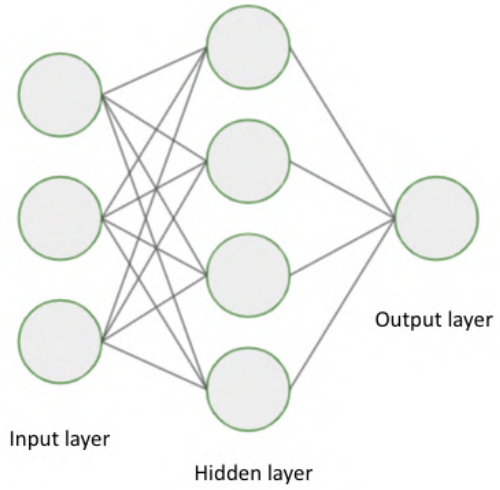

Figure 3.3.9 - Example of simple Neural Network with one hidden layer

3.3.2. Convolutional Neural Networks

Convolutional neural networks are a type of neural network used to process data with a grid-like topology (Goodfellow et al., 2016, p.330). The most blatant example of this type of data are images, as an image can be represented as a matrix of pixel values, but CNN's have been used successfully in different kinds of applications (Goodfellow et al., 2016, p.330). A CNN has hidden layers called convolutional layers that differentiates them from deep artificial neural networks. The main reason for using CNN's over ANN for images is to reduce computational time as ANN's are not spatially aware and treat pixels that are far away from each other the same way. Hence, ANN's are very sensitive to the location of an object in an image. CNN's reduce this complexity with an operation called convolution.

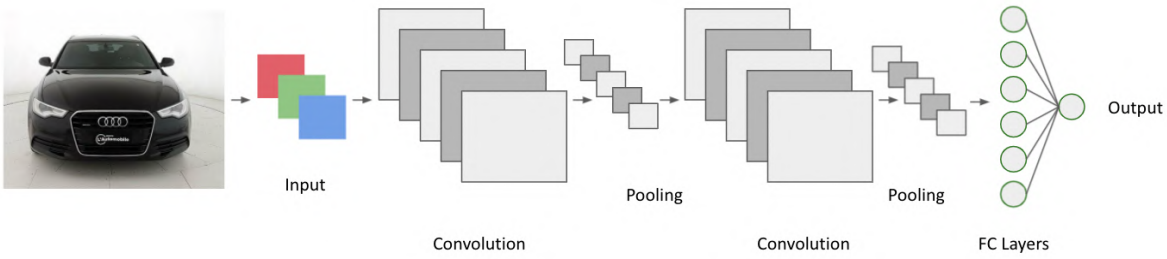

Figure 3.3.10 - Example of CNN architecture for regression

3.3.2.1. Convolution

A convolution is a mathematical function that allows two sets of information to merge together (Patterson et al., 2017, p.131). The main parts of the convolution layer are the input data, the kernel, and the feature map, which is nothing but the output layer (Goodfellow et al., 2016, p.332). The convolution layer takes as an input the raw data, and to understand and extract information from it, slides a filter (or kernel) on it to produce an output. At every step a dot product of a portion of the input image and the kernel is computed resulting in the output layer. These filters can understand patterns such as a line, a circle, or an edge. The main goal of the convolution layer is to extract features such as lines or edges from the input image.

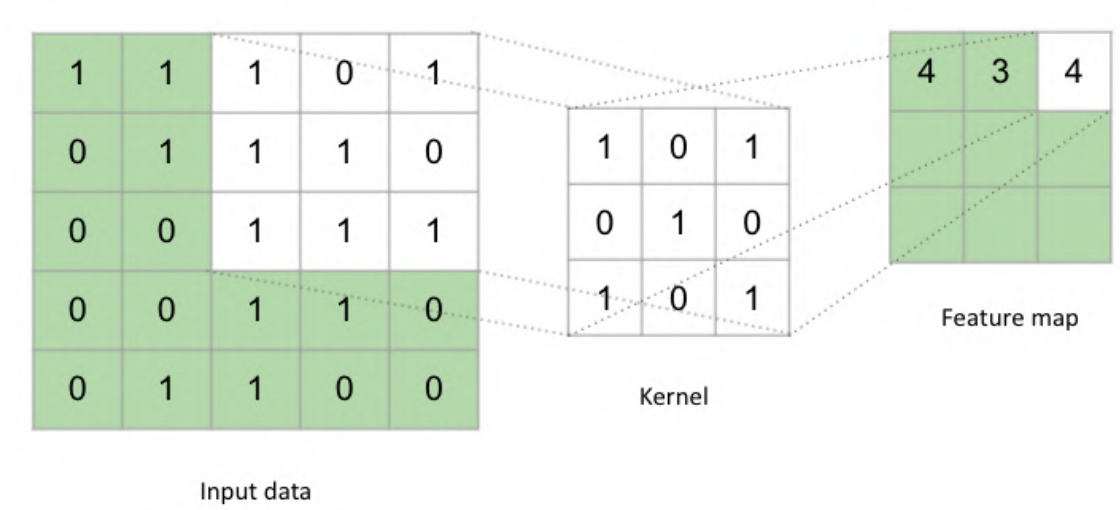

Figure 3.3.11 - Example of convolution

3.3.2.2. Pooling

Polling layers are usually found between convolutional layers and they are used to further downsample the spatial size of the input map and reduce the computational power to process it by making the model less sensitive to small changes in the input (Patterson et al., 2017, p.140). There are two different types of pooling operations: max pooling and average pooling. The max pooling operation takes the maximum value from the portion of the image at every step while the average pooling outputs the average of all the values within the sliding window.

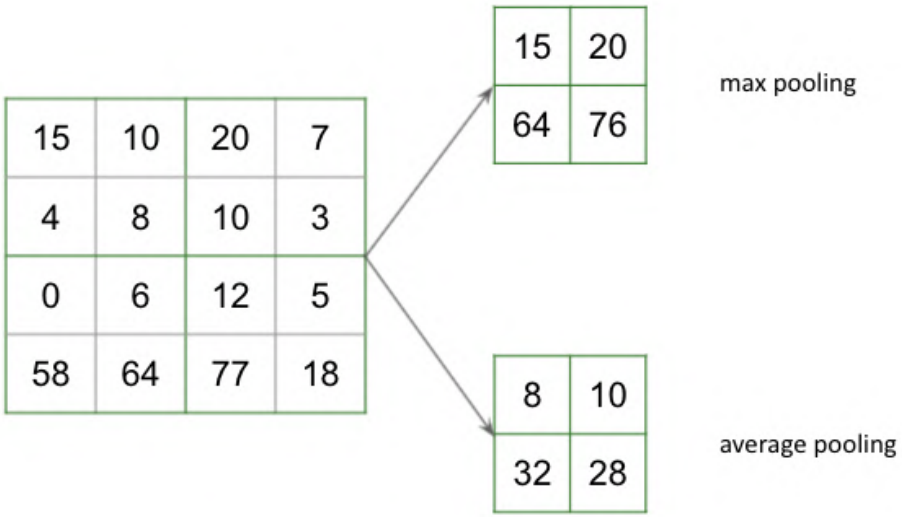

Figure 3.3.12 - Pooling operations

4. DATA

At the beginning of chapter 3, a thorough explanation of the main concepts for web scraping was done, and the main functions to extrapolate the data of interest from the online marketplace were described. To finally retrieve the data and call the functions *download_data* and *download_images*, a list of URLs to every single car listing had to be generated. In figure 4.1 the Python code to fetch all the URLs can be observed. In line 5, a for loop is initiated to go through a determined number of pages of the online marketplace. The URL of every single page containing a list of car listings can easily be generated by concatenating the root of the URL and the number returned by the range function (line 8). In line 9, a request is sent, and in line 10, Beautiful soup is again exploited to collect the page content. In line 13, a new for loop is initiated and every single *div* containing the car information is looked for. The attribute *data-link* containing the link of the car listing can now be stored in a variable (line 15). Finally, in line 16, the full URL is manually generated and finally appended to the list created in line 3.

```
1  def get_car_urls():
2
3    L= []
4
5    for page in range(1,100):
6
7        #creating link for every page
8        url ='https://www.automobile.it/usate/page-' + str(page)
9        req = requests.get(url)
10       soup = BeautifulSoup(req.text, 'lxml')
11
12       #look for car listing in every page
13       for listing in soup.find_all('div', {"class" : "jsx-1937958662 Card hover-effect CardAd"}):
14             try:
15                 link= listing['data-link']
16                 link = 'http://www.automobile.it' + str(link)
17                 L.add(link)
18             except:
19                 pass
20
21     return L
```

Figure 4.1 - Function to get all car URLs

Now that every single listing is directly accessible scraping can start. In figure 4.2, the function scrape can be observed. In line 3, a for loop iterating over the list of URLs is initiated. In line 5, every URL is split to store the car-id number in a variable and from line 6 to 8 a new folder destined to contain the download images is created in a predefined path per carid. In lines 10 and 12, the main functions are finally called and the scraping can start.

```
def scrape(L):

  for car in L:

      carid= car.split('/')[-1]
      SAVE_FOLDER= 'images/' + str(carid)
      if not os.path.exists(SAVE_FOLDER):
          os.makedirs(SAVE_FOLDER)
      #features
      download_data(car,carid)
      #car images
      download_images(car,carid)
```

Figure 4.2 – Function to run the scraper

The program was run overnight and took about 11 hours to collect all textual data and the images. After the scraping procedure, our dataset contains 750 cars and 13 features. The features selected and scraped for this work are the following:

- Car id - entry id assigned to the car on the trading platform
- Car link - URL to the car's listing
- Brand - brand of the car
- Model - model of the car
- Version - version of the car
- Year - year of production
- Mileage - mileage driven at the time of listing
- Seats - number of seats
- Gear - type of gear, whether automatic or manual
- Type - type of fuel
- Power - power of the car in horsepower
- Displacement - amount of air that goes through pistons in the engine
- Price - price of the car

Also, for every car, all the pictures available in the gallery for every single listing were downloaded. For some cars, only a few pictures were available, and for others, more than 10 per listing were available. Price is the target variable.

4.1. DATA CLEANING

Most machine learning algorithms do not accept missing data as input. The dataset contains many duplicates due to the scraper writing occasionally the same entry twice. These were simply dropped. After this procedure, the number of unique cars is 730. The only feature that contained missing data is the car version. As the missing rows equal 323, which is ~45 % of the dataset. This column was simply dropped. In this case, imputing the missing entries is not an appealing option as it would introduce bias as the number of fabricated values is too high. The current dataset consists of 730 rows and 12 columns.

4.2. DATA VISUALIZATION

Data visualization allows us to better understand the data by representing it graphically. In the boxplot below, it can be seen that most car prices fall under the range 5000 to 25000 €. This is the interquartile range (IQR), and it is where 50 % of the cars reside. There is the presence of two obvious outliers. These 2 cars are luxurious vehicles that even though are second-hand can retain a high resale value. These 2 rows were also dropped to avoid issues while evaluating the model. The dataset now contains 728 rows.

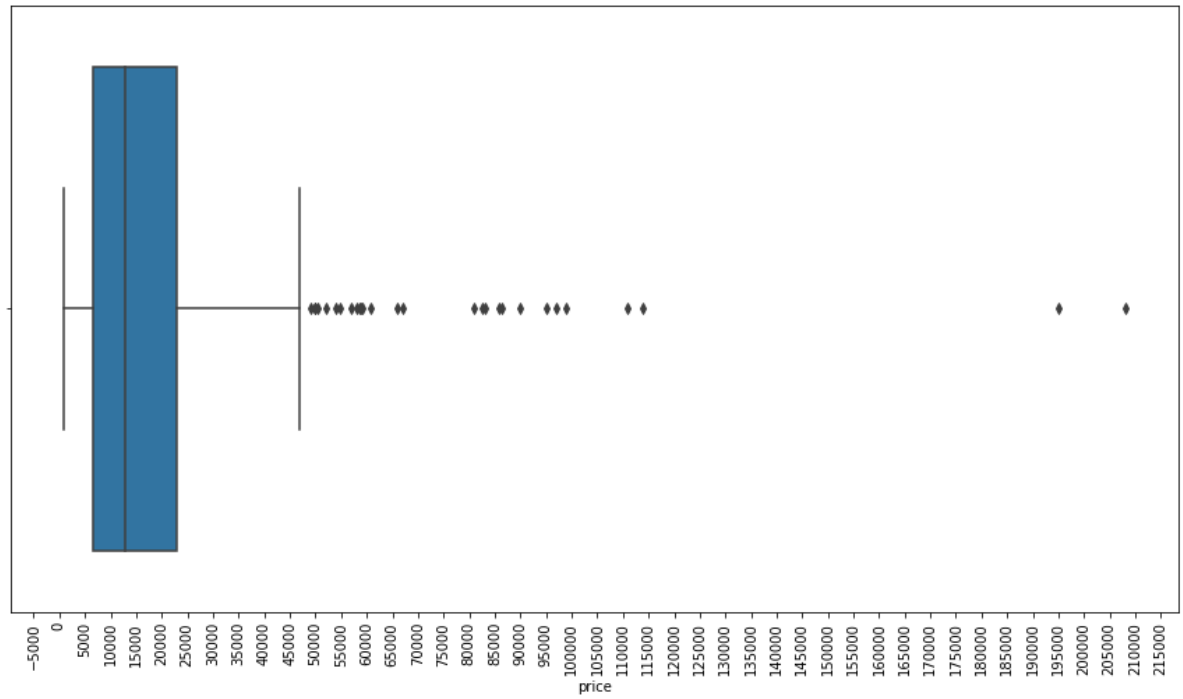

Figure 4.3 - Boxplot representing the distribution of car prices

4.3. CATEGORICAL VARIABLES

Machine learning algorithms can only be fed numerical data as input. Any categorical feature has to be encoded into numbers before training the phase. In our dataset, several features need encoding, such as brand and car model. There are different ways to encode these features, for instance, One-Hot-Encoding and Label Encoding. Label encoding means that every unique value of the categorical feature is assigned to a number whereas in One-hot encoding every unique value is transformed into a feature vector of *n* binary values where *n* is the number of unique values. (Burkov., 2019, p.4).

volvo = [1,0,0]	*volvo = 1*
opel = [0,1,0]	*opel = 2*
ford = [0,0,1]	*ford = 3*

Figure 4.4 - Example of One-hot encoding (left) and label encoder (right)

For this work, a one-hot encoder is preferred as label encoding entails risks such as increasing dimensionality of the feature space by implying an order for the encoded feature. (Burkov., 2019, p.4). If that order does exist then label encoding is indeed the best option, but if it does not exist then the model would try to understand the relationship between the car and the ascending order possibly leading to overfitting (Burkov., 2019, p.4). In simpler words, it does not matter whether Volvo is assigned to 1, ford to 3, or vice versa, the machine learning model will try to figure out why the value of ford is higher than the value of Volvo. For this work, OneHotEncoder from the scikit learn library was used.

4.4. IMAGE PROCESSING

After collecting all the images, the first step was to look into every folder, filter out bad quality pictures and most importantly label the images. As can be seen in the picture below, the car images do not have a label. This means that we will have to manually rename and select every single picture. For this work, it was decided to label the front, left-side, right-side, and back of the vehicle.

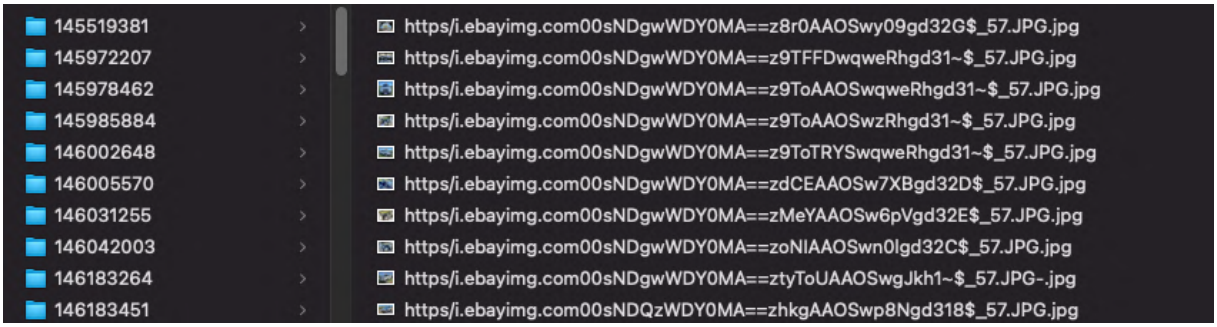

Figure 4.5 - Raw scraped pictures

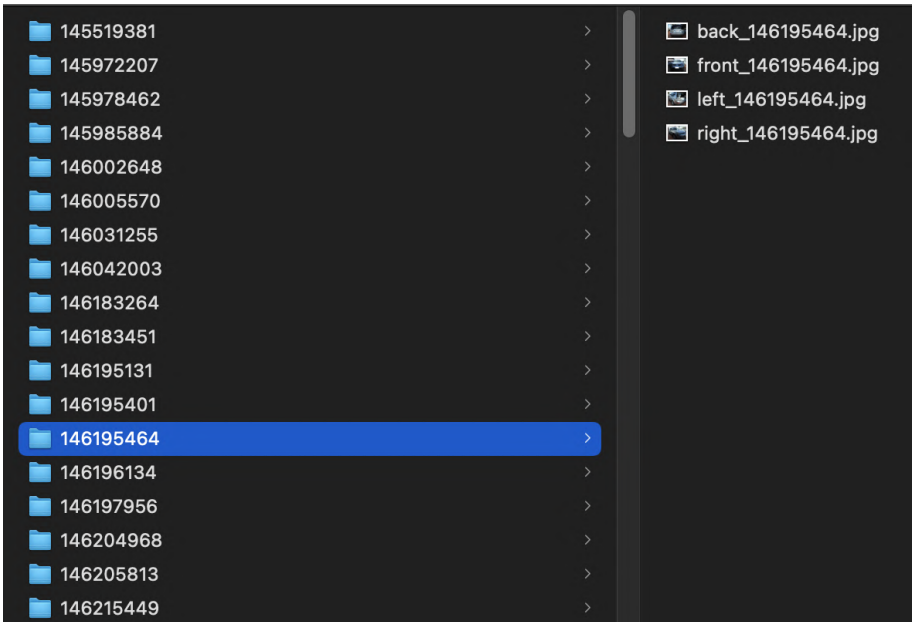

Figure 4.6 - Manually labelled pictures

Figure 4.7 - Example of selected pictures per used car

In order to use all scraped images per car, a function was developed to build a collage of 4 pictures and organize the pictures for training. First of all, the labelled pictures are moved into a new folder, with all the other training pictures.

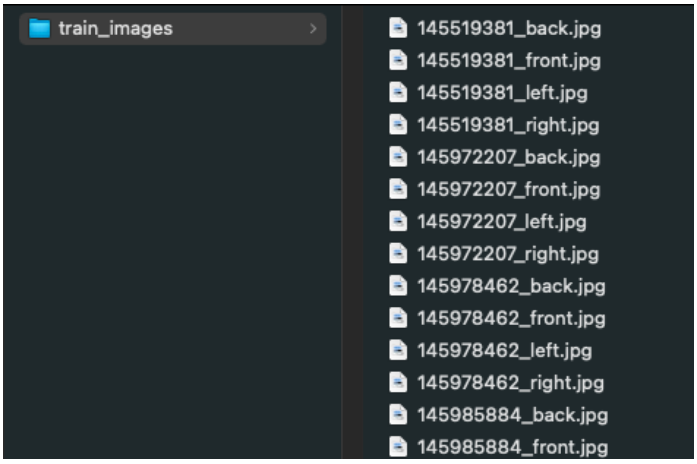

Figure 4.8 - Structure of training folder

To further clean the images, some of the present information that would confuse or make it harder for the model to learn is cropped out. Many second-hand car trading websites for example include watermarks with their logo, phone numbers, or other irrelevant information on the pictures they list on the website. Also, some pictures happen to have other cars present in the background or foreground. This noise is simply removed from the picture when possible.

In a loop, every car id in the folder is iterated, sorted and the pictures for the corresponding id resized into 128 * 128 pixels and organized in a 256*256 numpy array. The back side of the car goes on the top-left side, the front side on the top-right side, the left side on the bottom-left side, and the right side on the bottom-right side. Every collage is then appended in an empty list and the list is returned as a numpy array.

As already mentioned above, images have to be the same size to be fed into a CNN. It is not usually the case that pictures come with same width and height. Resizing the images to fit a specific format, in our case, 128*128 pixels, will inevitably stretch and deform them. Some researchers argue that if the input images are not too stretched or squeezed the CNN should still be able to predict accurately. However, recent studies explored how zero padding does not compromise prediction accuracy and actually reduces training time (Heshami, 2019, page 1). In convolutional neural networks, zero padding is a technique that is used to surround a matrix with pixels of value zero. In simple words, zero-padding allows to control the resizing of an image to make it fit into the desired size without losing any information by keeping its aspect ratio and adding black borders to fill up the empty space outside the image. The CNN will learn that this information is not important and the input units of these zero values are therefore not be activated. (Heshami, 2019, page 1)

$$X = \begin{bmatrix} a & b & c \\ d & f & g \\ h & j & k \end{bmatrix} \qquad Pad(1,X) = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & a & b & c & 0 \\ 0 & d & f & g & 0 \\ 0 & h & j & k & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Figure 4.9 - Mathematical representation of zero padding

```python
def load_car_images(data, inputPath):

    images = []
    # loop over the indexes of the cars
    for i in data.carid.values:
        # find four car images for the carid and sort them
        basePath = os.path.sep.join([inputPath, "{}_*".format(i)])
        carPaths= sorted(list(glob.glob(basePath)))

        # initialize our list of input images along with the output image
        inputImages = []
        outputImage = np.zeros((256, 256, 3), dtype="uint8")
        # loop over the input car paths
        for car in carPaths:
            # load the input image, resize it to be 128 128, add padding
            # update the list of input images
            image = Image.open(f'{car}')
            image = make_square(image,min_size=256, fill_color=(0, 0, 0, 0))
            image = asarray(image)
            image= cv2.resize(image, (128, 128))
            inputImages.append(image)

            outputImage[0:128, 0:128] = inputImages[0]
            outputImage[0:128, 128:256] = inputImages[1]
            outputImage[128:256, 128:256] = inputImages[2]
            outputImage[128:256, 0:128] = inputImages[3]

        # build the collage
        images.append(outputImage)
    # return our set of images
    return np.array(images)
```

Figure 4.10 - Function to load and process the car images

Figure 4.11 - Example of input for the CNN

Finally, all pictures are normalized by simply dividing them by 255 and bringing them into a range between 0 and 1.

4.5. Feature Selection

4.5.1. Boruta Algorithm

Boruta is a feature selection algorithm that performs a top-down search and compares the features with a permuted copy of themselves, also called “shadow features”. Features that are less statistically relevant than the random features are deemed to be not important and therefore rejected. Boruta is a wrapper feature selection method built around the Random Forest classification algorithm (Kursa M., 2010).

Boruta maps the features to three different areas:

- Green Area: the features are confirmed to be important and must be kept
- Red Area: the features than land here are statistically not significant and shall be dropped
- Blue Area: These features are in a limbo, they might help the model improve accuracy and are to be manually tested.

```
from boruta import BorutaPy
from sklearn.ensemble import RandomForestRegressor
###initialize Boruta
forest = RandomForestRegressor(
    n_jobs = -1,
    max_depth = 50
 )
boruta = BorutaPy(
    estimator = forest,
    n_estimators = 'auto',
    max_iter = 50 #number of trials to perform
)
### fit Boruta
boruta.fit(np.array(X1_train), np.array(y_train))
### print results
green_area = X1_train.columns[boruta.support_].to_list()
blue_area = X1_train.columns[boruta.support_weak_].to_list()
print('features in the green area:', green_area)
print('features in the blue area:', blue_area)
```

Figure 4.12 – Boruta in Python

features in the green area: ['brand','year', 'mileage', 'power', 'displacement','gear','type']

features in the blue area: ['seats']

features in the red area = ['model']

Boruta rejected the car model from the valuable features. This makes sense as almost every car in the dataset has inherently a different model and the descriptions in the listings are sometimes ambiguous. It would be complicated for a model to find a pattern with a limited dataset. For the time being, this feature is dropped, but it could become useful if the dataset were to grow.
Interestingly, the number of seats is in the blue area. This means it could or could not help improve the model. The number of seats is nonetheless an important feature that is usually very much correlated with car prices. Fewer seats means less space and by looking at the second-hand car market it is obvious that a car with 4 seats is in most cases less expensive than a car with 5 seats. This feature is therefore kept.

The final dataset consists of 728 rows and 12 columns.

4.6. DATA NORMALIZATION

Data is usually in different scales or measured in different units. It has been proved in the literature that this can lead machine learning algorithms to not perform well and training time to increase. Feature values that reside at higher levels of magnitude, lead to extremely high values when calculating network outputs in comparison to smaller feature values. (Sola et al, 2017)
For this work, we have for example mileage, from 1900 (km) to 360000 (km) and year, from 1999 to 2020. This process is typically called feature scaling. (Aurelien Geron, 2019, p.106). There are two main methods to bring data at the same scale: Standardisation and Normalisation.

4.6.1. Normalization Min-Max

Normalization or Min-max scaling is the most straightforward way to approach this issue. It works by subtracting the min value to our input and dividing it by the max observed value minus the minimum observed value, resulting in output ranging between 0 and 1. (Aurelien Geron,2019, p. 107)

$$Xnorm = \frac{X - Xmin}{Xmax - Xmin}$$

Where X is the input value, Xmin is the minimum value of the feature and Xmax is the maximum value of the feature.

Images also need to be normalized before training. As mentioned above, neural networks do not perform well when the input data is not in the range [0,1]. An Image is a matrix of pixels with value intensity ranging from 0 to 255. Normalization can be achieved by simply diving by 255.

4.6.1.1. Standardization

Standardization is a scaling technique that works by subtracting the mean value of an input feature and then dividing it by the standard deviation.

$$z = \frac{x - \mu}{\sigma}$$

Where u is the mean and delta is the standard deviation.
Standardization is much less affected by outliers than normalization because it does not bound values to a specific range. Some algorithms, for instance, neural networks, do not behave well with this kind of range as they are expecting input values in the range from -1 to 1 or 0 to 1. (Aurelien Geron,2019, p.107)

5. METHODOLOGY

5.1. MODEL EVALUATION

When developing a model, it is not wise to use the same data it is built on to assess its accuracy. A model that is evaluated on training data would very likely be biased and not able to generalize well on unseen data. For this work, Hold-out validation is used. Hold-out validation or train-test split validation is the easiest and most straightforward method for model evaluation. It's very easy to implement as there are already many libraries such as scikit-learn that implement it in a single line of code. Hold-out validation simply means splitting the dataset into a training set and a test set. The general sizes are 80% for training and 20% for testing. Nonetheless, these are just rules of thumb as a very large dataset could also have a much smaller percentage of data set aside for testing and still contain a very large amount of samples. (Geron, 2019, p.58) If the dataset is very large then usually this technique is enough to enclose a good enough representation of data regardless of the test size. For this work, the main split is 75% for training and 25% for testing. The test set is furtherly split into 2. The first 12,5% is used for validation whereas the other 12,5% for the actual final testing. The test data is not used at all while developing the model, the validation set can be considered part of the training data as it is a dataset upon which the models are compared and the best model selected.

Dataset	Size
Train	546 (75%)
Validation	91 (12,5%)
Test	91 (12,5%)

Table 5.1 - Shape of the dataset

5.2. EVALUATION METRICS

Machine learning tasks require performance metrics to evaluate how well a model is performing on a dataset (Alice Zheng, p 4). For, example when trying to predict a numeric target, an appropriate metric for the task would be the root-mean-square error (RMSE), whereas when classifying a binary outcome such as if a patient is at risk of cancer or not, a classification performance metric like accuracy, precision, recall or F1-score would be more appropriate. (Alice Zheng, p 4). As this work is trying to predict the price of second-hand cars, the focus will be on evaluation metrics for regression tasks.

5.2.1. Mean Squared Error

Mean squared error (MSE) is a metric for regression tasks. It is not very used in practice in its pure form as the output is not directly interpretable. The calculation is the average of the squared difference between the actual value and the predicted value.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_j - \hat{y}_j)^2$$

Where yi denotes the real price of a second-hand car whereas ŷi denotes the predicted car price. n is the number of cars used for training.

5.2.2. Mean Absolute Error

Mean absolute error (MAE) is the sum of all absolute errors yielding the total error, the sum is then divided by the number of data points. (Willmott et al, 2005). Or in other words, the average of the absolute difference between the real prices and the predicted prices of the cars of the test set. MAE is arguably the easiest metric for a regressions task. It can be read as how far on average a prediction is from its actual value.

$$MAE = \frac{1}{n}\sum_{j=1}^{n}\left|y_j - \hat{y}_j\right|$$

Where yi denotes the real price of a second-hand car whereas ŷi denotes the predicted car price. n is the number of cars used for training

5.2.3. Root Mean Square Error

Root mean Square Error (RMSE) is probably the most frequently used metric for regression tasks and it is defined as the square root of the average squared distance between the real value and the predicted value. (Alice Zheng, p 15). RMSE is based on MSE, the only difference is that it also takes the root of the MSE, to make it more interpretable and bring the result back on the same scale as the target feature in the dataset.

RMSE is always a positive number and a lower value yields a much better model and fit to the data. The closest to zero the better with zero meaning a perfect fit. RMSE can be misleading, a large prediction error will have a big effect on the error value. This makes RMSE very sensitive to outliers and therefore can lead to mistakes in its interpretation when the data has not been through a thorough

cleaning process. (Alice Zheng, p 15). This sensitivity to outliers can also be beneficial as it allows debugging the model by detecting their presence.

$$RMSE = \sqrt{\frac{1}{n}\sum_{j=1}^{n}(y_j - \hat{y}_j)^2}$$

Where yi denotes the real price of a second-hand car whereas ŷi denotes the predicted car price. n is the number of cars used for training

5.2.4. Mean Absolute Percentage Error

Mean absolute percentage error (MAPE) is very similar to MAE and a much more robust estimator compared to RMSE (Alice Zheng, p 15). MAPE is the average of the absolute percentage error of the actual values and the predicted values. This metric is very easy to interpret as it yields an error in percentage. It can be read as how far in percentage a prediction is from its actual value.

$$MAPE = \frac{1}{n}\sum_{j=1}^{n}\left|\left(y_j - \hat{y}_j\right)/y_j\right|$$

Where yi denotes the real price of a second-hand car whereas ŷi denotes the predicted car price. n is the number of cars used for training

5.3. Machine Learning Methods

In this section, several machine learning models will be applied to the dataset to compare which one performs better and to establish a baseline for later comparison with deep learning models and mixed-input models.

```python
def baseline(X_train, y_train, X_val, y_val):

  models = [
            ('LR', LinearRegression()),
            ('DT', DecisionTreeRegressor()),
            ('RF', RandomForestRegressor()),
            ('KNN', KNeighborsRegressor()),
           ]

  for name, model in models:
          regressor = model.fit(X_train, y_train)

          y_pred = regressor.predict(X_val)
          print(f'model: {name}')
          print('----------------')
          print('metrics --------')
          metrics(y_val,y_pred)
          print('----------------')

```

Figure 5.1 - Function to apply different machine learning algorithms and evaluate them

As can be seen in figure 5.1 the ML models are applied with their default parameters. Once the *baseline* function is called for every model the validation set will be predicted and metrics evaluated.

```
model: LR
----------------
metrics --------
RMSE = 9693.2392578125
MAPE = 0.5872461795806885
----------------
model: DT
----------------
metrics --------
RMSE = 8382.845483651585
MAPE = 0.28563529042515173
----------------
model: RF
----------------
metrics --------
RMSE = 6980.109946989922
MAPE = 0.25485472798347473
----------------
model: KNN
----------------
metrics --------
RMSE = 8141.322265625
MAPE = 0.2634193640947342
----------------
```

Figure 5.2 - Results of Machine learning models

As can be observed in fig 5.2 Random Forest provides the best baseline with an RMSE of 5671 and a MAPE of ~25% and is now selected as the best machine learning model. To find the best parameter for this model, the Grid-Search module from Scikit-learn was utilized. A manual dictionary of parameters is passed to the grid search that builds a different model with every single combination of them and compares the scores on the validation data. The model that performs best will then be used as the final model for prediction on the test set. This operation is computationally intensive. In the picture below the chosen parameters, the most important of them being the number of estimators, namely the number of trees.

```
from sklearn.model_selection import GridSearchCV

parameters = {
    'max_depth': [80, 90, 100, 110],
    'max_features': [2, 3, 4, 5,6],
    'min_samples_leaf': [3, 4, 5],
    'min_samples_split': [8, 10, 12],
    'n_estimators': [100, 200, 300, 1000]
        }

regressor = GridSearchCV(rf, parameters,verbose=True,n_jobs = -1)
regressor.fit(X_train, y_train)
rf_predictions = regressor.predict((X_val))

best_estimator =regressor.best_estimator_
```

Figure 5.3 - Hyperparameters tuning with Grid search

```
RandomForestRegressor(bootstrap=True, ccp_alpha=0.0, criterion='mse',
                      max_depth=110, max_features=5, max_leaf_nodes=None,
                      max_samples=None, min_impurity_decrease=0.0,
                      min_impurity_split=None, min_samples_leaf=3,
                      min_samples_split=8, min_weight_fraction_leaf=0.0,
                      n_estimators=100, n_jobs=None, oob_score=False,
                      random_state=None, verbose=0, warm_start=False)
```

Figure 5.4 - Best estimator for RF Regressor

The Random Forest regressor with the best parameters is now trained on all data and used to predict the test set.

```
RMSE = 5043.6774326138
MAPE = 0.24100516246581533
```

Figure 5.5 - Results on the test set for RF regressor

With hyperparameter tuning it was possible to improve to model results, improving the MAPE from ~25% to ~24%. These results will now be compared with deep learning models in the next chapter.

5.4. Deep Learning Methods

5.4.1. Training on numerical features

In this section, the application of a feedforward neural network with TensorFlow Keras is explored. A simple architecture with one layer is implemented and adapted to the Keras tuner library for parameter optimization. As a first step, we will define parameters such as the number of units the first hidden layer should have, the optimizer type, and the learning rate.

An Early stop callback parameter is also defined. This allows the training to stop automatically when the monitored metric is not improving anymore after a certain amount of epochs, avoiding overfitting the data.

```
def build_model_ann(hp):
    model = keras.Sequential([

          keras.layers.Dense(hp.Int('num_units', min_value=8, max_value=16, step=2),activation='relu'),
          keras.layers.Dense(1)
    ])

    optimizer_name=hp.Choice('optimizer', values=['adam', 'sgd'])

    if optimizer_name == "adam":
        optimizer = tf.keras.optimizers.Adam(hp.Choice('learning_rate', values = [1e-2, 1e-3, 1e-4]))
    elif optimizer_name == "sgd":
        optimizer = tf.keras.optimizers.SGD(hp.Choice('learning_rate', values = [1e-2, 1e-3, 1e-4]))
    else:
        raise ValueError("unexpected optimizer name: %r" % (optimizer_name,))

    #compile
    model.compile(
                optimizer = optimizer,
                loss='mean_absolute_percentage_error',
                metrics=['mape']
                )
    return model
```

Figure 5.6 - ANN architecture with Keras tuner

There are different tuning algorithms available for Keras tuner. For this work, Random Search is used. Random Search is one of the most straightforward tuning algorithms as it simply tries out different parameters randomly within the options given and selects the ones that provide the best results. In Figure 5.7 it can be seen how a Random Search instance is created and the parameter optimization started. A *n* number of trials can be selected which means that *n* different combinations of parameters are performed.

```
#importing random search
from keras_tuner import RandomSearch
#creating randomsearch object
tuner_ann = RandomSearch(build_model_ann,
                     objective='mape',
                     max_trials = 10,
                     overwrite=True
)
# search best parameter
tuner_ann.search(X1_train,y_train,
              validation_data=(X1_val, y_val),
              epochs=50,
              callbacks=[tf.keras.callbacks.EarlyStopping(patience=5)])
```

Figure 5.7 - Random Search in Keras tuner

Once the process is over, the *get_best_hyperparameters* method can be called to retrieve the best parameters.

```
# best hyperparameters ann
bestHP = tuner_ann.get_best_hyperparameters(num_trials=1)[0]
print("[INFO] optimal number of filters in num_units layer: {}".format(
  bestHP.get("num_units")))
print("[INFO] optimal number of filters in optimizer layer: {}".format(
  bestHP.get("optimizer")))
print("[INFO] optimal learning rate: {:.4f}".format(
  bestHP.get("learning_rate")))
```

```
[INFO] optimal number of filters in num_units layer: 12
[INFO] optimal number of filters in optimizer layer: adam
[INFO] optimal learning rate: 0.0010
```

Figure 5.8 - Best ANN Parameters

Now the model is retrained with the best parameters selected and with more epochs on all data.

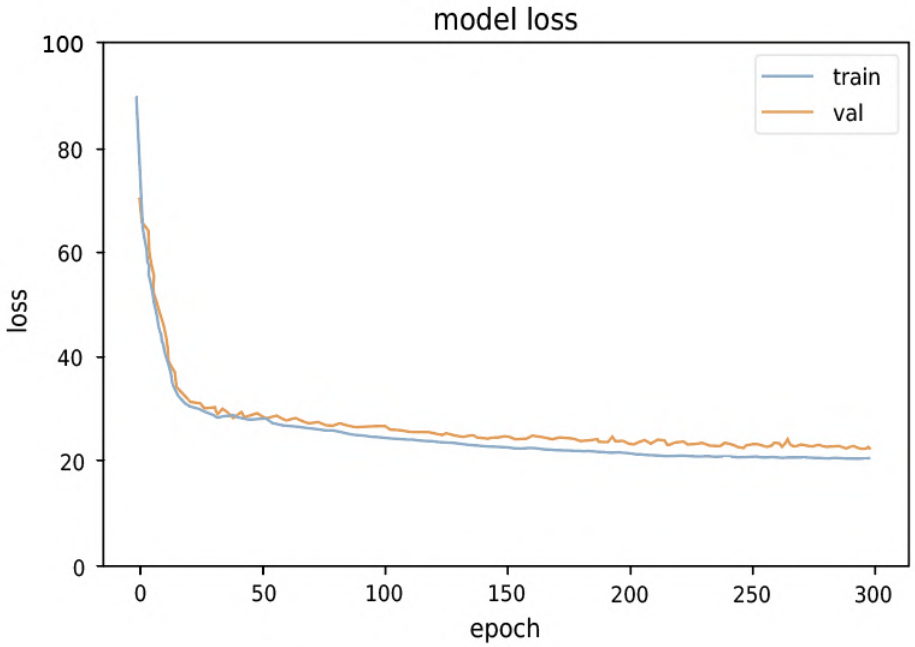

Figure 5.9 - ANN train and validation loss

As the graph shows, the callback function stopped the training around epoch 300 out of 500, avoiding the risk of overfitting the data.

With the best configuration of the neural network selected, the model is now used to perform the prediction on the test data. This process is the same done earlier with the machine learning methods. The RMSE and MAPE for the whole test data are:

```
RMSE = 3994.29236823894146
MAPE = 0.21678323133568221
```

Figure 5.10 - ANN results

5.4.2. Training on image data

In this section, a convolutional neural network based solely on image data is developed and tested to assess the capability of a CNN to predict car prices without numerical and categorical features. In order to utilize a CNN for regression in Keras, a few adjustments need to be made compared to the many architectures that are available for classifications tasks. Simply speaking, a convolutional neural is a series of Convolution and Pooling layers stacked on top of each other. The output of these operations is a 3D tensor that is fed into fully connected layers, or dense layers, in order to make prediction by the network possible. (Francois Colleit,2018,p.120). For image classification tasks, this generally means adding an activation function such as softmax, that calculates the probability of each input to belong

to one of the input classes. The highest probability in the probability distribution is considered as the actual output, namely the class to which that input belongs. In a regression task, such as price estimation, there are no classes, but the output will be a continuous value. Since there is no need to infer a class and therefore transform the actual values into a probability, a linear activation function with one node is used as the last layer of the CNN. Also, the loss function has to be changed accordingly to fit a regression task. As already mentioned in the chapter about evaluation metrics, common choices for regressions are either MAPE or RMSE.

Rather than trying different types of parameters empirically, Keras tuner library is used here also for hyperparameter optimization. A simple architecture with 2 Convolution layers, 2 Pooling layers, a fully connected layer, and an output layer is created and adapted to run with Keras tuner for parameter tuning. The CNN model adapted for Keras tuner can be observed in figure 5.11

```
def build_model(hp):
    model = keras.Sequential([
    #convolutional layer 1
    keras.layers.Conv2D(
        filters=hp.Int('conv_1_filter', min_value=32, max_value=64, step=16),
        kernel_size=hp.Choice('conv_1_kernel', values = [3,5]),
        activation='relu',
        input_shape=(256,256,3)
       ),
    #pooling 1
    keras.layers.MaxPool2D(pool_size=(2,2)),
    #convolutional layer 2
    keras.layers.Conv2D(
        filters=hp.Int('conv_2_filter', min_value=64, max_value=128, step=16),
        kernel_size=hp.Choice('conv_2_kernel', values = [3,5]),
        activation='relu'
    ),
    #pooling 2
    keras.layers.MaxPool2D(pool_size=(2,2)),
    # flatten layer
    keras.layers.Flatten(),
    # dense layer
    keras.layers.Dense(
        units=hp.Int('dense_1_units', min_value=16, max_value=128, step=16),
        activation='relu'
    ),
    # output layer
    keras.layers.Dense(1, activation='linear')
    ])
    #compile
    model.compile(
                optimizer=tf.keras.optimizers.Adam(
                  hp.Float('learning_rate', 1e-4, 1e-2, sampling='log')),
                loss='mean_absolute_percentage_error',
                metrics=['mape']
                )
    return model
```

Figure 5.11 - CNN with Keras tuner

For this CNN model, random search is also used and the different 10 number of trials performed.

```
# grab the best hyperparameters
bestHP_cnn = tuner.get_best_hyperparameters(num_trials=1)[0]
print("[INFO] optimal number of filters in conv_1 layer: {}".format(
  bestHP_cnn.get("conv_1_filter")))
print("[INFO] optimal number of filters in conv_2 layer: {}".format(
  bestHP_cnn.get("conv_2_filter")))
print("[INFO] optimal number of units in dense layer: {}".format(
  bestHP_cnn.get("dense_1_units")))
print("[INFO] optimal learning rate: {:.4f}".format(
  bestHP_cnn.get("learning_rate")))
```

```
[INFO] optimal number of filters in conv_1 layer: 48
[INFO] optimal number of filters in conv_2 layer: 128
[INFO] optimal number of units in dense layer: 80
[INFO] optimal learning rate: 0.0057
```

Figure 5.12 - CNN best parameters

As it can be observed in figure 5.12, for the first convolutional layer the number of optimal units is 48, whereas for the second is 128. This makes sense, as the network goes deeper, the patterns to learn become more complex. The optimal number of units for the dense layers is 80 and the best learning rate is 0.0057.

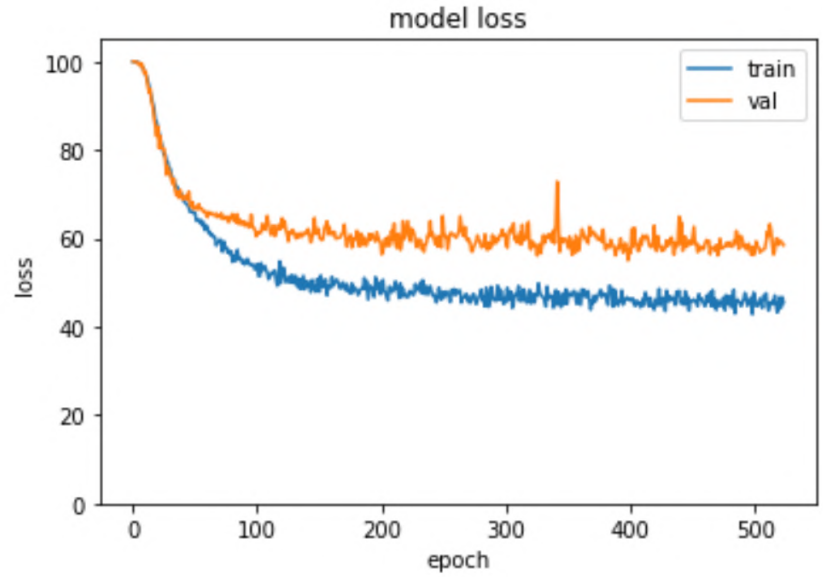

Figure 5.13 - CNN train and validation loss

The model is now trained on all data and the model is ready to predict the test set. The MAPE for the whole test data is:

```
RMSE = 15070.910156256143
MAPE = 0.6141334867477417
```

Figure 5.14 - CNN results

The final result yields an RMSE of 15070 and a MAPE of ~61%. By looking at the model loss in figure 5.13 it is clear that the CNN is underfitting. Both training loss and validation loss are high and the model performs poorly on the validation data and test data. As expected, the use of only image features to predict car prices does not behave as well as its counterpart based on numerical and categorical features. The features to be learned from images for the problem at hand are too complicated and the network is struggling. More training images and a deeper architecture might be able to improve the results but there are many features such as mileage driven or type of gear that are intrinsically complicated to infer from image data. Nonetheless, many features can be learned from images that are not available as features in the original dataset. An example of this can be features regarding the vehicle frame that can easily be learned with enough training images.

5.4.3. Grad-CAM

Class activation Map (Grad-CAM) is a technique to make CNN more transparent and explainable (Selvaraju et al., 2019, p.1). It is based on its predecessor CAM, and improves upon it by being more adaptable to different kinds of state-of-the-art CNN's without modifying their architecture, and, more importantly, it doesn't require training the model again (Selvaraju et al., 2019, p.2). It uses the gradient information in the last convolution layer of the CNN, to visually explain where the model is looking to predict a class. The convolution layer is the last layer that retains spatial information and semantics of the input images (Selvaraju et al., 2019, p.4).

In simple words, Grad-Cam is used to debug and to better understand what the CNN is looking at when predicting a class, or for this work, a price. In the case of suspicious prediction, it is therefore easy to interpret what makes the model fail. For instance, if in our case the model is more active looking at an object in the background rather than at the cars it is clear that the model is not learning properly. Originally Grad-Cam was developed for image classification problems, but the same architecture can also be used for regression tasks as shown in other studies (Wang et al, 2019). The equivalent to GRAD-Cam for Regression tasks is Regression Activation Maps (RAM). Grad-cam outputs a class-activation heatmap that can be overlaid to the original input image for CNN explainability.

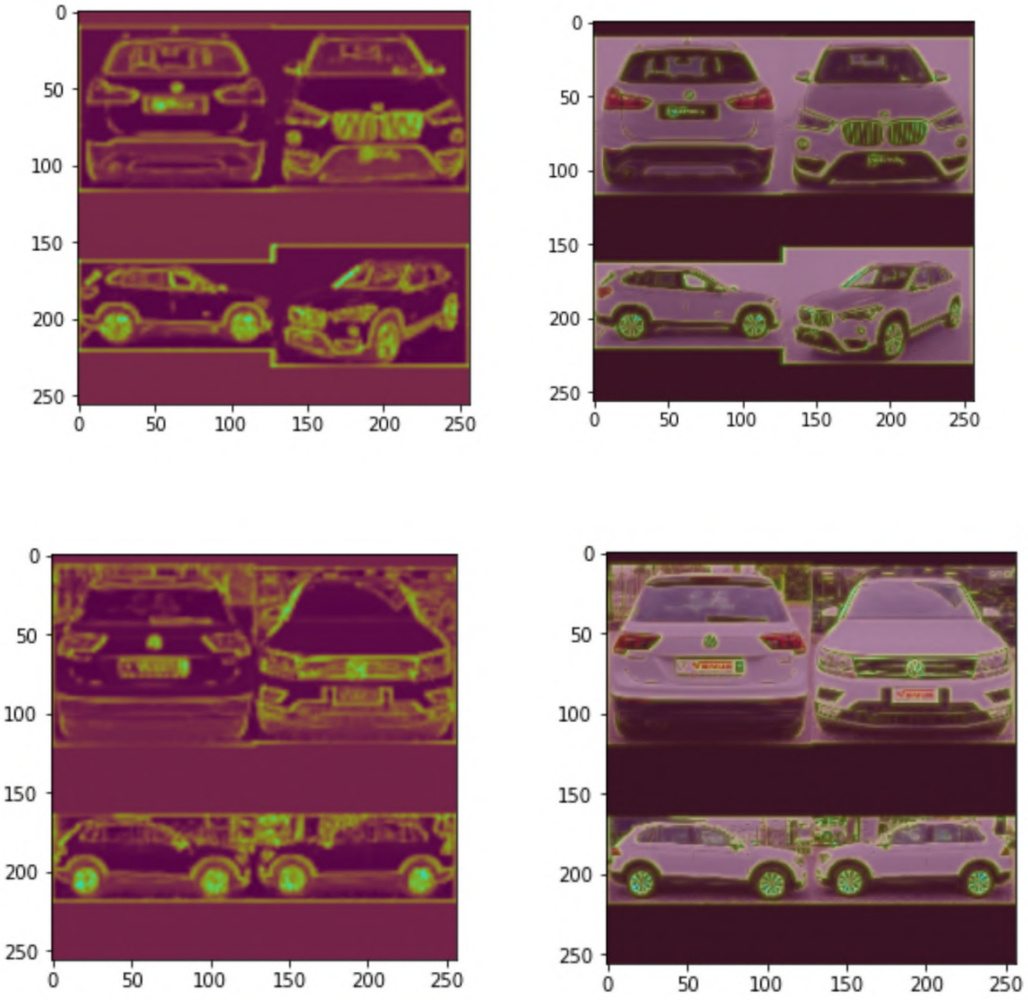

Figure 5.15 - GRAD-Cam output: heatmap (left) and original test image with overlaid heatmap (right)

Based on figure 5.16, looking at the first example, it seems that CNN is learning the right features when the images are high quality and noise-free. Looking at the second example it can be noticed how the network is not only picking up car features but also activating in the background part of the image. This indicates that the network is struggling with noisy images. A solution to this could be to either exclude all low-quality images from training or increase the total number of training samples to help the network learn only the right features. It can also be observed how zero-padding is working and the network is understanding that it should not give importance to that part of the image. Also, it appears that the CNN is looking at the car plates. This could be an issue as usually car plates in second-hand car images on online marketplaces do not carry any important information regarding the price of the car. In fact, car plates are usually replaced with the logo of the dealer. A solution to this problem would be blurring out all car plates from the images.

5.5. MIXED–INPUT MODEL

Hybrid Deep Neural Networks (Hybrid-DNNs) are a type of neural network architecture that can support multiple data types as input. The ability to accept different inputs makes such models more flexible and adaptable to different scenarios and allows more sophisticated feature extraction capabilities from the data (Zhenyu Yuan et al., 2019, p.1). Every data type is handled by a different network, the outputs of these networks are then concatenated and fed into another fully connected layer or the final prediction. Different machine learning methods can be applied to such architectures. In the case of images, CNNs will be the method to process the inputs, but the numerical data can be dealt with by a big variety of machine learning and deep learning algorithms such as KNN, Random Forest, or MLP. (Zhenyu Yuan et al., 2019, p.4). In this work, the TensorFlow Keras is used to develop all the deep learning models. Specifically, the functional API is used to develop the mixed-input model. The functional API is a way to build models that are much more flexible than the standard sequential API and have non-linear architectures, multiple input, or outputs.[14]

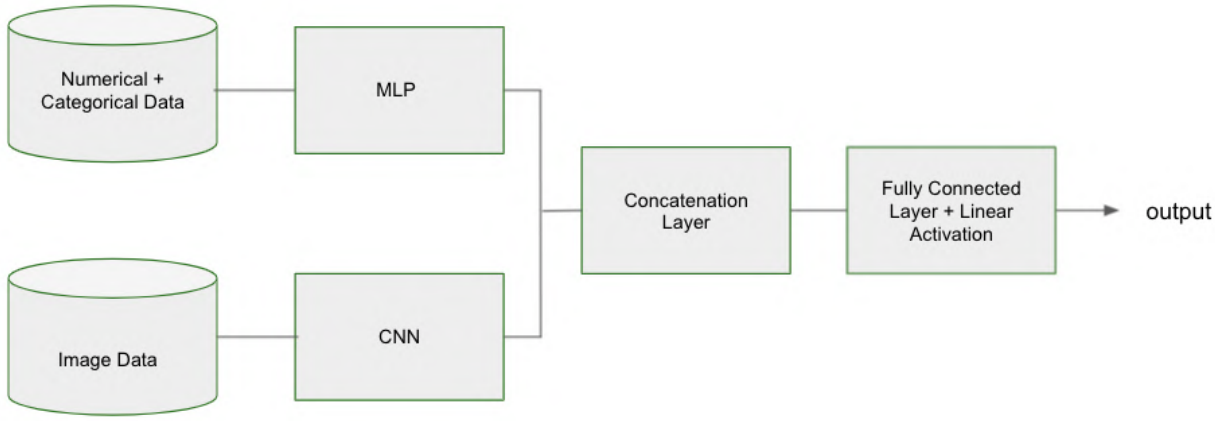

Figure 5.16 - Mixed-input model architecture

```
# build the MLP and CNN models
mlp = create_ann(X1_train.shape[1])
cnn = create_cnn(X1_train.shape[1], X1_train.shape[2], X1_train.shape[3])
# build the input of the last network layer as the output of MLP and CNN
combinedInput = concatenate([mlp.output, cnn.output])
#final FC layer with last layer as regression layer
x = layers.Dense(4, activation="relu")(combinedInput)
x = layers.Dense(1, activation="linear")(x)
#building the model
model = Model(inputs=[mlp.input, cnn.input], outputs=x)
opt = Adam(lr=1e-3)
#compile the model
model.compile(loss="mean_absolute_percentage_error", optimizer=opt)
# train the model
history_mixed = model.fit(
  x=[X1_train, X2_train], y=y_train,
  validation_data=([X1_val, X2_val], y_val),
epochs=2000, batch_size=8,verbose=1,callbacks=[earlystop])
# make predictions
predictions = model.predict([X1_val, X2_val])
metrics(y_val,predictions)
```

Figure 5.17 - Mixed-input model with Keras functional API

In figure 5.18 the Mixed-input model is developed with Keras functional API. In the first lines, the MLP and the CNN branches are built separately. Is it important to point out that in both branches the last layers are composed of a Dense layer of 4 units with no regression, that is, prediction is not yet done at this stage, but the outputs of both branches are concatenated into a single vector. The regression layer for prediction is then added later after the concatenation.

The model is then fit to the training data and learning can start. Early stopping is set to a value of 100 to avoid overfitting. For parameters, different learning rates have been tested. Namely 0.1 0.001,0.001. As smaller batch sizes tend to lead to better results, different parameters were tested. The final batch size was then set to 8, as it provided the best trade-off between accuracy and training speed.

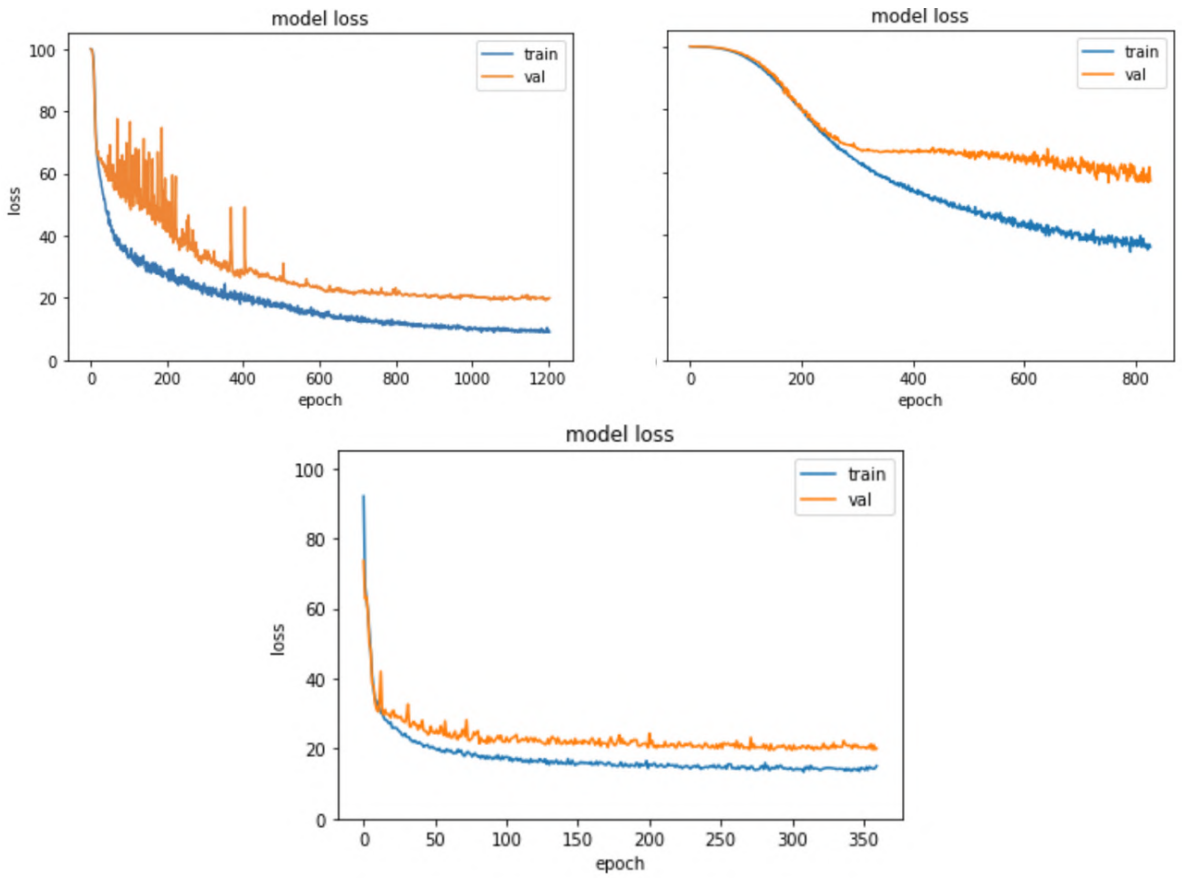

Figure 5.18 - Mixed-input model training and validation losses

In figure 5.19 the training and validation losses for the different learning rates can be observed. In the top-left figure, the learning rate is set to 0.001. It can be observed how the validation loss is initially very bumpy and becomes more stable around epoch 400. Succeedingly, validation and training loss go down steadily. At epoch 1200 early stopping is triggered. In the top-right figure, the learning rate is set to 0.0001. Validation loss and training loss are slowly decreasing but around 200 epochs the losses seem to diverge and the model starts to overfit. After 800 the early stopping is triggered and training stops. It can be concluded that the learning rate is too small and the network struggles to converge to an optimal solution. The bottom figure shows training and validation loss for learning rate of 0.01. The model converges quickly and both losses keep decreasing uniformly. After epoch 350 early stopping is triggered. The losses deem this to be the best learning rate as the validation loss is only slightly above the training loss compared to learning of 0.001. The model is then retrained on all training data with selected learning rate and ready to predict the test set. The MAPE for the whole test data is.

```
RMSE = 7503.87656663212231
MAPE = 0.25931504249572754
```

Figure 5.19 - Mixed-input model results

6. RESULTS

model	description	RMSE	MAPE
Linear Regression	Standard parameter	9693	0.587
Decision Tree	Standard parameter	8382	0.285
KNN	Standard parameter	8141	0.263
Random Forest	Max depth 110 – max features 5 – n estimators 100	5043	0.241
MLP	learning rate 0.001 - optimizer ADAM	3994	0.212
CNN	learning rate 0.057 - optimizer ADAM	15070	0.614
MLP + CNN	learning rate 0.01 - optimizer ADAM	7503	0.259

Table 6.1 - Summary of results

7. CONCLUSIONS

Different machine learning and deep learning algorithms were applied and their performance compared on textual features of the scraped second-hand car dataset to achieve a baseline. Firstly, different ML algorithms were explored and Random Forest regressor yielded the best results with an RMSE of 5043 and MAPE of ~24%. Secondly, an MLP was developed and yielded better results than ML algorithms. The RMSE was reduced to 3994 and the MAPE to ~21%. Thirdly, a CNN was trained on the car images and results were compared to the former models based solely on textual features. The CNN performance was poorer based on the selected metrics, achieving an RMSE of 15070 and MAPE ~61%, indicating that training a price estimation model only based on the pictures is not the optimal solution when the same amount of textual features and images are available. The activation layers of the CNN were visualized and it could be noticed how the model was able to learn the right features in high quality - low noise images but was struggling much more on low quality - high noise images indicating that high-quality pictures are fundamental for computer vision tasks. Finally, the main research question of this thesis was answered and a mixed-input model was developed combining the inputs of an ANN for the textual features and CNN for the image data. The mixed-input model did not outperform machine learning models based only on textual and numerical features but yielded better results than a standalone CNN model with an RMSE of 7503 and MAPE = ~25%. The best performing model is therefore an MLP predicting on average 21% away from the real price value. By only looking at MAPE and RMSE the prediction error can still seem a bit high. Also, these evaluation metrics can be misleading as they are averages. While looking at single predictions in figure 5.21, it can be observed that in many cases the predicted prices are very close to the actual prices and that the error for the test cars for which the price is wrongly predicted is very high, skewing the results and the interpretation of the evaluation metrics. These predictions indicate that the model is in many cases doing a good job but is not able to generalize enough. The main reason for this is likely the lack of enough training data.

	actual	prd
0	28000.0	30062.384766
1	22500.0	20140.447266
2	16000.0	16421.447266
3	2900.0	3185.393066
4	25900.0	21692.900391
5	21900.0	25274.501953
6	32500.0	39091.843750
7	23400.0	24041.408203
8	3150.0	6818.373535
9	15000.0	16725.443359
10	21900.0	15868.585938
11	5500.0	5169.888184
12	9500.0	9358.605469
13	22000.0	27085.970703
14	13900.0	13252.068359

Figure 7.1 - Predictions for the test set (only a subsample is showed)

The main objective of this work was achieved. A mixed-input model for second-hand car price estimation was developed and its results compared to more traditional approaches. The findings indicate that mixed-input models are not as effective in price estimation for second-hand cars as standard approaches with a limited amount of training data.

8. FUTURE WORK

Recommendations for future improvement would mainly address the limitations of this work. The focus should be put into collecting more and better quality pictures from P2P trading platforms and explore the impact that a higher number and better quality data may have on model performance. To make image labeling a less cumbersome process, a car pose detection could be developed to enable automatic labeling while scraping. Furthermore, a model for car plate detection could also be developed and applied to detect and automatically blur all car plates and have a more uniform representation of images and facilitate the learning phase for the CNN model. Also, a higher number of textual car features could be scraped and some feature engineering could be implemented. Further parameter optimization for the deep learning models could also lead to better results, different tuning algorithms such as Bayesian optimization and also the implementation of genetic algorithms could be explored. Finally, it would be interesting to explore and apply mixed-input models to other domains.

9. BIBLIOGRAPHY

[1] Harrington P. - Machine learning in action-Manning (2012) page 154

[2] T. Hastie et al., The Elements of Statistical Learning, Second Edition

[3] Aurelien Geron,(2019) Hands-On Machine Learning with Scikit-Learn pages 106,107

[4] Kursa M., Rudnicki W., "Feature Selection with the Boruta Package" Journal of Statistical Software, Vol. 36, Issue 11, Sep 2010

[5] Kuhn, Max_Johnson, Kjell - Applied Predictive Modeling-Springer New York (2018).pdf

[6] Zheng, Alice - Evaluating machine learning models _ a beginner's guide to key concepts and pitfalls-O'Reilly Media (2015)

[7] Seppe Vanden Broucke, Bart Baesens - Practical Web Scraping for Data Science_ Best Practices and Examples with Python-Apress (2018)

[8] Jay M. Patel - Getting Structured Data from the Internet_ Running Web Crawlers_Scrapers on a Big Data Production Scale-Apress (2020)

[9] Cort J. Willmott, Kenji Matsuura(2005) Advantages of the mean absolute error (MAE) over the root mean square error (RMSE) in assessing average model performance. Climate Research, 30, 79–82.

[10]https://www.researchgate.net/publication/272024186_Root_mean_square_error_RMSE_or_mean_absolute_error_MAE-_Arguments_against_avoiding_RMSE_in_the_literature

[11] arXiv:1610.02391v4 [cs.CV] 3 Dec 2019 Grad-CAM: Visual Explanations from Deep Networks via Gradient-based Localization Ramprasaath R. Selvaraju · Michael Cogswell · Abhishek Das · Ramakrishna Vedantam · Devi Parikh · Dhruv Batra

[12] arXiv:1703.10757v3 2 Dec 2019 Diabetic Retinopathy Detection via Deep Convolutional Networks for Discriminative Localization and Visual Explanation - Zhiguang Wang, Jianbo Yang ∗ GE Global Research San Ramon, CA

[13] NIPS-2012- Alex Krizhevsky ImageNet Classification with Deep Convolutional Neural Networks

[14] https://www.tensorflow.org/guide/keras/functional

[15] https://www.tensorflow.org/api_docs/python/tf/keras/Sequential

[16] arXiv:1609.04747v2 [cs.LG] 15 Jun 2017 An overview of gradient descent optimization algorithms

[17] arXiv:1609.08399 Eman Ahmed, Mohamed Moustafa 2016 House Price Estimation from Visual and Textual Features

[18] 2020 Deep MLP-CNN Model Using Mixed-Data to Distinguish between COVID-19 and Non-COVID-19 Patients

[19] Enis Gegic, TEM Journal. Volume 8, Issue 1, Pages 113-118, ISSN 2217-8309, DOI: 10.18421/TEM81-16, February 2019. Car Price Prediction using Machine Learning Techniques

[20] Listiani M. 2009. Support Vector Regression Analysis for Price Prediction in a Car Leasing Application. Master Thesis. Hamburg University of Technology

[21]Peerun, Saamiyah & Chummun, Nushrah & Pudaruth, Sameerchand. (2015). Predicting the Price of Second-hand Cars using Artificial Neural Networks.

[22] Gegic, Isakovic, Keco, Masetic, Kevric (2019 Car Price Prediction using Machine Learning Techniques

[23] Hashemi (2019) Enlarging smaller images before inputting into convolutional neural network: zero-padding vs. interpolation

[24] François Chollet - Deep Learning with Python-Manning (2018)

[25] Charu C. Aggarwal - Data Mining: The Textbook-Springer International Publishing (2015).pdf

[26] Sola, Sevilla - Importance of Input Data Normalization for the Application of Neural Networks to Complex Industrial Problems IEEE TRANSACTIONS ON NUCLEAR SCIENCE, VOL. 44, NO. 3, JUNE 1997

[27] Yuan, Jiang, Li, Huang (2020) Hybrid-DNNs: Hybrid Deep Neural Networks for Mixed Inputs

[28 Unmesh Gundecha, (2014)Learning Selenium Testing Tools with Python

[29] Anish Chapagain, (2019) Hands-On Web Scraping with Python

[30] arXiv:1610.02391v4 [cs.CV] 3 Dec 2019 Grad-CAM: Visual Explanations from Deep Networks via Gradient-based Localization Ramprasaath R. Selvaraju · Michael Cogswell · Abhishek Das · Ramakrishna Vedantam · Devi Parikh · Dhruv Batra

[31] arXiv:1703.10757v3 2 Dec 2019 Diabetic Retinopathy Detection via Deep Convolutional Networks for Discriminative Localization and Visual Explanation - Zhiguang Wang, Jianbo Yang ∗ GE Global Research San Ramon, CA