Berner Fachhochschule
Bern University of Applied Sciences
Haute école spécialisée bernoise

Bachelor Thesis

BTI7321

Logos Recognition for Webshop Services

Author: Noli Manzoni

Supervisor: Prof. Dr. Olivier Biberstein

Expert: Jean-Marie Leclerc

V 0.1

14.06.2018

Contents

Declaration of the graduands

Selbständige Arbeit / Travail autonome

Ich bestätige mit meiner Unterschrift, dass ich meine vorliegende Bachelor-Thesis selbständig durchgeführt habe. Alle Informationsquellen (Fachliteratur, Besprechungen mit Fachleuten, usw.) und anderen Hilfsmittel, die wesentlich zu meiner Arbeit beigetragen haben, sind in meinem Arbeitsbericht im Anhang vollständig aufgeführt. Sämtliche Inhalte, die nicht von mir stammen, sind mit dem genauen Hinweis auf ihre Quelle gekennzeichnet.

Par ma signature, je confirme avoir effectué ma présente thèse de bachelor de manière autonome. Toutes les sources d'information (littérature spécialisée, discussions avec spécialistes etc.) et autres ressources qui m'ont fortement aidée dans mon travail sont intégralement mentionnées dans l'annexe de ma thèse. Tous les contenus non rédigés par mes soins sont dûment référencés avec indication précise de leur provenance.

Last Name, Name

Date

Signature

List of Figures

List of Tables

Listings

Abstract

Nowadays, just a click and few days and almost any goods can be received at home. Unfortunately, this convenience comes with many risks for the customer who could buy an object without knowing that it is counterfeit. A possible solution could be to navigate the web in search of fraudulent websites to report. Pertinent information to increase the probability of detecting this type of webshop is the presence of logos belonging to delivery and payment service companies. This logos' recognition task is not trivial because these logos could be part of larger images. For this reason, this thesis aims to develop an application which detects whether an image contains certain types of logos.

1. Introduction

1.1 Motivation and purposes of the thesis

Due to the growth of technology and the reduction of hardware prices, a greater number of people have started to use the internet. This new technology not only has brought benefits to our lives but it has also brought many dangers. If 20 years ago goods were bought in the store because they were very difficult to order, today it is no longer the case. In fact, just a click and few days to receive any product directly at home. However, this convenience comes with many risks for the consumer. As an example, can be cited a customer who buys an object on the internet without knowing that it is counterfeit. This is not only a waste of money for the consumer but above all, if he is aware of this it would cause a huge loss to the manufacturer. For this reason, a possible solution could be to navigate the web in search of websites that sell counterfeit products. An extra information which can increase the probability of detecting such websites is the presence of logos belonging to delivery and payment service companies. Of course, such a system is not realizable in this project. In fact, this Bachelor thesis aims to develop a single part of the system i.e. a program that evaluates if a given website is a webshop by means of logos recognition.

This thesis is a follow-up of the project *Logos Recognition* [16] which had the objective of comparing different approaches for logos recognition. The techniques studied in that project were the most common ones that are described in most of the computer perception literature. Although the capability of each technology was already known, it has been decided to revisit them to get a more precise idea of how they behave with specific images such as logos belonging to payment and delivery service companies. The project came to an end and it has confirmed that the best technique are the Scale-invariant feature transform (SIFT) descriptors. For this reason, the image recognition process of this project is based on them. Unfortunately, the number of SIFT descriptors vary from image to image and therefore, they cannot be used directly in a machine learning classifier. Consequently, one of the objectives of this project is to find out how these descriptors could be used to create a fixed size features vector. As stated in the project *Logos Recog-*

nition [16], the SIFT descriptors should work perfectly with a single logo, but they might not work with figures containing multiple of them. Therefore, the second aim of this project should be to discover a technique that allows detecting and recognizing multiple logos contained in a single image. Once the most suitable technologies have been identified and tested, they should be incorporated into an application that has the goal of discovering if a website is a webshop or not.

In the last years a new technology called Artificial Neural Networks (ANNs) have become more and more popular and because of its extraordinary results, all the classic techniques have been almost abandoned. This thesis has a hidden objective because in addition to develop an application to recognizes payment's and delivery's logos, it should assess if the classic technologies are still useful or if they are completely outperformed by the ANNs. For this reason, this project executes a multitude of tests to discover the best logos' recognition technique so that it can be compared with the ANNs. However, all the prototypes developed to assess the different technologies are implemented without any quality restriction. In fact, they are created to be functional and compact without considering time performances. This choice is due to the limited amount of time available for this project.

This document is realised with latex[17], MiKTeX [18] and TeXstudio [19].

1.2 Structure

First of all, Chapter 2 presents all the information needed to approach safely the object recognition field. To be more precise, this chapter contains an introduction to the concept of object recognition, a short history of the domain and information concerning research related to this topic. Chapter 3 explains the different choices which have been made at the beginning of this project. These choices are really important because they are similar to a preliminary study, in fact, they are necessary to understand what can be done or not throughout this thesis. Chapter 4 is similar to the previous two because it introduces the reader to the concept of a classifier. This chapter gives definitions with the help of a simple example. More importantly, it explains why the classifier is useful for this thesis and how it is used. In Chapter 5 begins the description of the real project. In fact, this chapter contains the explanation of the Scale-invariant feature transform (SIFT) descriptors and the new model in which they are incorporated. This description is necessary because the these two technologies are the fundamentals for the whole logos' recognition process. Then, Chapter 6 explains how the previously analysed technologies are used to recognize a single logo. This chapter describes different classifiers, and in the end, it asses which one is the best for this project. In addition, this chapter contains also an introduction to another tool called WEKA. At the end of this chapter, there

is a comparison between the technologies discovered and one of the newest idea in computer vision i.e. ANNs. Because this project needs to deal with the recognition of multiple logos in a single image, Chapter 7 illustrates this problem and the solutions. In this part, in addition to these explanations, there are the final results which assess the different implementations/prototypes. Chapter 8 illustrates the development of the final application which includes all the explained algorithms. Finally, Chapter 9 gives a very high level overview of the topic ANNs which have been briefly discussed in Section 6.4

The integral code and other useful files to understand the explanations discussed in this document can be found in the GIT repository at `https://gitlab.ti.bfh.ch/nolim2/Logos-Recognition-for-Webshop-Services`. To navigate or to find specific files, please refer to the *README* on the main page.

Throughout the document, there is crossed text (i.e. ~~lorem Ipsum is simply dummy text of the printing and typesetting industry~~) which represents parts of the project which have been changed. The main adjustment was the modification of the optional requirements i.e. other sources of information to Artificial Neural Networks (see project management document). These outdated pieces of information are still there to show the reader the development of the project.

2. Object recognition preface

Object recognition is a vast and complicated field of computer vision and its objective is the identification of objects in an image or a video (a.k.a. a sequence of images). The focus of this project is the identification of a special type of object called *logo*. For this reason, an introduction to recognition technologies, their history, and related research is an important prerequisite to understand the concepts explained in the following chapters. In fact, this chapter tries to give a high-level overview of the development of these technologies, and they current situation. Moreover, an introduction to previous research and their results are presented here. To be more precise, this chapter contains a brief explanation of the content and results of the project that allowed this Bachelor thesis to born.

Object recognition, a.k.a. object classification is a sub-domain of computer vision which tries to determine whether an object is contained in an image. According to [20], object recognition is the area of Artificial Intelligence (AI) concerned with the abilities of AI implementations to recognize various things and entities. Moreover, the author thinks that this domain is at the convergence points of robotics, machine vision, ANNs, and AI. In addition, [21] says that object recognition algorithms rely on matching, learning or pattern recognition systems using appearance-based or feature-based techniques.

In the last decades, many algorithms and approaches to this field have been proposed. For this reason, and to have a better understanding of the filed, a short descriptor of the major inventions is essential.

2.1 A very short history

According to [22] modern computer vision had its origin in the early 1960s. One of the first application was a pattern recognition system for character recognition. A second application in 1964, tried to automate the wire-bounding process of the transistors to replace human workers. Unfortunately, the system achieved an accuracy of 95% which was considered not enough to replace a worker. Approximately 10 years later in the 1973s, a

fully automated assembly machine was created. This evolution in the semiconductor industry enabled the mass production of complex semiconductors such as Wafers. This is a really important step because the complexity of the task did not allow humans to perform the operation efficiently. Then, recognition systems started to be applied in biomedical research for the chromosome recognition task, in the food industry for the classification of tablets and capsules, in medical imaging for robust segmentation of anatomical structures, in handwritten character recognition for mail sorting and many other fields. Unfortunately, even if recognition systems were applied for many tasks, nobody found a robust solution to a more general problem of recognizing complex object classes until the early 2000s, when machine learning approaches i.e. Bag of Words (BoW) with Scale-invariant feature transform (SIFT) [3], became popular. In the following years, similar techniques, such as Histogram of Oriented Gradients (HOG) [23], Viola-Jones object detection framework [24], etc. were developed without revolutionizing the field. Contrary, the decade after brought a big revolution, because in the 2012 AlexNet [25], a Convolution Neural Network (CNN), won a big competition called ImageNet Large Scale Visual Recognition Competition (ILSVRC).

2.2 Today's situation

Ever since the victory of AlexNet [25] at ILSVRC back in 2012, Convolution Neural Networks became the new standard for object recognition, opening a new era for computer vision. Unfortunately, the ANNs and CNNs are not a mandatory part of this project, therefore there is not a deep explanation of the subject (for more information please refer to Chapter 9). Anyway, according to [26] AlexNet won the ImageNet Large Scale Visual Recognition Competition with a large margin. In fact, although it was based on the decades-old LeNet, it achieved an extraordinary result as shown in Figure 2.1 by the red rectangle. In the next two years, this technology was ameliorated and in fact, the winners of the ImageNet Large Scale Visual Recognition Competition had each time better results (see Figure 2.1). Then, according to [27], in 2014 the first application of CNNs to object detection had been developed and it took the name of Regional CNN (R-CNN). In that year, You Only Look Once (YOLO), an application which is still state-of-the-art today, was developed. In 2015, the object recognition field made another big step forward. In fact, that year the ResNet [28] won the ImageNet Large Scale Visual Recognition Competition with an unbelievable error rate of 3.6% . This value is so incredible because human performance is around an error rate of 5 to 10 %. Therefore, that year, human capabilities to recognize objects were outperformed by a machine. In the following years until today some improvements were made but without new breakthroughs.

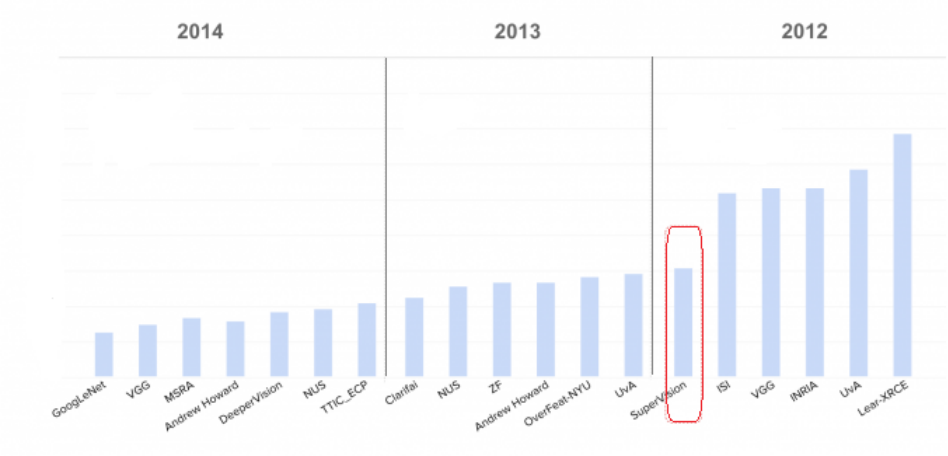


Figure 2.1: ImageNet classification error in 2012/13 and 2014 (source [1])

2.3 Related researches

This Bachelor thesis is the follow-up of the project *Logos Recognition* [16], which was a feasibility study with the objective of evaluating the topic *logo recognition*. Since the arrival of ANNs and CNNs, the classical object recognition approaches have been abandoned. For this reason, and because I was keen to understand the concepts behind the different techniques that I learned at school, the objective of the viability study was to understand if classical approaches had the capability to be still used today. For this reason, in that study, different techniques such as perceptual hash, feature points, etc. were tested. All the different implementations realised to understand if there was a suitable technique to recognize logos, were tested without any machine learning technique. To be more precise, all the implementations were prototypes with the only aim of extracting information from an image and compare the extracted vector with a threshold (unsupervised learning). These implementations were not intelligent algorithms but rather the simplest possible implementations. This simplicity allowed to test a larger number of techniques and to evaluate which of them was the best. At the end of this viability study, only a technique stood out with its good results and its name is Scale-invariant feature transform (SIFT). This technology, even if it was used in a very simple classifier, it achieved a sufficient result which was the highest among all. Because a technique with a theoretically good result had been discovered, the feasibility of this Bachelor thesis was confirmed. For this reason, this Bachelor focuses on the implementation of a smarter SIFT system and a smarter classification technique. These improvements should allow increasing considerably the result so that a real application could be developed.

3. Preliminary research

This chapter introduces and explains all the initial decisions which have been taken to start the project. This chapter is called *introductory research* because it should help to verify the feasibility of this Bachelor thesis. To be more precise, this chapter lists different technologies and implementations for all the parts of this project. This step should help understand if all the components of this project can be implemented and how much effort is required for each of them.

As first step, the different possibilities to recognize logos as single image are listed. Then, they are evaluated at a high level to decide which one will become the centre of the project. As second step, to better plan the different tasks and to understand if there are existing technologies to identify logos in larger images, a small research about object detection is done. ~~Once these aspects are covered, different technologies to extract other sources of information such as keywords, hyperlinks, and JavaScript code are shortly explained~~ [**update 03.04.2018**]. Then to finish, a comparison between the different programming languages available for this project is made. This step should help understand which language is the most suitable for the realisation of the tasks listed above.

3.1 Logos as single images

As it has been shown in Section 2.3, this Bachelor thesis is a follow-up of the project *Logos Recognition* [16]. The results obtained in that previous work showed that the Scale-invariant feature transform descriptors are one of the best technique to identify logos. For this reason, this project should use this technology but with some improvements. These enhancements are necessary because the results achieved in the project *Logos Recognition* [16] were sufficient but not enough for a real application. Consequently, this thesis must be based on a more complex and advanced algorithm to take full advantage of the SIFT descriptors.

A small research about image classification with SIFT descriptors allowed the discovery of a model called Bag of Words. This model uses the SIFT

algorithm to generate a more complex and precise image feature which can be used in a more advanced classification system such as Support Vector Machine. Therefore, this system is perfect for this project because it allows using the SIFT descriptors in a smarter way to classify images, a.k.a. logos, with a more intelligent classifier.

3.2 Not only single logos

Single logos can be recognized with simple techniques as proved in the project *Logos Recognition* [16] and with more complex ones as seen in Section 3.1. The task of recognizing logos contained in larger images seems to become more complex. In the reading of the paper *Deep Learning for Logos Recognition* [29], where the authors explain how they managed to detect multiples object in an image, it has been discovered a technique called Selective Search. A careful research allowed to learn that this technique is one of the many available for object detection task. The simplest algorithm to identify single objects/logos in a larger image, according to [30], is the sliding window. This technique simply slides a box or a window over an image to select a patch. Then, it classifies each image patch covered by the windows with a recognition technique. [30] In addition to this system, there are more complicated algorithms that fall under the category *region proposal*. These techniques are slightly smarter because this time they generate different bounding boxes that represent the objects included in an image. According to [30] there are five different methods for region proposal and they are:

- Objectness [31]
- Constrained Parametric Min-Cuts for Automatic Object Segmentation [32]
- Category Independent Object Proposals [33]
- Randomized Prim [34]
- Selective Search [9]

The technique most used amongst the ones presented here is the Selective Search. Therefore, to solve the problem of identifying logos contained in a larger image a prototype based on the original Selective Search's paper [9] will be developed.

3.3 Other sources of information

[Update 03.04.2018] Because of the good results obtained in Section 6.4 and the increase in complexity of the technologies used to classify logos in

larger images, the optional requirements changed. In fact, other sources of information are no more a requirement. Instead, once all the mandatory requirements are completed, the ANNs should be studied and used to increase the accuracy of the system. The research of this technology must stay at a high level and its only purpose should be to assess if ANNs achieve better results than the classic methods.

This project must deal with information such as images ~~, text, hyperlinks, JavaScript codes~~ contained in a website **[Update 03.04.2018]**. The analysis of other sources of information is an optional requirement but the images' analysis is not. For this reason, and to understand the complexity of the task, techniques to extract figures from a website must be discovered.

A deep research shown that this field of study is called *web scraping*. It is defined as the practice of using a computer program to surf through a web page and gather the data that is needed in format most useful while at the same time preserving the structure of the data [35]. To achieve the mandatory requirements (download all images from a website) there are many libraries and techniques that are specialized in finding only images on a given website. ~~For the optional requirements (keywords, hyperlinks and JavaScript code) the task becomes slightly more complicated. The keywords' extraction is an easy task because, as in the case of images extraction, there are many libraries to solve the problem. Contrary, the analysis of hyperlinks in a website is always used to compute the relevance of a website and not to find specific information [36]. Furthermore, the analysis of JavaScript code is not used to extract information about payment's and delivery's services but to detect incoherences in a website development. For these two reasons, to analyse these two components, the typical characteristic in terms of hyperlinks and JavaScript code of a web shop must be understood. Once this step is completed, different techniques of extraction and analysis can be explored. Since this is an optional requirement, it will be implemented only once all the compulsory functionalities will be completed.~~**[Update 03.04.2018]**

3.4 The developer's choice

This section lists the three programming languages available for this Bachelor thesis which are Scala, Java and Python. To choose which of these languages is the most suitable for this project, two comparisons are made. These two confrontations are useful to understand the capabilities of each programming language and their efficiency in logos recognition by means of available libraries.

3.4.1 Java vs Python

On the internet, more precisely in the developers' community, it is often said *with Python a developer can be more productive than with Java.* Until now, in my short experience as a developer, I have almost always programmed with Java but, from my experience in developing with Python for the project *Logos Recognition* [16] here at the Bern University of Applied Science, I can agree with the above statement. In fact, other than the time needed to understand the principles of the language, my experience with Python was really straightforward. Putting this fact aside, why the community thinks Python is more productive than Java? For a lot of people, as explained in [37], Python is more productive because it is a dynamically typed language in contrast to the well know static typing Java. This is not the only argument that proves Python is more productive in fact, [37] says that Python is more concise (a.k.a. tense) and compact than the verbose Java. Although these statements are against Java, with my experience I can say that Java has always a place as a programming language. Indeed, as it is shown in the statistics from [38], Java is still the most used.

These two languages have almost the same number of libraries for image processing. In [39] for Java and [40] for Python, are listed different libraries to manipulate images. Java disposes of a good library called ImageJ2 [41] and its distribution Fiji [42] with which I have already worked. Python, on the other hand, disposes of more libraries such us Pillow [43], scikit-image [44], scipy [45], etc. Most of these Python's libraries have been used for the realisation of the project *Logos Recognition*[16]. For this reason, I can say that they are complete and really easy to use. Additionally, both languages have a wrapper for the C++ OpenCV [46] implementation. It is a really good library for computer vision that contains a lot of functionalities for images processing.

3.4.2 Java vs Scala

As it has been already said in Section 3.4.1, Java is the leader of the programming languages because, according to [47], it had at its launch five killer features that were:

- Runs on any platform
- Backward compatibility
- Garbage collector
- Classical object-oriented paradigm
- Syntax similar to C++

In addition, Java was able to stay on top until today because of the continuous introduction of important and useful features. If Java is so good why the emergence of Scala? Scala, differently from Java, is a hybrid programming language which unites two worlds i.e. Object-Oriented Programming (OOP) and functional programming (FP).[47] Moreover, Scala is compatible with Java, and therefore it can use all the Java's libraries. In fact, Scala is like an upgraded version of Java. It has all Java advantages with additional features like more concise syntax, implicit arguments and values, pattern matching, Read-Eval-Print-Loop (REPL), etc. (refer to [47] for more information).

Because of Scala interoperability, a.k.a. its possibility to generate Java code and to use all the Java's libraries, Scala can use all the libraries cited in Section 3.4.1 and more like [48]. For these reasons, Scala could be a good candidate to develop an application. Furthermore, since Scala is unknown to me, this project could be an excellent opportunity to learn a new programming language and to explore the depths of the functional programming paradigm.

3.4.3 Verdict

As it has been shown in the two sections above (3.4.1 and 3.4.2), the three cited programming languages have some differences and therefore they have specific applications. On one hand, as observed in Section 3.4.1, Python helps to be more productive and therefore it is more suitable to build prototypes. On the other hand, Java and Scala seem to be more used for big and commercial applications. According to these considerations, Python is more suitable to test out the different situations discussed in this Bachelor thesis. Contrary, Scala as an *improved* version of Java, is more suitable to developing the final application. Therefore, Python is used for the realisation of the different prototypes (classifier, logo recognition, logos in larger images, extracting information from other sources, etc.). ~~In the other hand, Scala is used for the combination of the different parts and to deliver a stable multi-platform application~~ [**update 17.04.2018**] Today, an implementation with sufficient results has not been found yet. For this reason, new ideas i.e. Artificial Neural Networks must be approached. Unfortunately, Java and therefore Scala do not have a lot of tools that work with ANNs. Therefore, the development of the final application with these languages is not realisable. In addition to this motivation, Scala was chosen only to give me the possibility to learn a new programming language. Therefore, its exclusion does not affect the project realisation. For these reasons, the final application is developed with Python. This change allows to not waste the work accomplished until today. Moreover, the saved time can be used to write a better and more tested code.

4. Classifier

This chapter explains the working principles of a classifier and how its results can be evaluated. Before this explanation, at the beginning of the chapter, all the preconditions are explained so that the discussed concepts can be better understood.

4.1 A straightforward definition

To understand the different techniques for image classification and their working principles, the concept *classifier* must be first understood. A classifier is a program which assigns a class to a new observation based on data extracted from a training set. According to [49, p. 9], classification is a supervised learning problem where there is an input X, an output Y and the task is to learn the mapping from input to output. There are a larger number of classification algorithms, and they are divided in different categories such as linear classifiers, Support Vector Machines (SVMs), kernel estimation, decision trees, etc. Because of the great number of techniques, there are many application domains such as computer vision, speech recognition, biological classification, etc. This definition gives a clearer idea of what is a classifier, but why it is so important? A classifier or classification technique is the last step of the image analysis process. [50, p. 5] It receives, from the segmentation representation step, multiple features that should be classified.[50, p. 5] By assigning an object class to these features during the training process, the class of a new feature can be identified by applying a classifier function to it.[50, p. 5][50, p. 40]

4.2 The confusion matrix

As it has been explained in the chapter's introduction, the objective of this part of the document is to explain the working principles of a classifier. This is an important part because all the implementations which will be explained in the following chapters are based on a classifier. To decide which of the implementations is the best, a technique to evaluate the results of a classifier is needed. In fact, the purpose of this section is the explanation of all the

concepts needed to evaluate a classifier. The first thing to do to assess a classification technique is to think about its quality by answering two questions *What is a good classifier?* and *How can its quality be measured?*.[50, p. 41] After some thoughts, it can be noticed that a possible class prediction cannot be only *true* or *false*. In fact, the class prediction can fall into four different categories. Here, comes into play the confusion matrix which allows understanding whether a classification system is good or not. The advantage of an evaluation with this confusion matrix is that it does not consider factors such as usability, efficiency, speed, portability, accessibility, etc. This instrument is perfect for this case because the classifiers developed in this thesis, are prototypes created with the only purpose of testing different technologies of classification without thinking about the various factors mentioned above. To be more precise, a confusion matrix is a table used to describe the performances of a classifier. The classification technique needs to be tested on a dataset where each component, and therefore each class, is known in advance. The concepts and values of the confusion matrix are simple to understand but unfortunately, they are often confused and mixed together. For this reason, each value must be studied in depth, so that they cannot be mistaken.

The information shown in this section is partially based on the project *Logos Recognition* [16].

4.2.1 Simple example

In order to understand the different values of a confusion matrix, let's take as an example a very simple classifier that must distinguish the difference between MasterCard's logos and all the other types of logos. Table 4.1 shows how a confusion matrix can efficiently summarize the results of such a program. Before the different values can be explained, the differences between actual class and predicted class must be understood. The actual class, as the name says, is the actual a.k.a. correct class of the processed logo. Contrary, the predicted class is the class assigned by the classifier and therefore, it could be correct or not. When the actual class and the predicted class are the same, the classifier has made a good prediction. In the opposite case, if the two classes are different, the classifier has made an error.

		Actual class	
		MasterCard	**Other**
Predicted class	**MasterCard**	8	5
	Other	3	4

Table 4.1: Confusion Matrix

The table, a.k.a. confusion matrix 4.1, contains a lot of useful information. For example, it shows that the classifier has 20 logos to classify, where $8 + 3 = 11$ are MasterCard and $5 + 4 = 9$ are of another type. This information shows how the classifier, out of 11 MasterCard logos, has wrongly recognized 3 logos as another type. Furthermore, this confusion matrix shows that out of 9 other type of logos, the classifier has wrongly recognized 5 of them as MasterCard. This information allows understanding that this system struggles to recognize non-MasterCard logos (5 errors on 9 logos $\approx$ 55% error rate) but that it recognizes better MasterCard logos (3 errors on 11 logos $\approx$ 30% error rate). This table is really useful because all the correct predictions are visible in the diagonal. Moreover, errors are also easy to identify because they are represented by the other values.

4.2.2 Official definition

After this overview of the structure of a confusion matrix, the official definitions and correct terminologies can be explained.

In predictive analytics, the confusion matrix, error matrix or table of confusion is a table composed of two columns and two rows. This table shows false positives (FPs), false negatives (FNs), true positives (TPs) and true negatives (TNs). The values of the example of Section 4.2.1 are illustrated in Table 4.2. These values are very important because, in addition to give an optimal overview over the results of the classifier, they can be used to calculate other quality measures such as precision, specificity, sensitivity, etc. These quality measures are really useful, because, with a single value i.e. precision, the quality of a classifier can be easily assessed. Before being able to analyse these more specific values, the four basic values a.k.a. FPs, FNs, TPs and TNs, must be explained.

		Actual class	
		MasterCard	**Non-MasterCard**
Predicted class	**MasterCard**	8 True Positives	5 False Positives
	Non-MasterCard	3 False Negatives	4 True Negatives

Table 4.2: Confusion Matrix in Predictive Analytic

True Positives

True positives are logos that are correctly identified to belong to the analysed class. This means that the predicted class is the same as the actual class.

In the example of Section 4.2.1, the true positive value is the number of MasterCard logos that are recognized as MasterCard logos, therefore 8.

False Positives

False positives are logos that are wrongly identified to belong to the analysed class. This means that actual class is not the same as the predicted class. In the example of Section 4.2.1, the false positive value is the number of non-MasterCard logos that are recognized as MasterCard logos, therefore 5.

True Negatives

True negatives are logos that are correctly identified to not belong to the analysed class. This means that the actual class is the same as the predicted class like the case of true positives. In the example of Section 4.2.1, the true negative value is the number of non-MasterCard logos that are recognized as non-MasterCard logos, therefore 4.

False Negatives

False negatives are logos that are wrongly identified to not belong to the analysed class. This means that the actual class is not the same as the predicted class like the case of false positives. In the example of Section 4.2.1, the false negative value is the number of MasterCard logos that are recognized as non-MasterCard logos, therefore 3.

4.3 A faster evaluation

After this detailed explanation of the four fundamental values, the generation of the quality measures cited in Section 4.2.2 can be explained. The objective of this part is to show why these quality values are so important and which of them is the more essential for this Bachelor thesis.

The analysis of the data in Table 4.2 allows extracting the four basic values (false positives, false negatives, true positives and true negatives). These parameters can be used to evaluate the effectiveness of a classifier. Unfortunately, this process of reasoning often leads to confusion, and therefore to an incorrect evaluation. For this reason, it is better to use more sophisticated values called quality measures [50, p. 41]. There are five quality measures, and they can be obtained by the four basic values. They are very important because they allow understanding very quickly the efficiency and/or usefulness of a classifier. For this reason, in most or even all evaluations of this thesis, one or more of these quality measures are used. Before the research of the most important quality measure for this project, let's see how they are computed and for what they are used.

Sensitivity

Sensitivity, a.k.a. True Positive Rate (TPR), is the proportion of logos correctly identified which belong to the analysed class over all the logos of the analysed class. In the example of Section 4.2.2, sensitivity is the proportion of MasterCard logos that are correctly identified over the total number of MasterCard logos, therefore $^{8}/_{(8+3)} = {}^{8}/_{11} = 0.727 = 72.7\%$.

$$TPR = \frac{number\ of\ true\ positives}{number\ of\ true\ positives\ +\ number\ of\ false\ negatives}$$

Specificity

Specificity, a.k.a. True Negative Rate (TNR), is the proportion of logos correctly identified which belong to the non-analysed class over all the logos of the non-analysed class. In the example of Section 4.2.2, specificity is the proportion of non-MasterCard logos that are correctly identified over the total number of non-MasterCard logos, therefore $^{4}/_{(4+5)} = {}^{4}/_{9} = 0.444 = 44.4\%$.

$$TNR = \frac{number\ of\ true\ negatives}{number\ of\ true\ negatives\ +\ number\ of\ false\ positives}$$

Precision

Precision, a.k.a. Positive Predictive Value (PPV), is the proportion of logos correctly identified which belong to the analysed class over the total number of logos identified to belong to the analysed class. In other words, the precision can be seen as repeatability, or reproducibility of the measurement. In the example of Section 4.2.2, the precision is the number of MasterCard logos that are correctly identified over the total number of logos that are identified as MasterCard, therefore $^{8}/_{(8+5)} = {}^{8}/_{13} = 0.615 = 61.5\%$.

$$PPV = \frac{number\ of\ true\ positives}{number\ of\ true\ positives\ +\ number\ of\ false\ positives}$$

Negative Predictive Value

Negative Predictive Value (NPV) is the proportion of logos correctly identified which belong to the non-analysed class over the total number of logos identified to belong to the non-analysed class. NPV is exactly the opposite of the PPV, in the sense that the NPV is the same as the PPV if it is computed for the other class. In the example of Section 4.2.2, the precision is the number of non-MasterCard logos that are correctly identified over the total number of logos that are identified as non-MasterCard, therefore $^{4}/_{(4+3)} = {}^{4}/_{7} = 0.571 = 57.1\%$.

$$NPV = \frac{number\ of\ true\ negatives}{number\ of\ true\ negatives\ +\ number\ of\ false\ negatives}$$

Accuracy

Accuracy is the proportion of correct predictions over the number of all predictions. In other words, the accuracy can be seen as the proximity of measurements results to the true value. In the example of Section 4.2.2, the accuracy is the number of all logos that are correctly identified over the total number of logos, therefore $^{(8+4)}/_{(8+4+5+3)} = {}^{12}/_{20} = 0.600 = 60.0\%$.

$$ACC = \frac{number\ of\ TP\ +\ number\ of\ TN}{number\ of\ TP\ +\ number\ of\ TN\ +\ number\ of\ FP\ +\ number\ of\ FN}$$

4.3.1 The essential value

As it has been explained in the introduction, one of the purposes of this chapter is to explain how a classification technology can be evaluated. This step is very important because this project uses a lot of classification techniques and therefore, a valid method to compare them must be used. Moreover, a comparison or evaluation often needs a decisive factor that can be used as a key value. For these reasons, the most essential quality value for the classifiers of this Bachelor thesis must be found.

Section 1.1 presented a possible field of application for logos' classifiers. In fact, a possible domain could be a support in the identification of sellers of counterfeit objects on the web. In this specific case, all the sellers should be identified without errors. Unfortunately, developing a perfect classifier is practically impossible because the identification does not always work perfectly. For this reason, a classifier should not strive for perfection but rather attempt to minimize a specific error. The worst error that could happen in a similar application, is the identification of a logo when it is not there e.g. identify a MasterCard logo when in reality it is another figure. This error is to avoid because it would lead to the accusation of a website when it is not in the wrong. In fact, for this project, a classifier must have rather less true positives but zero false positives.

Now that the classifier context and goals have been set, the last step to do is the identification of a primary quality value for the evaluation and/or comparison for the classifiers proposed in this document.

Sensitivity gives information about the number of logos correctly identified which belong to the analysed class (TPs), over all the logos of the analysed class (TPs + TPs). The specificity, similarly, gives information on the quantity of logos correctly identified which do not belong to the analysed class (TNs), over the total number of logos that do not belong to the analysed class (TNs + FPs). In these two cases, these values do not give a lot of information about the error that must be avoided. The same thing happens for the accuracy and Negative Predictive Value because even in these cases little information about the number of false positives is given. This is because accuracy is a parameter that takes into account all the values

contained in the confusion matrix, while the NPV is the exact opposite of the searched value.

The last remaining parameter to explain is the precision. In fact, it is exactly the searched value. Precision gives a ratio obtained with true positives and false positives. This value is really important because it shows instantly if a classifier has false positives. For this reason, precision is the most important value for this project and therefore, the system must try to achieve the highest value. However, this is not the only quality parameter to consider in fact, the evaluation process of the different logos' classifiers, uses all the quality measures with a special focus on the precision.

5. Logos recognition

The realisation of the project *Logos Recognition* [16] helped to understand the differences between most of the classical technologies for features extraction. In addition to this, the research allowed to evaluate the different technologies and to decide which one is the most suitable for this Bachelor thesis. The aim of that project has been reached because a winner has been found. In fact, the choice has fallen upon the Scale-invariant feature transform, a.k.a. SIFT descriptors. For this reason, this project should use this technology to classify logos. At the beginning of this chapter, the SIFT descriptors are explained so the reader does not need to read through the entire document of project *Logos Recognition* [16]. After the introduction, the use of these features in a more intelligent system called Bag of Words (BoW) is explained. At the end of this chapter, to assess if the SIFT descriptors and the BoW model are the most suitable for this specific classification task, some tests are run and analysed.

5.1 Scale-invariant feature transform

In 2004 D. Lowe published a paper [3] where he explains the Scale-invariant feature transform (SIFT). This section tries to recap, in a simpler way, the main concepts and steps of the paper. For this reason, users that seek more information should refer to it or to the project *Logos Recognition* [16]. The most important advantage of this technology is its invariance to scaling, rotation, change in illumination and 3D camera viewpoint. Moreover, this algorithm generates a larger number of descriptors (in a 500x500 pixel image, it gives about 2000 descriptors). This larger number of descriptors is highly distinctive and therefore, it allows a single point to be correctly matched in a large database of descriptors.

There are four major steps to generate the SIFT descriptors. Therefore, this section tries to summarize them with a high-level overview of the processes.

5.1.1 Scale-space extrema detection

This algorithm's step should find key-points in different scales. Therefore, to achieve a similar result a scale-space filtering must be used. To detect stable

key-point locations in a scale-space, the author proposed the use of scale-space extrema. These extrema are present in the Difference-of-Gaussian (DoG) function convolved with the image computed from the difference of two nearby scales which are separated by a constant multiplicative factor k (see Figure 5.1 left side). Once all the DoGs are computed, these extrema can be efficiently detected over all the scales. To be more precise, in this process each pixel in an image is compared with its eight neighbours, the nine pixels in the next scale and the nine pixels in the previous scale (see Figure 5.1 right side).

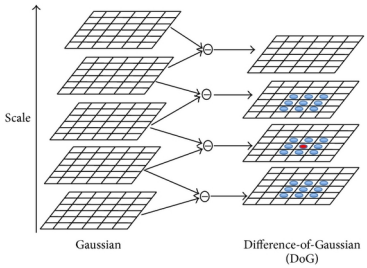
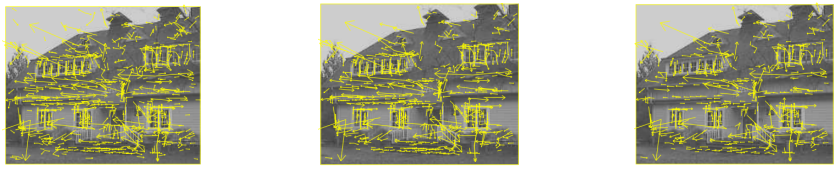


Figure 5.1: Difference-of-Gaussian (source [2])

5.1.2 Accurate key-points localization

The previous step generated a lot of key-points candidates which must be refined to get a more accurate result (see Figure 5.2a). To be more precise, candidates filtering refers to the removal of key-points with low contrast or which are poorly localized along an edge. First, to delete candidates with low contrast, the location of each extreme is computed. Then, the key-point is discarded if its location value is less than 0.03. This step allows to slightly decrease the number of key-points in an image as shown in Figure 5.2b. Secondly, to remove poorly located key-points, the principal curvature, which gives an idea of how good a key-point is located, is used. To compute it, the Hessian matrix, its trace and its determinant are calculated at the location and scale of each key-point. Then, these three values are used to compute a ratio that if it is under a threshold, eliminates the key-point. This step further decreases the number of key-points to an optimal number (see Figure 5.2c).



(a) Initial 832 key-points (b) Mid 729 key-points (c) Final 536 key-points

Figure 5.2: Key-points on a 233x189 pixels image (source [3])

5.1.3 Orientation assignment

The previous step redefined the key-points to an optimal number. These survived key-points are good but unfortunately, they are not rotation invariant. To achieve this invariance, a consistent orientation to each key-point should be assigned so that, the descriptor can be represented relative to this orientation. First, an orientation histogram is formed from the gradient orientations of sample points within a region around the key-point. The histogram has 36 bins of 10 degrees each so that it can cover a 360-degree range (see Figure 5.3).

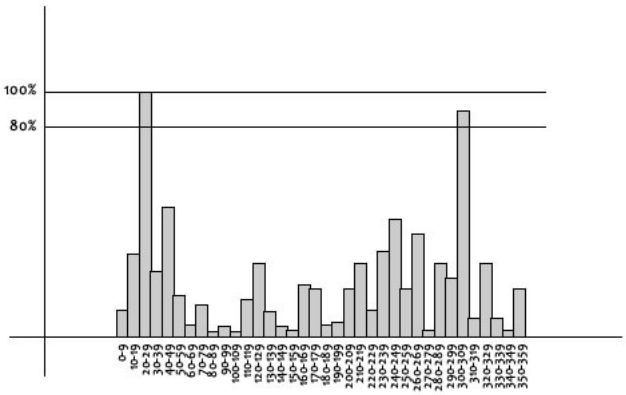
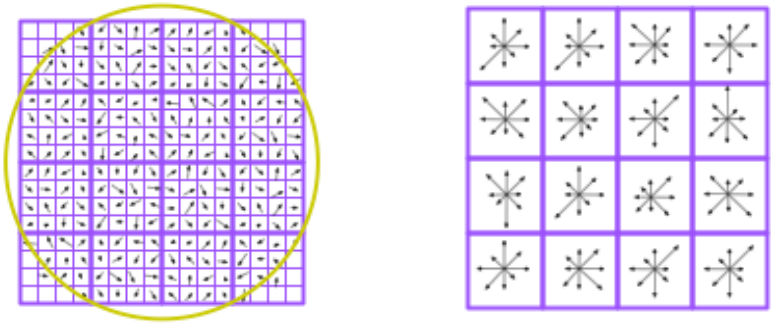


Figure 5.3: 36 bins histogram (source [4])

Secondly, each sample is added to the histogram weighted by its gradient magnitude and distance from the key-point. The highest peak in the histogram is used as direction for the key-point. In the case that there are two or more peaks above 80%, a new key-point is created with the same location and scale but with a different direction.

5.1.4 The local image descriptor

The previous steps assigned an image location, scale, and orientation to each key-point. Now, this information can be used to create a highly distinctive descriptor. Moreover, this new descriptor must be invariant to the remaining variations i.e. change in illumination and 3D viewpoint. To create this invariant descriptor, the algorithm computes the gradient magnitudes and orientations at each image sample point in a 16x16 pixels region around the key-point (see Figure 5.4a). Then, these values are weighted by the distance from the centre. Furthermore, they are summarized in 4x4 sub areas in eight different angles, one every 45 degrees (see Figure 5.4b). At the end, for each key-point, there is a $4 \times 4 \times 8 = 128$ dimensional feature vector a.k.a. descriptor.



(a) Image gradients (b) Key-point descriptor

Figure 5.4: Key-point descriptor (source [5])

Fortunately, the generation of these descriptors is easily achievable with the OpenCV library [46]. Their generation can be seen in Listing 5.1.

```
1    # Compute SIFT
2    kp, des = sift.detectAndCompute(gray, None)
```

Listing 5.1: SIFT descriptors

5.2 Bag of words

This section explains how the SIFT descriptors are used to identify different categories of logos. The SIFT algorithm returns a variable number of 128 dimension descriptors for each image. For this reason, each image has a different number of descriptors and therefore, they cannot be used in a machine learning algorithm (a machine learning algorithm requires the same quantity of values for each image/element). To solve this problem, the Bag of Words (BoW), a.k.a. bag of features or bag of key-points can be used. The BoW is one of the most general and frequently algorithms used for categories' recognition.[51] The information presented here is based on [51, 52, 53, 54, 55].

5.2.1 Origins

In the world of Natural Language Processing (NPL) multiple documents, with different words, are compared and categorized for different purposes.[55] This task requires a specific model to analyse and classify different words' groups. For this reason, the BoW model has been invented. This model has been progressively used, and today one of its classic use is spam filtering. In this specific case, the system is trained to find the difference between spam and normal text.

In a more general case of the BoW model, the words are thrown into a bag. Then, the occurrences of each word in the corpus are counted, and the results are converted into histograms. This simple technique allows creating a features vector which can be used in machine learning algorithm. Usually, the corpus is composed of words extracted from a training set. This fixed size corpus is necessary because otherwise, the number of features in an element's descriptor might vary from case to case and therefore, the descriptors could not be used in a machine learning classifier. This constraint on the corpus size is a valid solution but consequently, new words are ignored.

To better understand this model, let's take an example from [52]. Modern English spelling is almost standardized and therefore, there is only a single version of the word *herself*. This is not the case for Middle English from the Canterbury Tales [56], which contains more version of *herself* such as *hire-self*, *herselven*, *hireselve*, etc. Therefore, if histograms are computed from a series of documents written in Middle English, the whole dictionary would contain multiple versions of the word *herself*. Unfortunately, the presence of these multiple terms makes their identification harder. To solve this problem, the different variants of the word must be clustered together, so that they represent the same class.

5.2.2 Bag of features

Because of the success of BoW in NPL, this model has been adapted for many computer vision tasks such as image classification, video search, robot localization and texture recognition.[53]

According to [53], there are two ways to explain the Bag of Words model used in image classification. For this reason, this section explains this technique with both approaches.

Classic explanation

The first approach is based on the BoW concept in NPL. In the normal BoW, a document is represented as a normalized histogram of words occurrences. As it has been explained in Section 5.2.1 firstly, a vocabulary (a.k.a. corpus) is built from all the words in the training set. Secondly, all the words of the document which are contained in the dictionary are counted to create a histogram. In fact, each element of the histogram is a word in the dictionary and therefore, each element's value represents the number of times the term appears in the document. A document in the Bag of Words model is called a *bag* because all ordering of the words has been lost [53]. The application in image classification is very similar. In this case, the vocabulary is generated by clustering all the descriptors extracted from a set of training images. These points represent words in the normal BoW

model. Unfortunately, images can have a lot of slightly different descriptors and therefore, a clustering is needed to create a more compact vocabulary as seen in the Canterbury Tales example. In fact, each descriptor's cluster can be visualized as a visual word. To process new images, their descriptors are extracted and then, they are assigned to the nearest cluster centre of the visual vocabulary. Then, as in the training, the number of occurrences for each word is counted, and the result is used to generate a histogram. Finally, the histogram can be used to classify the image.

Visual codebook approach

The second way to explain this technique is with a codebook perspective. The descriptors extracted from the training images which have passed a clustering phase, can be used to develop a visual codebook. Then, the descriptors of a new image are assigned to the nearest code in the codebook. In the end, the occurrences of each code are counted and a histogram is created.

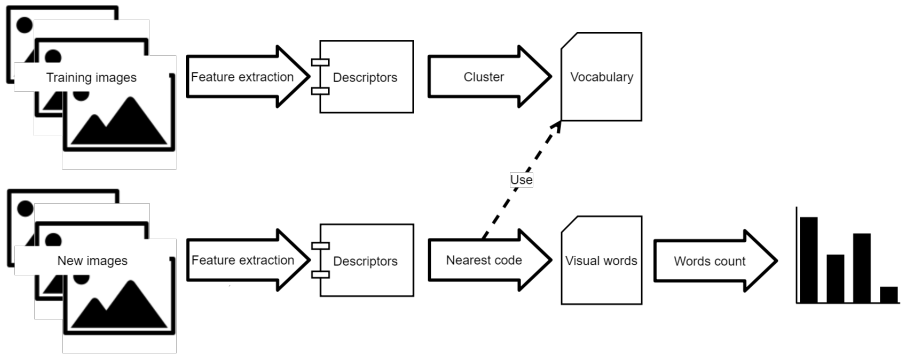


Figure 5.5: Bag of features work-flow

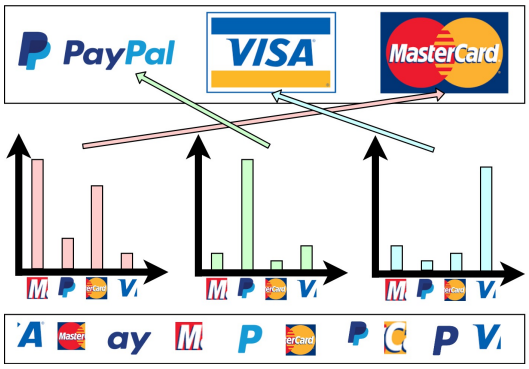


Figure 5.6: Bag of features work-flow simplified

The entire process is shown in Figure 5.5 and 5.6, but to simplify the explanation let's take as an example the recognition of logos.

A simple example

As it has been seen in Section 5.1, this application uses the SIFT descriptors. In Bag of Words a SIFT descriptor can be seen as a *word* and therefore, the BoW concept can be extended to image classification. This descriptor, a.k.a. a visual word could be seen as a representation of a specific part of the image such as the MasterCard logo circle. Unfortunately, the SIFT descriptors do not represent such precise piece of information but rather a bright zone which turns into darker ones. Logically, there are many variations of each logo part and therefore, a clustering phase is needed to group together zones which represent the same thing (see Figure 5.7).

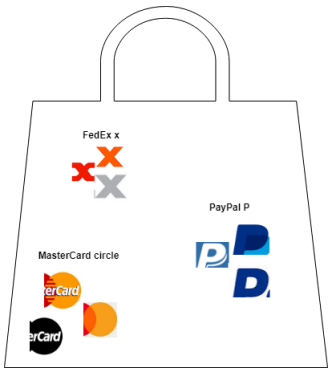


Figure 5.7: Bag of visual words

Once the dictionary or codebook is generated, all images a.k.a. SIFT descriptors, can be analysed and assigned to the bin they belong. In this process, all the MasterCard circle's descriptors are labelled as such. Then, the number of occurrences of each descriptor is counted and finally, a histogram of x values is generated. In this case, the number of visual words in the vocabulary is 3 and therefore, the histogram contains three values (see Figure 5.8).

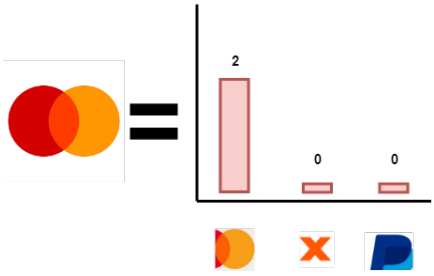


Figure 5.8: Histogram

Why is the clustering process so important? Unfortunately, the generated SIFT descriptors for a specific part of a logo could be slightly different for each image. This is a problem because similar points are seen as different. To solve this inconvenient, a clustering technique can be used. According to [54], clustering can be defined as the grouping of a set of objects in such a way that objects in the same group are much similar, then to those in other groups/sets. To perform clustering, there are different techniques, i.e. centroid oriented, K-Means, or distribution based models.[54] The technique most used in the BoW is K-Means which objective is to divide all the objects $X = \{x_1, x_2, ..., x_n\}$ into k clusters. In other words, its goal is to minimize the distance between each point (SIFT descriptor) and the assigned centroid. This step can be seen in Equation 5.1, where μ is the mean of all points in a cluster S_i, and S denotes the set of points partitioned into clusters $\{S_1, S_2, ..., S_i\}$.[54]

$$\arg\min \sum_{i=1}^{k} \sum_{x \in S_i} \|x - \mu_i\|^2 \tag{5.1}$$

Put more simply, according to [57], the algorithm starts with an initial estimation for k centres. This estimation can either be randomly generated or randomly selected from the dataset. Secondly, each data point is assigned to its nearest centre based on the squared Euclidean distance. Then, the centres are recomputed by taking the mean of all data points assigned to that cluster. This second step is executed multiple times until a stopping criterion is met (i.e. a specific accuracy, a specified number of iterations, etc.). The only problem with this algorithm is the choice of k because according to [52], the determination of this value is an art, not a science. Because in this thesis the different algorithms must be used/implemented with any optimization, the tests showed in Section 6.1 are executed only with four k values so that the time available for this project could be optimized. These four values are $100, 500, 1000, 10000$ and they have been chosen to see if a bigger k means better results.

5.2.3 The prototype

The Bag of Words implementation developed for this project follows the steps explained in Section 5.2.2. First, to be able to generate the vocabulary, the SIFT descriptors for each image are computed. Then, they are stored in a three-dimensional array that has an element for each processed image. Each array's element contains a different number of SIFT descriptors (images return a different quantity of SIFT descriptors). For this reason, to be able to perform the K-Mean clustering the array must be flattened as shown in Listing 5.2.

```
1   # computer descriptors
2   descriptors = self.compute_descriptors()
3
4   # Put data together
5   t_descriptors = descriptors[0][1]
6   for i, descriptor in enumerate(descriptors[1:]):
7     if isinstance(descriptor, np.ndarray):
8       if i > max_voc_size:
9         break
10      t_descriptors = np.vstack((t_descriptors, descriptor))
11
12  # t_descriptor len = total number of images features
13  # t_descriptor[x] len = 128 -> SIFT descriptor
14
15  # VOCABULARY GENERATION
16  # Stop the iteration when any of the condition is met (accuracy and max number of
      ↪ iterations)
17  criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER,
      ↪ iterations, 1.0)
18  # K-mean clustering, vocabulary has k words
19  ret, label, vocabulary = cv2.kmeans(np.float32(t_descriptors), k, None, criteria,
      ↪ attempts,
20                  cv2.KMEANS_RANDOM_CENTERS)
```

Listing 5.2: BoW with SIFT

This clustering process is executed ten times with k random centres. For each run, the algorithm stops after ten iterations or if every cluster centre has moved less than 1 unit in the last iteration. Once the dictionary of visual words is generated, a histogram for each image is created so that each image is represented by a fixed size vector. To create these histograms, a code from the codebook is assigned to each descriptor. Then, the occurrences of each code are counted for each image as it is shown in Listing 5.3. The generated histogram have k values as shown in Figure 5.9.

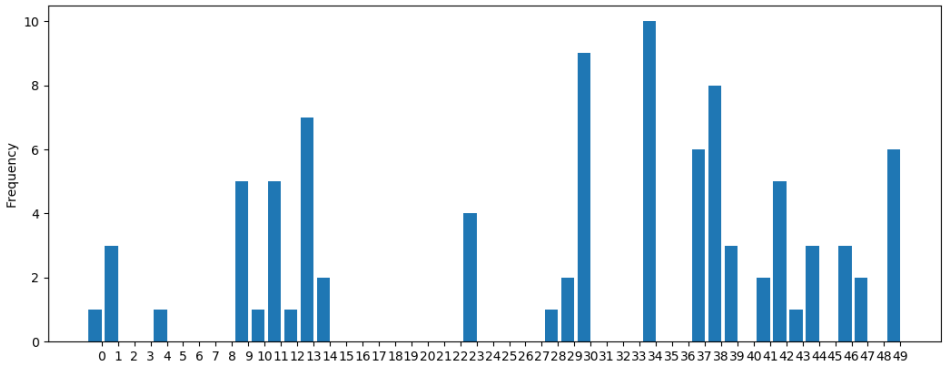


Figure 5.9: Histogram with k = 50

```
for i in range(len(self.images_path)):
  if isinstance(descriptors[i], np.ndarray):
    # Assign codes from a code book to observations.
    words, distance = vq(descriptors[i], vocabulary)
    for w in words:
      histograms[i][w] += 1
```

Listing 5.3: Histogram generation

In the end, the histograms are standardized because the classification techniques used in this project often prefer data that look more or less like a standard distribution (i.e. Gaussian with zero mean and unit variance). These standardized histograms are then used by the classifiers as explained in Section 6.1.

5.2.4 Final impression

This technique is really simple and it achieves good results (see Section 6.2), but it requires a lot of time to generate the histograms. In fact, the main problem of this algorithm is the time needed to perform the K-Mean clustering. Unfortunately, a good classification needs a lot of training images and therefore, the clustering step needs to process a lot of information. For the execution of this algorithm 642 images, a.k.a. 87942 128-dimensional SIFT descriptors have been used. The execution time of this algorithm depends on the number of clusters the K-Means must create. Table 5.1 shows the times of the training process with different k values. The K-Means has been executed with the default parameters of the OpenCV [46] implementation.

K-Means	**hh:mm**
100	$\sim 00:07$
500	$\sim 00:18$
1000	$\sim 00:32$
10000	$\sim 04:42$

Table 5.1: BoW training time

The BoW model created for this project has been developed without implementing any improvements in speed or other quality requirements. As it has been seen in this section, the time needed to execute this algorithm is quite long but fortunately, there are possible improvements which are described in [58, 59]. These improvements are not implemented in this project, but they could be applied in a future project.

6. Single logos classification

The previous chapters explained the definition of a classifier and the concept of Bag of Words. Unfortunately, these techniques alone do not allow the recognition or classification of images. For this reason, this chapter explains how their combination can be used to classify different groups of logos (logos as a single image). Firstly, different classification techniques with the BoW model are tested, so that the best technique for this task can be found. Then, a short description of another program called WEKA [60], which is an open source machine learning software, is made. In the end, to see if these classic classification techniques are still usable today, a short comparison between them and the ANNs is made.

6.1 Types of classifiers

To be able to identify the best classification technique to recognize logo as single images, five different classifiers are assessed. Furthermore, the presence of multiple techniques allows increasing the probability to achieve a better result. For this reason, this section explains the different classifiers concepts and lists their results. Then, to be able to choose the best technology, a comparison between them using the quality measures explained in Section 4.3 is made.

Before being able to begin the explanations and evaluations, the testing processes must be clarified. For the classification, each technique needs at least two classes because otherwise, all the logos will belong to the same class. For this reason, different types of logos have been downloaded from Google images, and they have been stored in two different directories. The first directory called *train* contains most of the logos (2/3), and it is used by the classifier to train. In most cases, the training is a simple computation of the logos' descriptors (see Section 5) and their insertion in a multi-dimensional space (to a deeper explanation refer to the specific technique). The second directory called *test* contains the rest of the downloaded logos (1/3), and it is used to test the different techniques. The logos contained in the test folder are not the same as the ones contained in the train folder. To be more

precise, they could be really similar, but they should not be the same exact images in term of resolutions or sizes. Furthermore, in addition to the logos downloaded, there are modified versions of them. These modifications, i.e. shearing, flipping, rotation, grey-scale, etc. are there to understand the real capabilities of each algorithm. As a clarification, the logos' partitions sizes (1/3 and 2/3) are the same as the usual quantities used in data mining [61].

The training and test phases use only two classes of logos instead of one for each logo's type (MasterCard, Visa, DHL, etc.). This choice allows to generate the values of the confusion matrix in a simpler way (see Section 4.2.1 and 4.2.2). With these settings, there is only a class for the processed logo (e.g. MasterCard) and a class for all the other types of logos. Unfortunately, all the results for each logo's type (i.e. MasterCard vs Other, PayPal vs Other, etc.) cannot be listed here because otherwise, the document would be composed only of tables. Whereas, the results are the averages of the confusion matrix values for each type of logo. This simplification does not allow seeing if a technique struggles to recognize some specific logo. For this reason, if a classifier has problems with a specific type, the issues are reported in the form of text.

Each of the following subsections introduces one of the five technologies and explains its working principles. Then, all the techniques are tested in the same way, so that at the end the best one can be found. The realisation of these classifiers' prototypes (algorithms) is based on scikit-learn [62].

6.1.1 Support vector machine

> A Support Vector Machine (SVM) is a discriminative classifier formally defined by a separating hyperplane. In other words, given labelled training data (supervised learning), the algorithm outputs an optimal hyperplane which categorizes new examples. In two-dimensional space, this hyperplane is a line dividing a plane in two parts wherein each class lay in either side
>
> (Savan Patel — [63])

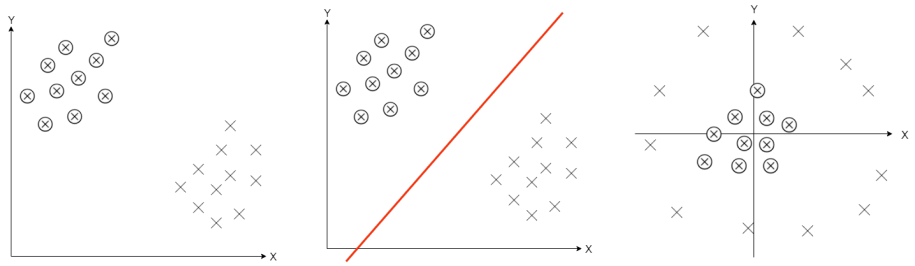
This sentence completely describes what is the SVM classifier, but to better understand the functioning of the algorithm let's elaborate further the concept. The information proposed here is based on [64, 6, 63].

Working principles

SVM is a supervised machine learning algorithm that can be used for classification and regression [64]. In this algorithm, the data are plotted in an n-dimensional space, where n is the number of the descriptors available.

Then, the system finds hyper-planes that divide the different classes, so that it can perform a classification.

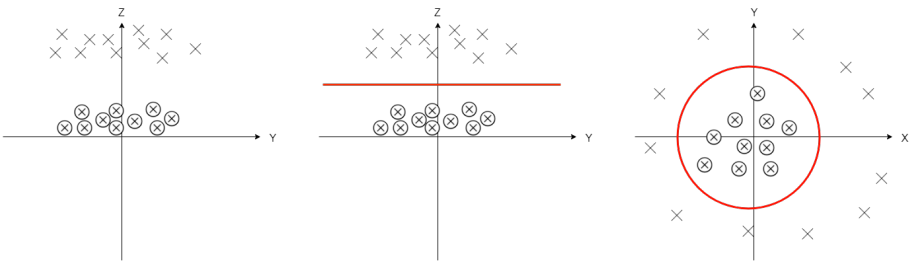
To better understand what is a hyper-plane let's take as an example the data points shown in Figure 6.1a.



(a) (b) (c)

Figure 6.1: SVM examples

In this case, it is easy to find out a line (hyper-plane) which separates the two classes as it is shown in Figure 6.1b. This is a simple example, but how can be found a line in Figure 6.1c? In this case, for most people, a line that separates the two classes is not visible. In fact, the solution is not so obvious because to solve the problem the high dimensionality of SVM must be used. Let's assume that another axis called z which is computed as follows $z = x^2 + y^2$, is added to the diagram. To be more precise, this new axis represents the distance of each point from the origin. Then, let's plot the result as shown in Figure 6.2a. Thanks to this new graph, the line that divides the two classes can be easily found (see Figure 6.2b). Now, if this new line is kept in the backward transformation, it becomes a circle as it is shown in Figure 6.2c. This is a very clever idea, in fact, it is one of the fundamentals of the SVM classifier. Indeed, this transformation is done a lot of time by this algorithm, and for this reason, it has its own name, namely *kernel*.



(a) (b) (c)

Figure 6.2: SVM complicated example

The SVM implementation of the sklearn library [62] used for this prototype, has different tuning parameters which can be used to refine the final result. The first thing to consider is that there are different types of kernels (linear, polynomial, Radial Basis Function (RBF), sigmoid). These types modify the separation line generated by the algorithm (see Figure 6.3).

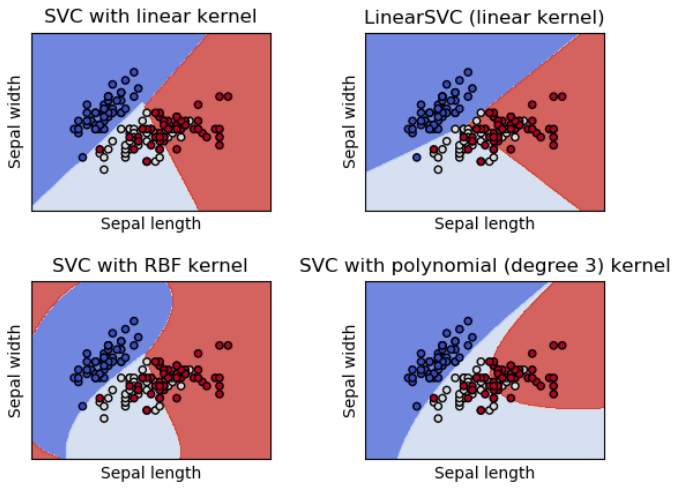


Figure 6.3: Types of kernels (source [6])

The second value to consider is the regularization parameter named *c*. It tells the algorithm how much misclassifying there must be. In other words, with a large *c*, the algorithm tries to get a hyperplane that classifies correctly all the values in the training sets (see Figure 6.4).

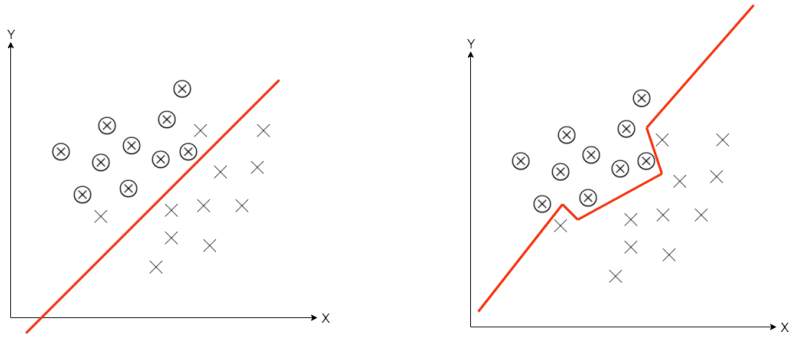
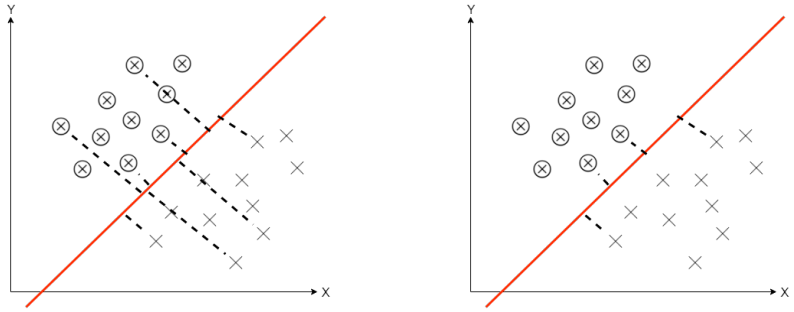


Figure 6.4: SVM C value

The third and last parameter is *gamma*. This value defines how far must be a point to influence the generation of the hyperplane. With low *gamma* values' points far away will also influence the generation of the separation line (see Figure 6.5).



(a) Low gamma (b) High gamma

Figure 6.5: SVM gamma value

For more information about the tuning parameters please refer to [64, 63].

Measurements

Originally the idea was to test the classifier by varying the three parameters described in this section and the number of clusters for the K-Means (see Section 5.2.2). Unfortunately, if the tests would be made with 5 different values for c $(1, 10, 100, 1000, 10000)$, 5 values for *gamma* $(10, 1, 0.1, 0.01, 0.001)$, all the five kernels and four k values the table generated would be too big $5 \times 5 \times 5 \times 4 = 500$ rows. For this reason, the number of possibilities has been reduced. According to [65], the best c value is 100. Unfortunately, the results of some tests with this value have shown only small improvements in the sensitivity and therefore, it has been decided to use the default value $c = 1$. In case of *gamma*, the article says that it must be as low as possible. Its default value decrease with the number of descriptors $gamma = \frac{1}{\text{n of features}}$. Therefore, it is perfect for this application. Now, with this reduction, the number of tests decreased to 5 kernels $\times$ 4 k parameters = 20.

k	Kernel	m:s.ms	TPR	TNR	PPV	NPV	ACC
100	Linear	00:01.125	71.5	93.2	71.8	94.1	89.6
	LinearSVC	00:01.187	74.8	90.7	65.4	94.7	88.1
	RBF	00:01.187	50.7	99.6	97.8	91.0	91.5
	Sigmoid	00:01.140	56.0	96.7	78.1	91.5	89.8
	Polynomial	00:01.218	40.8	95.6	65.2	89.4	86.8
500	Linear	00:05.187	74.2	95.0	74.2	94.7	91.3
	LinearSVC	00:06.296	86.5	89.7	62.7	96.8	89.0
	RBF	00:05.625	46.4	100.0	100.0	90.4	91.2
	Sigmoid	00:05.125	63.3	98.4	88.3	93.4	92.8
	Polynomial	00:06.203	32.2	98.4	78.5	88.2	87.6

1000	Linear	00:10.671	73.3	96.6	81.9	94.5	92.6
	LinearSVC	00:14.516	86.1	89.5	65.4	96.9	88.9
	RBF	00:11.453	41.2	100.0	100.0	89.5	90.2
	Sigmoid	00:10.071	70.9	98.7	92.4	94.4	94.0
	Polynomial	00:12.359	20.8	99.3	86.3	86.5	86.5
10000	Linear	02:04.265	78.3	98.8	93.2	95.8	95.4
	LinearSVC	03:11.375	92.3	88.7	64.1	98.2	89.0
	RBF	02:11.375	30.6	100.0	100.0	87.9	88.5
	Sigmoid	02:06.031	77.3	99.7	97.8	95.6	95.9
	Polynomial	02:12.469	6.0	100.0	100.0	80.2	80.4

Table 6.1: SVM evaluation

Table 6.1 shows all the tests executed for the SVM classifier. All the periods of time showed in this table represent only the classification process and not the generation of the data with the Bag of Words model. The only problem in this implementation occurred with the linear kernel and $k = 10000$. In this case, the classification process identified only MasterCard (MasterCard vs Other) and failed with all the other pairings.

As it has been shown in Section 4.3.1, the most important quality measure is the precision, a.k.a. PPV. The highest Positive Predictive Value for this implementation occurred with the RBF and polynomial kernels. Unfortunately, the TPR a.k.a. sensitivity is really low and therefore, this means that the two implementations recognize correctly very few logos. Luckily, the sigmoid kernel implementation achieves a good result as well. In fact, this implementation achieves a PPV of 97.8% and asensitivity of 77.3% with a k value of 10000. This is a good result because all the other values are also high and therefore, the implementation recognized a good number of logos minimizing at the same time the number of false positives.

6.1.2 Naive Bayes

The Naïve Bayes classifier is based on the Bayes theorem which describes the probability of an event based on the related events. This technique uses the Bayes law with a slight difference because it assumes the independence among predictors.[66] In other terms, this technique assumes that the presence of a particular feature in a class is unrelated to the presence of any other feature [66]. To be more precise, the theorem describes how to compute the probability of $P[H|E]$ from $P(H)$ (see Equation 6.1). The information proposed here is based on [67, 68, 66].

$$Pr[H|E] = \frac{Pr[E|H] \times Pr[H]}{Pr[E]} \tag{6.1}$$

Working principles

Since this algorithm uses probabilities, it is not so trivial to understand. For this reason, its working principles are explained with an example based on [61]. In this example the objective is to find the probability of a yes to the question *can I play outside?* knowing the evidence E : outlook = sunny, temperature = cool, humidity = high and windy = true. To be more precise, the probability must be determined using the data of Table 6.2. The first thing to do is to adapt Equation 6.1 for this example. In order to do this, the equation must be rewritten for multiple variables and then, its content must be replaced with the example's data. The result of these steps is visible in Equation 6.2.

$$Pr[y|E] = \frac{Pr[E_1|y] \times Pr[E_2|y] \times Pr[E_3|y] \times Pr[E_4|y] \times Pr[y]}{Pr[E]} \tag{6.2}$$

A careful investigation of Table 6.2 shows that almost all the necessary information for Equation 6.2 can be extracted without problems. The missing piece of data is the probability $Pr[E]$ that cannot be extracted so easily.

Outlook	**Temperature**	**Humidity**	**Windy**	**Play**
sunny	hot	high	false	no
sunny	hot	high	true	no
overcast	hot	high	false	yes
rainy	mild	high	false	yes
rainy	cool	normal	false	yes
rainy	cool	normal	true	no
overcast	cool	normal	true	yes
sunny	mild	high	false	no
sunny	cool	normal	false	yes
rainy	mild	normal	false	yes
sunny	mild	normal	true	yes
overcast	mild	high	true	yes
overcast	hot	normal	false	yes
rainy	mild	high	true	no

Table 6.2: Naïve Bayes Example

To find this unknown value and solve this problem, the opposite of the Equation 6.2 can be used (probability of a no to the question *can I play outside?* knowing the evidence E : outlook = sunny, temperature = cool, humidity = high and windy = true). With these two equations and a normalization process ($Pr[y|E] + Pr[n|E] = 1$) the denominator disappears as it is shown in Equation 6.3.

$$
\begin{aligned}
Pr[y|E] &= \frac{Pr[E_1|y] \times Pr[E_2|y] \times Pr[E_3|y] \times Pr[E_4|y] \times Pr[y]}{Pr[E]} \\
&= \frac{\frac{2}{9} \times \frac{3}{9} \times \frac{3}{9} \times \frac{3}{9} \times \frac{9}{14}}{Pr[E]} = 0.205 \\
Pr[n|E] &= \frac{Pr[E_1|n] \times Pr[E_2|n] \times Pr[E_3|n] \times Pr[E_4|n] \times Pr[n]}{Pr[E]} \\
&= \frac{\frac{3}{5} \times \frac{1}{5} \times \frac{4}{5} \times \frac{3}{5} \times \frac{5}{14}}{Pr[E]} = 0.795
\end{aligned}
\tag{6.3}
$$

According to [66], the Naïve Bayes classier uses a similar method to predict the probabilities of different classes based on various attributes.

Measurements

The Naïve Bayes classifier of sklearn [62] has three different implementations, i.e. Gaussian, Multinomial and Bernoulli. These implementations use a slight different equation to compute the probability. For example, the Naïve Bayes Gaussian classifier assumes that the likelihood of a descriptor is Gaussian (see Equation 6.4).[67]

$$
Pr[x_i|y] = \frac{1}{\sqrt{2\pi\sigma_y^2}} exp(-\frac{(x_i - \mu_y)^2}{2\sigma_y^2}) \tag{6.4}
$$

To find out which of the implementations has the best results, the classifier has been tested with the three types and with the four k values of the K-Means as it has been already seen in Section 6.1.1.

k	Type	m:s.ms	TPR	TNR	PPV	NPV	ACC
100	Gaussian	00:01.078	69.6	66.7	30.5	91.5	67.3
	Multinomial	00:00.789	66.6	89.0	59.8	93.1	85.1
	Bernoulli	00:01.218	61.4	90.0	61.2	92.3	85.3
500	Gaussian	00:04.875	54.3	91.3	61.4	90.9	85.1
	Multinomial	00:03.656	69.2	95.2	79.6	93.9	90.8
	Bernoulli	00:04.937	63.8	95.2	77.5	93.2	90.1
1000	Gaussian	00:09.296	42.2	97.9	80.0	89.4	88.5
	Multinomial	00:07.062	75.6	96.1	83.2	95.1	92.6
	Bernoulli	00:09.156	69.3	96.6	83.4	94.2	92.2
10000	Gaussian	01:33.359	63.8	99.8	99.1	93.4	93.9
	Multinomial	01:14.016	83.4	96.2	85.4	96.5	93.8
	Bernoulli	01:33.265	78.1	98.4	91.2	95.8	95.1

Table 6.3: Naïve Bayes evaluation

Table 6.3 shows the results of the tests executed with the different parameters. As it has been already said for the Support Vector Machine classifier, the periods of time listed here represent only the classification process and not the generation of the data with the BoW model. The only problem occurred with these tests was with the multinomial classifier. This implementation does not allow negative values. Unfortunately, to use the other classifiers it is recommended to standardize the values with a standard scaler. This process removes the mean and scales the descriptors to unit variance. Unfortunately, this step leads to negative values which are not suitable for the multinomial classifier. To solve this problem, the tests were executed without standardization, so that all the values were positive.

The highest Positive Predictive Value has been achieved with the Gaussian classifier and $k = 10000$. Sadly, this implementation has a low sensitivity (63.8%) and therefore, it could not achieve a very good classification. Fortunately, there is a better solution. The Bernoulli implementation with $k = 10000$ achieves a high precision (91.2%) with also a good sensitivity (78.1%). This is a good alternative if the decrease of 8.1% in the PPV has not a high impact on the results.

6.1.3 K-nearest neighbours

K-Nearest Neighbour (KNN), as stated in [7, 69], can be used for both classification and regression. Moreover, it is a non-parametric lazy learning algorithm and one of the simplest classification techniques. According to [70] because of KNN is a non-parametric method, it has often success where the decision boundaries are very irregular. What is a non-parametric algorithm? A non-parametric technique does not make any assumptions on the underlying data distribution.[69] In other words, a lazy non-parametric algorithm does not use the training data to do any generalization [69]. For this reason, the training phase should be faster than the other algorithms but consequently, the training data is needed during the test phase.

Working principles

As the name can suggest, the objective of K-Nearest Neighbour is to find the k-nearest neighbours to the tested point. There are two main ways to define the number of neighbours. The first method is based on a constant defined by the user (k-nearest neighbour learning), while the second way is based on the local density of points (radius-based neighbour learning). Unfortunately, as reported by [70], the second technique becomes less effective in high-dimensional parameter spaces due to the so-called *course of dimensionality* (for more information please refer to [71]). For this reason, radius-based neighbours learning is not studied in this section.

K-Nearest Neighbour searches the k-nearest neighbours, but how it computes the distance between points? There are different functions to compute the distances, i.e. Euclidean, Manhattan, Minkowski, etc. Scikit-learn [62], whic is the library used for the implementation, uses the Minkowski distance (see Equation 6.5) which corresponds to the standard Euclidean distance when $p = 2$ and to the Manhattan distance when $p = 1$.

$$\left(\sum_{i=1}^{k} (|x_i - y_i|)^p \right)^{\frac{1}{p}} \tag{6.5}$$

Figure 6.6 shows an example of the KNN algorithm with $k = 3$. In this example, the objective is to assign the point x to the blue or green class. In this case, the new data point is assigned to the blue class because the majority (in this case three) of its neighbours are of that class.

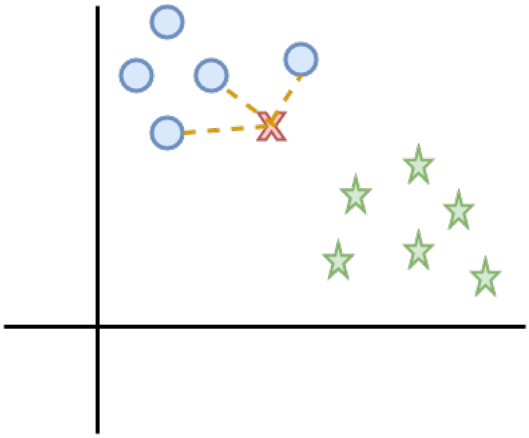


Figure 6.6: K-Nearest Neighbour example

This example is very simple and therefore, the discovery of the neighbours is not a problem. Unfortunately, in a more complicated example where there are more dimensions the task become more complicated. The first thing people think is to use a brute force to compute distances between all pairs of points in the dataset. For N samples in D dimensions this approach scale as $O(DN^2)$ and therefore it can be very competitive for small data samples. [70] Unfortunately, in this project there are a lot of dimensions and samples therefore, this is not the best choice. To solve the computational inefficiencies of the brute force, a tree-based data structures can be used. K-D tree is an example of these structures and it attempts to reduce the number of computations by storing distances information. The idea behind this structure is that if point A is very far from point B, and point B is very close to point C, then A and C are very distant.[70] This simple idea can reduce the computation to $O(DN \log(N))$. Unluckily, this data structure

becomes inefficient as D grows very large for the manifestation of the *course of dimensionality* which has been already cited early in this chapter. [70] To solve this high dimension problem, the ball-tree structure was invented. This special data structure uses hyper-spheres (n-sphere) to partition the data. This expedient allows decreasing the computational cost to $O(D \log(N))$.[70]

Until now, it has been shown how the algorithm works without looking to the number of neighbours. According to [7], if the KNN is applied to the training samples, the error rate increase continuously with the increase of k. In this case, there is a special situation with $k = 1$. With this configuration, the error rate is always zero because for the training samples the closest point to any training data is always itself (see Figure 6.7).

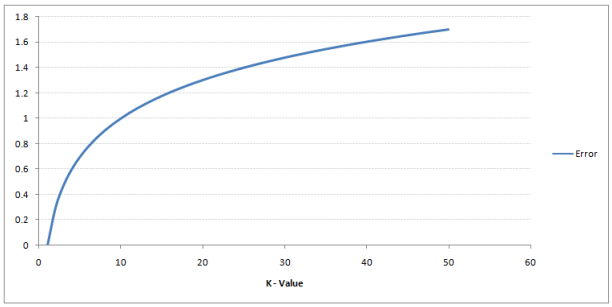


Figure 6.7: K-Nearest Neighbour error rate training samples (source [7])

On the other hand, if the error rate is measured on the test set of [7], it decreases and reaches a minimum with k = 10 (see Figure 6.8). Therefore, to get the optimal k value the algorithm must be executed by starting with k = 1 until a minimum is reached.

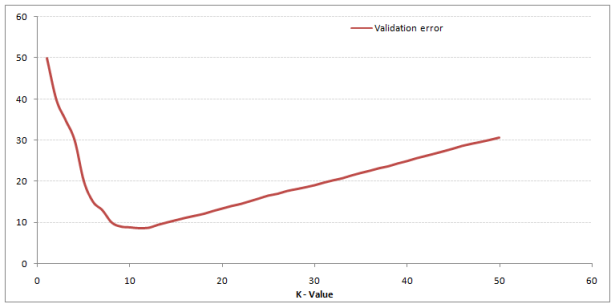


Figure 6.8: K-Nearest Neighbour error rate test samples (source [7])

Measurements

The sklearn [62] KNN implementation can be used with the three data structures explained in the previous section (KD-tree, ball-tree and brute

force). Because their performances are not known, for this implementation the choice is left to the library that decides which technique to use based on the number of samples, dimensions, and other parameters (refer to [70] for more information). On the other hand, to find the minimum error rate, different neighbours values (k) have been used. In addition, as it has been seen with the other classifiers, the implementation is tested with different K-Means values.

k	Neighbors	m:s.ms	TPR	TNR	PPV	NPV	ACC
100	9	00:01.171	65.8	95.1	72.7	93.5	90.4
	21	00:01.171	61.1	96.6	79.9	92.8	90.9
	39	00:01.187	54.5	97.4	79.7	91.8	90.5
500	9	00:05.421	57.0	97.4	77.8	92.7	91.3
	21	00:05.421	50.5	96.6	76.1	91.6	89.4
	39	00:05.421	41.9	96.4	90.9	90.1	87.8
1000	9	00:10.703	57.0	93.6	79.5	92.6	88.4
	21	00:10.671	47.5	98.2	92.9	91.2	90.4
	39	00:10.687	36.4	99.8	97.4	89.4	89.9
10000	9	01:44.500	38.0	99.5	95.0	85.9	86.7
	21	01:44.515	90.0	45.0	24.7	95.7	52.5
	39	01:44.422	100	2.0	16.9	100	18.3

Table 6.4: K-Nearest Neighbour evaluation

Table 6.4 shows all the results of the tests with different numbers of neighbours. As in the case of the SVM classifier, the periods of time showed here represent only the classification process and not the generation of the data with the BoW model. This implementation had some problems in fact, all three different numbers of neighbours (9, 21, 39) had difficulties with $k = 1000$. To be precise, the 9-KNN failed to recognize the pair MasterCard vs Other while the 21 and 39-KNN failed to recognize the pair PayPal vs Other. In this table, the highest PPV value is achieved with the 39-KNN and $k = 1000$. Unfortunately, as already said, this implementation had problems with the recognition of the pair PayPal vs Other. In addition to this, it has a really low sensitivity (36.45%) and therefore, it is not a suitable for a real implementation. The highest precision achieved without problems is generated by the 39-KNN with $k = 500$. This implementation has a precision of 90.9% but unfortunately, it has always a low sensitivity (41.9%).

6.1.4 Decision tree

According to [72, 73], tree-based learning algorithms are non-parametric methods (refer to Section 6.1.3 for non-parametric description). Moreover,

they are considered to be the best and mostly used supervised learning techniques. These methods are used a lot because they have high accuracy, stability, and they are easy to understand. These models, like the algorithm seen in Section 6.1.1 can be used for both regression and classification. Tree-based learning algorithms do not include only decision tree but also random forests which are studied in Section 6.1.5. The information proposed here is based on [74, 72].

Working principles

A decision tree, as the name can suggest, is a tree that split the sample into two or more homogeneous sub-samples. This split is based on most significant differentiator in input variables [72]. How can an algorithm be represented as a tree? To answer this question let's make an example using the data of Table 6.2.

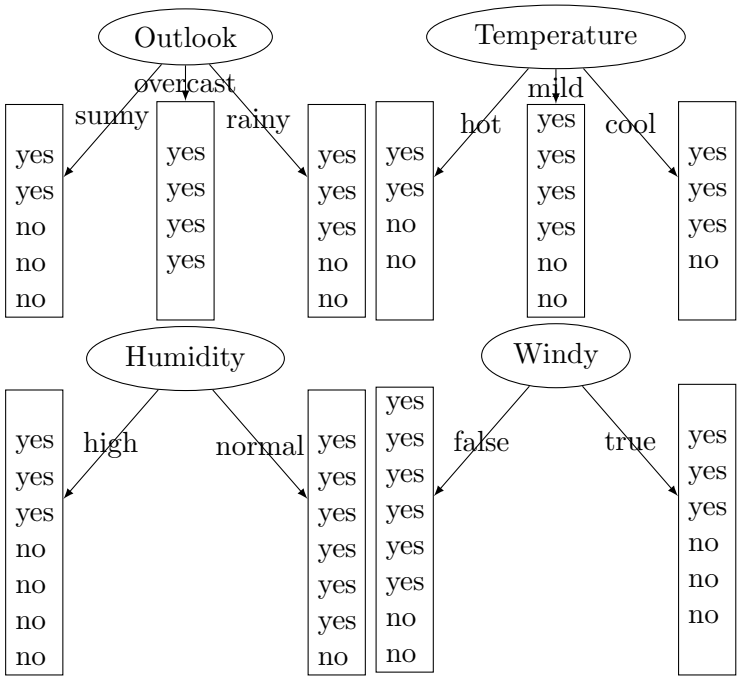


Figure 6.9: Decision Tree Example

In this example, there are 14 entries which tell if the weather conditions are enough good to play outside. Here, the objective is to generate a tree which predicts, based on the more significant input variables, if someone can play outside. In fact, the decision tree tries to identify the variable (outlook, temperature, humidity, windy) which has the best homogeneous sets of answers (yes/no). In this case, the task of identifying the best variable is difficult because, as shown in Figure 6.9, the best variable cannot be

immediately found. Therefore, if the decision is so difficult for a human being, how can the algorithm succeed? To make this decision, the system uses multiple algorithms which decide if a node must be split in two or not. In other words, the algorithms try to increase the homogeneity of new sub-nodes, so that the purity of the node increases with respect to the target variable.[72] As already said, there are multiple algorithms to split a node but here are presented only the two available in scikit-learn [62].

The default algorithm for splitting in scikit-learn [62] uses the *Gini* impurity which, according to [72], is defined as *if two items are selected from a population at random then they must be of same class and probability for this is 1 if the population is pure.* The formal definition of *Gini* impurity, a.k.a. index is available in Equation 6.6. According to [74], a perfect class purity occurs when a group contains only inputs of the same class. In other words, a perfect purity $Gini = 0$ can be seen as $p_i = 0$ or $p_i = 1$. On the other hand, the worst purity $Gini = 0.5$ happens when a node has a 50—50 split which translates to a p_i of 0.5.[74]

$$Gini = \sum_{i=1}^{J} p_i \times (1 - p_i) = 1 - \sum_{i=1}^{J} p_i^2 \tag{6.6}$$

Let's now compute which of the four variable is the best. The first thing to do is to compute the different indexes for all sub-nodes. Then, the weighted index of each variable must be computed (see Equation 6.7). To save space, the equation contains only two of the four variables.

$$\begin{aligned}
\text{Outlook} =& \frac{5}{14}(1 - \left(\left(\frac{2}{5}\right)^2 + \left(\frac{3}{5}\right)^2\right))_{\text{sunny}} \\
&+ \frac{4}{14}(1 - \left(1^2 + 0^2\right))_{\text{overcast}} \\
&+ \frac{5}{14}(1 - \left(\left(\frac{3}{5}\right)^2 + \left(\frac{2}{5}\right)^2\right))_{\text{rainy}} = 0.342 \\
\text{Temperature} =& \frac{4}{14}(1 - \left(\left(\frac{2}{4}\right)^2 + \left(\frac{2}{4}\right)^2\right))_{\text{hot}} \\
&+ \frac{6}{14}(1 - \left(\left(\frac{4}{6}\right)^2 + \left(\frac{2}{6}\right)^2\right))_{\text{mild}} \\
&+ \frac{4}{14}(1 - \left(\left(\frac{3}{4}\right)^2 + \left(\frac{1}{4}\right)^2\right))_{\text{cool}} = 0.445
\end{aligned} \tag{6.7}$$

These weighted *Gini* indexes can be used to identify the best value and therefore, where the three must be split. In this case, the best value is *outlook* because it has the lowest result. To compute the entire tree these steps must be repeated with the split values and variables (see Figure 6.10).

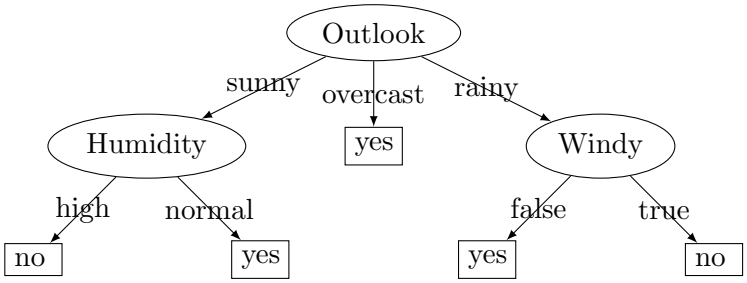


Figure 6.10: Decision Tree Example Solution

The second algorithm in scikit-learn [62] uses information gain and entropy to find the best node. The steps of this algorithm are quite similar to the one seen above. Contrary to the other, it uses another value to decide where the split must take place. To be more precise, this algorithm uses a value from the information theory which defines the degree of disorganization in a system. This value is called *entropy* and it is described in Equation 6.8.

$$entropy(p_1, p_2, ..., p_n) = -\sum_{i=1}^{n} p_i \times \log_2(p_1) \tag{6.8}$$

Here, like the previous algorithm, the weighted entropy is computed for each variable. To save space, the equation contains only two of the four variables. In addition, in this case, the root node entropy must be computed as well (see Equation 6.9).

$$\begin{aligned}
\text{Outlook} =& \frac{5}{14}(\frac{2}{5} \times \log_2(\frac{2}{5}) + \frac{3}{5} \times \log_2(\frac{3}{5}))_{\text{sunny}} \\
&+ \frac{4}{14}(1 \times \log_2(1) + 0 \times \log_2(0))_{\text{overcast}} \\
&+ \frac{5}{14}(\frac{3}{5} \times \log_2(\frac{3}{5}) + \frac{2}{5} \times \log_2(\frac{2}{5}))_{\text{rainy}} = 0.693 \\
\text{Temperature} =& \frac{4}{14}(\frac{2}{4} \times \log_2(\frac{2}{4}) + \frac{2}{4} \times \log_2(\frac{2}{4}))_{\text{hot}} \\
&+ \frac{6}{14}(\frac{4}{6} \times \log_2(\frac{4}{6}) + \frac{2}{6} \times \log_2(\frac{2}{6}))_{\text{mild}} \\
&+ \frac{4}{14}(\frac{3}{4} \times \log_2(\frac{3}{4}) + \frac{1}{4} \times \log_2(\frac{1}{4}))_{\text{cool}} = 0.911 \\
\text{Root} =& \frac{9}{14} \times \log_2(\frac{9}{14}) + \frac{5}{14} \times \log_2(\frac{5}{14}) = 0.940
\end{aligned} \tag{6.9}$$

Once all the entropies are computed, they are subtracted from the root entropy so that the gain of each variable can be found (see Equation 6.10).

$$\begin{aligned}
\text{gain(Outlook)} &= 0.940 - 0.693 = 0.247 \\
\text{gain(Temperature)} &= 0.940 - 0.911 = 0.029
\end{aligned} \tag{6.10}$$

Contrary to the previous case, here the split is executed on the variable with the highest gain which is *outlook*. To complete the tree, a new entropy and gain for each remaining variable must be computed from the parent node. In this example, the solution is equal to the one found with the *Gini* index (see Figure 6.10 but it must be remembered that these two algorithms can have different results.

Measurements

As it has been already said, scikit-learn [62] offers two types of algorithms (*Gini* and *entropy*) therefore, this technology has been tested with both implementations and four different k values for the K-Means.

k	Type	m:s.ms	TPR	TNR	PPV	NPV	ACC
100	Gini	00:01.062	59.8	89.7	57.2	91.6	84.5
	Entropy	00:01.171	60.5	93.5	66.4	91.8	87.8
500	Gini	00:04.984	66.3	89.9	59.8	93.0	86.2
	Entropy	00:04.968	66.9	93.3	69.3	93.2	88.9
1000	Gini	00:09.234	66.9	94.6	72.0	93.7	90.2
	Entropy	00:09.125	69.3	93.9	71.2	94.0	90.0
10000	Gini	01:37.312	80.9	96.8	84.5	96.1	94.1
	Entropy	01:36.625	77.6	96.7	82.9	95.5	93.4

Table 6.5: Decision tree evaluation

Table 6.5 shows all the tests' results with the different values for the decision tree. As in the other classifier, the periods of time showed here represent only the classification process and not the generation of the data with the Bag of Words (BoW) model.

With this classifier, the highest precision is achieved with $k = 10000$ and the Gini index. To be more precise, both the PPV and TPR are around $80\% - 85\%$ and probably they can become even better with higher K-Means values. Unfortunately, as Section 5.2.3 has shown, the time to train the data with $k = 10000$ is already enormous and therefore, higher k values could be used only if the time needed to train the system is not an important factor.

6.1.5 Random forest

As it has been already explained in Section 6.1.4, tree-based learning algorithms are considered to be one of the best-supervised learning technique and to be able to handle both regression and classification. In addition to this, according to [72], this algorithm is considered as the panacea of all data science problems.

Working principles

Contrary to the decision tree (Section 6.1.4) a random forest has multiple decision trees which give a vote.[72] In other words, the forest gives as answer the class with most votes. According to [75], there are two stages in the random forest algorithm. The first one is the random forest formation and the second one is the prediction's generation from the results received from the forest. To build up the forest the first thing to do is to pick k random features from the total m features (K << m). Then these K features are used to create a tree (with both techniques studied in Section 6.1.4). Now, to create a forest of n trees, these two steps must be repeated n times.

Finally, to have the classification each tree gives a prediction. Then, the number of occurrences for each class is counted, and the highest voted class is taken as the final prediction. Figure 6.11 shows a random forest which gives as classification the red class.

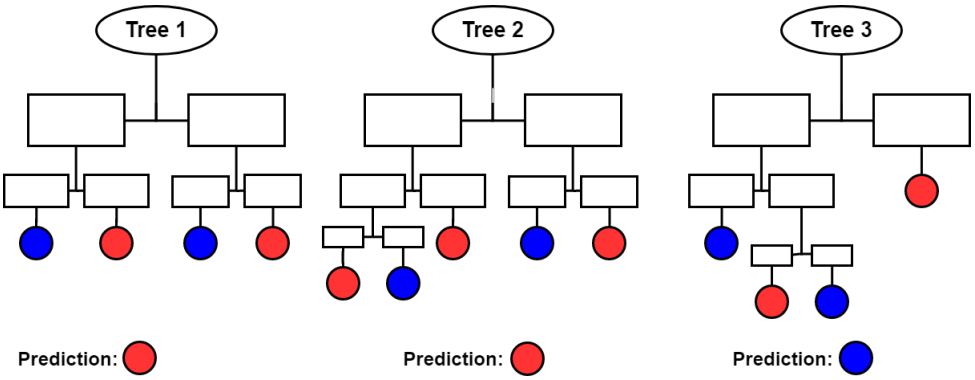


Figure 6.11: Random forest example

Measurements

To test this algorithm that relies on decision trees, it has been decided to use the technique that gave the best result in Section 6.1.4 (i.e. *Gini* index). In addition, the tests have been executed with a variable number of trees for the forest (10, 100, 1000, 10000) and the usual k values for the K-Means.

k	Type	m:s.ms	TPR	TNR	PPV	NPV	ACC
100	10	00:01.140	58.0	96.4	77.6	92.1	90.1
	100	00:02.140	53.8	98.5	91.3	91.6	91.2
	1000	00:11.765	51.5	98.8	93.6	91.3	91.0
	10000	01:48.031	52.3	98.7	91.8	91.4	91.1
500	10	00:05.015	69.3	97.4	83.8	94.2	92.8
	100	00:06.046	65.3	99.6	98.6	93.8	94.2
	1000	00:17.171	62.6	99.9	99.6	93.3	93.9
	10000	02:27.268	61.9	99.9	99.6	93.2	93.8

1000	10	00:09.156	71.0	97.9	88.4	94.6	93.5
	100	00:10.265	68.3	99.7	98.9	94.2	94.6
	1000	00:21.437	66.3	99.8	99.3	93.9	94.4
	10000	02:17.687	67.9	99.8	99.3	94.1	94.6
10000	10	01:32.531	76.9	97.7	87.3	95.5	94.3
	100	01:39.734	78.0	99.7	98.3	95.8	96.1
	1000	02:48.734	77.0	99.8	98.7	95.7	96.1
	10000	14:09.158	77.6	99.8	98.1	95.8	96.1

Table 6.6: Random forest evaluation

Table 6.6 shows the results of the tests made with the random forest classifier. As with the other classifiers, the period of time showed here represent only the classification process and not the generation of the data with the BoW model. This classifier has the best results, in fact, it achieves high results with all quality measurements. The highest PPV is achieved with $k = 500$ and a population of 1000 trees. Unfortunately, in this case, the sensitivity is low (62.6%) and therefore this implementation is not the most suitable for this project. Luckily, the test with $k = 10000$ and a population of 1000 trees achieves a much better result. With these parameters, the precision is a little lower than before (98.7%) but the sensitivity is much higher (77.0%). With these settings, this classifier has the best results so far but it comes with a little disadvantage. Sadly, the time to perform the required operations is greater than the other implementations and the impact on the usability of a possible application is very high. Fortunately, the quality requirements are of very low priority in this project and therefore, this problem can be omitted.

6.2 Results assessment

This section assesses the results achieved with the different types of classifiers seen in the previous sections. To be sure about the different results, all the tests have been executed also without data standardization (see Section 5.2.3 for more information about data standardization).

Classifier	**k**	**TPR**	**TNR**	**PPV**	**NPV**	**ACC**	**Score**
Sigmoid-SVM	10000	77.3	99.7	97.8	95.6	95.9	94.0
Naive-Gauss	10000	63.8	99.8	99.1	93.4	93.9	91.5
Naive-Bernoulli	10000	78.1	98.4	91.2	95.8	95.1	91.6
39-KNN	500	41.9	96.4	90.9	90.1	87.8	83.0
Gini-Tree	10000	80.9	96.8	84.5	96.1	94.1	89.5
1000-Forest	10000	77.0	99.8	98.7	95.7	96.1	94.3

Table 6.7: Classifiers evaluation

Table 6.7 contains the best classifier-settings pairs which achieved the best results in each category. To simplify the assessment of the quality measures of the confusion matrix, a simple score has been created. This new value is a simple sum of all parameters with the PPV counted two times (see Equation 6.11).

$$Score = \frac{1}{6} \times (TPR + TNR + 2 \times PPV + NPV + ACC) \tag{6.11}$$

As it has been already predicted, the best classifier-settings pair is the random forest with a population of 1000 trees. This pairing is followed by the Support Vector Machine with a sigmoid kernel which has a score difference of 0.3%. These two implementations are almost the same because the former has a little higher score but with an execution time of 02m:48s.734ms which is 46s.703ms longer than the latter. This time difference compensates for the lower result of the SVM implementation. As it has been already said multiple times, the execution time and the quality requirements are not important in this project and therefore, the best implementation remains the random forest. For this reason, this technology will be used for further comparisons and for the final implementation.

6.2.1 A possible improvement

In the introduction of Section 6.1, it has been said that the classifiers would use only two classes instead of one for each logo's types. This allowed to compute the confusion matrix values in a simpler way. Now that the best classifier has been found let's test it with six different classes to see if its results could improve even more.

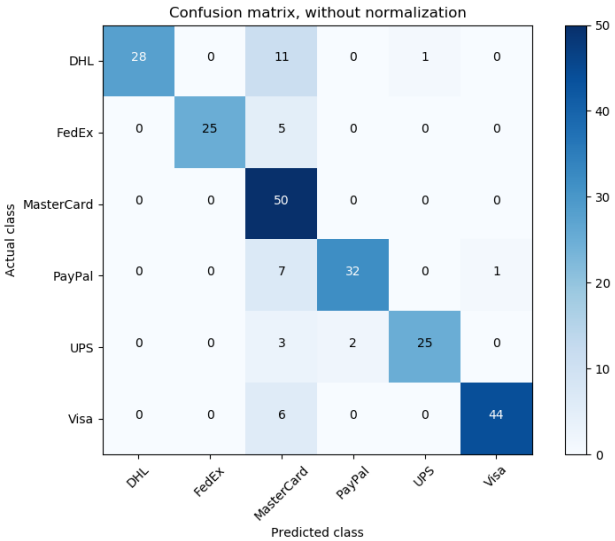


Figure 6.12: Confusion matrix with six classes

Unfortunately, as it can be seen from the Figure 6.12, the performances of this implementation are lower. In fact, it achieves a TPR of 84.1%, a TNR of 96.9%, a PPV of 91.5%, a NPV of 97.1% and an accuracy of 95.0%. To be in compliance with the evaluation seen in Section 6.2 this implementation has a score of 92.5% which is less than the original implementation.

6.3 An introduction to WEKA

The Weka workbench [76] is a collection of state-of-the-art machine learning algorithms and data processing tools [76]. This program includes all the algorithms described in [76] and it is designed to be flexible to be able to try different methods on different datasets. The most interesting feature of this program is its ability to visualize the input data and the results of learning processes. In this project, the visualisation of the algorithms could be a good idea to see how the random forest (see Section 6.1) works and how it is structured. Unfortunately, this program uses a specific file format called Attribute Relation File Format (ARFF) which has the form of a single relation table.[76] The prototypes developed for this project do not use this type of format and therefore this program cannot be easily used. Unfortunately, the modification of the prototypes to generate this type of files would require a lot of time and therefore it is not realisable in this project. In addition to that, probably the representation of the trees used in the random forest would take a lot of space because they should have a lot of leaves and therefore, they should not be easily representable in an image like Figure 6.13.

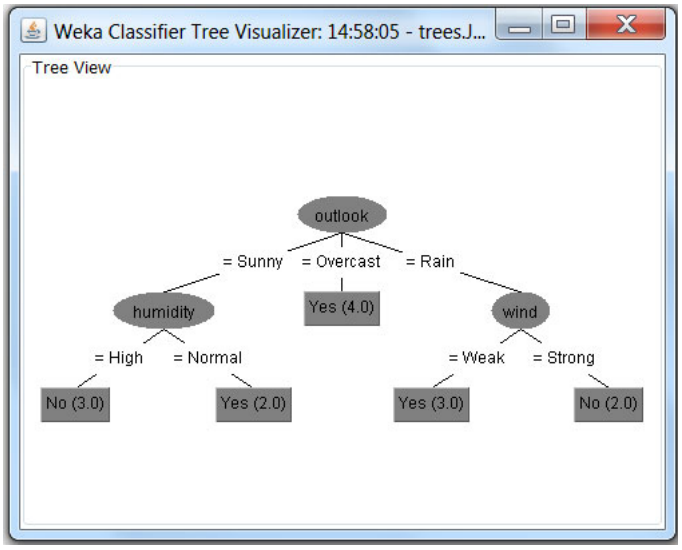


Figure 6.13: WEKA tree visualisation example (source [8])

6.4 A possible breakthrough

The methods tested in this project are classic computer vision tasks where the different features, steps, etc. of the process are known. In the last few years the world of computer vision and images classification has been invaded by a new technology called Artificial Neural Networks (ANNs). This new method, which is not really stable for reasons that are not dealt here, has become the hype in many computer science fields.[77] One of these domains, as already said, is the image recognition where the Convolution Neural Networks (CNNs) have become a good method to classify objects. To understand if this technology will replace the classic methods let's take an example with the Neural Network Toolbox of MatLab [78]. As a premise, it must be said that the implementation of this test has been made without studying in detail the concepts of ANNs and CNNs ~~and therefore no further explanation is present in this document~~. [**update 03.04.2018**] Because the optional requirements of this thesis changed, the ANNs are studied instead of other sources of information (see Chapter 9). The results presented here are the outcome of a combination of the BoW model with $k = 10000$ and CNNs. For simplicity, in this case, logos are classified all together and not divided into pairs (i.e. MasterCard vs Others).

Figure 6.14: ANN confusion matrix

As it can be seen from Figure 6.14, the results of this implementation are really good. The test's confusion matrix has a TPR of 98.7%, a TNR of 99.6%, a PPV of 98.7%, a NPV of 99.6% and an accuracy of 99.4%. These values give a score (see Section 6.2) of 99.1% which is the highest score amongst all. This result shows how this technique, that does not need a lot of time to set up, could do better of all other techniques in image classification.

7. Logos in larger images

The previous chapters explained the process of descriptors extraction from images (see Chapter 5) and how to use them to classify the image itself (see Chapter 6). Unfortunately, images or better logos are almost always contained in a larger image as shown in Figure 7.1.

(a) (b)

Figure 7.1: Logos in larger images

Here arises a problem because if the Bag of Words model is applied i.e. to Figure 7.1b, it would generate a too generic histogram and therefore, it would assign a wrong class to the image. To solve this problem, according to [30], an algorithm to localize objects or sub-regions (patches) in an image must be used. One of the first and simpler approach that a person thinks about is the sliding window's algorithm. In this case, a box or window is slit over an image to select a patch. Then, each patch is classified with a normal classification technique. Unfortunately, this window not only should slide overall image but also in different scales. This process even if it is an exhaustive search, it needs thousands of patches comparisons. Fortunately, this issue is solved with a region proposal algorithm that take an image as input and generate bounding boxes that are most likely an object.

7.1 Object detection

As it has been shown in the introduction, objects can be found in an image with region proposal algorithms. These methods identify objects in an image using segmentation where similar adjacent regions are grouped together using criteria such as colour, texture, etc. [30] These algorithms allow to

reduce the number of image patches to classify. Unfortunately, these region proposals methods can create overlapping windows which may not contains the searched object. In fact, according to [30], region proposal method has a very high recall a.k.a. sensitivity. This means that the generated windows have to contain the searched objects. In fact, these types of algorithms generate a lot of false positives patches to be sure to find also the true positives. As it has been seen in Section 3.2, there are several region proposal methods but because of the short duration of this project, only one approach is studied.

7.2 Selective search

According to the author of the selective search algorithm [9], segmentation was invented to create a unique partitioning of an image where there is one part for all object silhouettes. Unfortunately, this task is not always obvious because images are intrinsically hierarchical. For this reason, in the majority of the segmentation applications multiple scales are a necessity. In addition, there is another problem. In fact, regions with very different characteristics can only be combined together if the object is already identified (for example a face and a jumper)[9]. For this reason, the traditional approaches to perform localisation through the identification of an object have been put aside.

As it has been already said, the first method to perform an exhaustive search is the sliding window. Unfortunately, an object can be located at any position and scale in the image and therefore this type of exhaustive search is computationally expensive. To solve this problem, [9] used a bottom-up segmentation with the principles of an exhaustive search so that the structure of the image could be exploited to generate object locations. In fact, according to [9], the algorithm's goal is to generate a class independent and data-driven selective search strategy that generates a small set of high-quality object locations. For this reason, selective search can be applied without problem for object recognition.

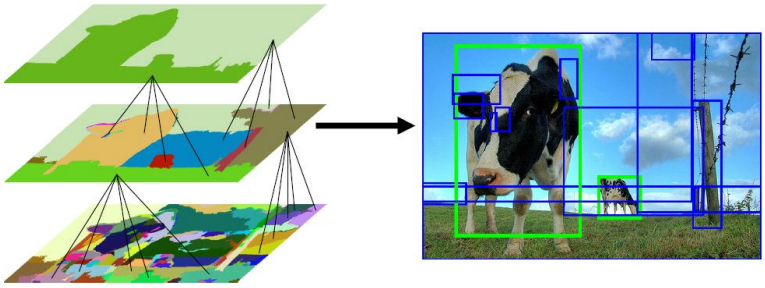


Figure 7.2: Selective search bottom-up grouping (source [9])

Objects can occur at any scale within the image and therefore, in the selective algorithm, all object scales should be taken into account. For this reason, an adapted version of the bottom-up grouping is used. To be more precise, because the image is intrinsically hierarchical, locations in all scales can be found by continuing the grouping process until the whole image becomes a single region (see Figure 7.2). To be more precise, first small starting regions are generated with region-based features. Then, a greedy algorithm is used to iteratively group regions together. To do so, similarities between all neighbouring region are calculated and the two most similar regions are grouped together. Because objects in an image can be grouped by colour, texture, etc., no single optimal strategy exists. For this reason, the hierarchical grouping algorithm is performed in a variety of colour spaces, with different similarity measures and with variation in the starting regions.

7.2.1 Implementation and results

Fortunately, OpenCV [46] proposes an implementation of the selective algorithm which has two different approaches. The former is called *selective search fast* and as the name suggests, its objective is to be fast. Unfortunately, this characteristic decrease drastically the recall a.k.a. sensitivity of the algorithm. The latter, called *selective search quality*, contrary to the first one keeps a high sensitivity but with a slower execution time. Because this project does not have any quality restrictions in sense of time, the following implementation uses the selective search with the best results.

Figure 7.3 shows the Figure 7.1a with different numbers of bounding boxes generated with the selective search. The objective of this image is to illustrate that with images like Figure 7.1a and 7.1b 100 bounding boxes are enough to capture all the essential data for a classification. For this reason, the representation with 834 bounding boxes (maximum for the image) has too much redundant information. In fact, it has green zones in all locations which in most of the cases are overlapping rectangles. In the classification process theoretically, this is not a problem because it would simply increase the number of true positives. The real problem is in the difference of time needed to compare 100 or 834 patches. In fact, the comparison of 100 bounding boxes requires 33s.417ms while the 834 comparisons require 4m:35s.110ms.

(a) 10 (b) 100 (c) 834

Figure 7.3: Selective search with a variable number of bounding boxes

Figure 7.4 exposes all the 100 bounding boxes, confirming that this number of images' comparisons is enough. In fact, this figure contains perfect cuts of the researched logos which are exactly as the images used for the classifier training (see Chapter 6).

Figure 7.4: Selective search with 100 bounding boxes results

Because there are so good representations of the searched logos, the results of the classification are really good. In fact, as it is shown in Listing 7, the logos contained in the processed image have a high value (above 70%).

```
INFO: DHL is present at : 10.9 %
INFO: FedEx is present at : 6.7 %
INFO: Mastercard is present at : 71.7 %
INFO: Paypal is present at : 92.7 %
INFO: UPS is present at : 5.6 %
INFO: Visa is present at : 88.0 %
```

Listing 7.1: Figure 7.1a classification results

In the opposite case, when in the processed image there are not the searched logos (see Figure 7.5a), the different values stay under 70%. In this prototype, 70% represents the threshold of when a logo is contained in an image or not. Fortunately, this classifier works well also with images which contain not easily identifiable logos like Figure 7.5b. In this case, the classifier detects really well that the image contains only MasterCard's and Visa's logos because their values are above 70% (see Listing 7).

```
INFO: DHL is present at : 47.7 %
INFO: FedEx is present at : 8.3 %
INFO: Mastercard is present at : 90.5 %
INFO: Paypal is present at : 31.6 %
INFO: UPS is present at : 23.5 %
INFO: Visa is present at : 83.7 %
```

Listing 7.2: Figure 7.5b classification results

(a) Image with no searched logos

(b) Image with difficult logos

Figure 7.5: Selective search test images

Problems arise when normal images (objects or environments) such us Figure 7.6a and 7.6b are processed by the classifier. In this case, the results are really bad. In fact, with both images, the algorithm finds that the researched logos are present with a probability above 70%. To be more precise, Figure 7.6a has a probability of 78% to contain the DHL logo while Figure 7.6b has a probability of 85% to contain DHL and 81% to contain MasterCard. These results are a really big problem because websites could easily contain images like that. Therefore, these websites could generate a lot of false positives which will decrease drastically the Positive Predictive Value. Because PPV is the most important measure to evaluate the classifier in this project (see Section 4.3), a further investigation to understand the problem is needed.

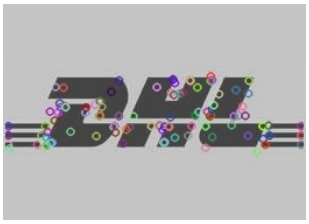
(a) Duomo of Milan (source [79])

(b) Tree (source [80])

Figure 7.6: Normal images (objects or environments)

After a detailed research, it was noticed that the pieces of an image which generates high probabilities had often a larger number of SIFT descriptors/visual words (see Chapter 5). In the specific case of Figure 7.7a, which gave a probability of 70% to contain a DHL logo, it can be seen that the number of descriptors is really high compared to a DHL logo (1200 >> 130).

(a) 1200 SIFT descriptors



(b) 130 SIFT descriptors

Figure 7.7: High probability image vs normal DHL logo

The number of SIFT descriptors is exactly the cause of the high number of false positives. To be more precise, Figure 7.8 shows the average number of SIFT descriptors of the logos used for the training phase in Chapter 4. This average is 140 descriptors and it is much lower than the 1200 descriptors of Figure 7.7a.

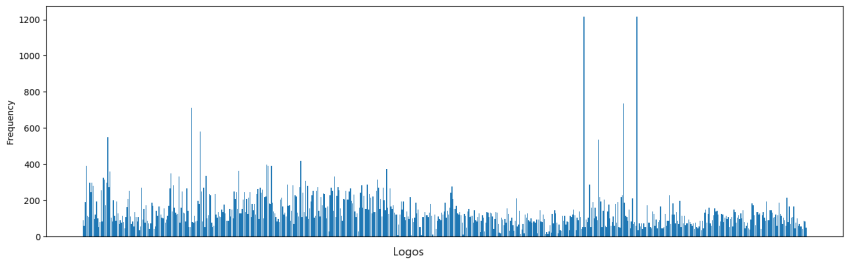
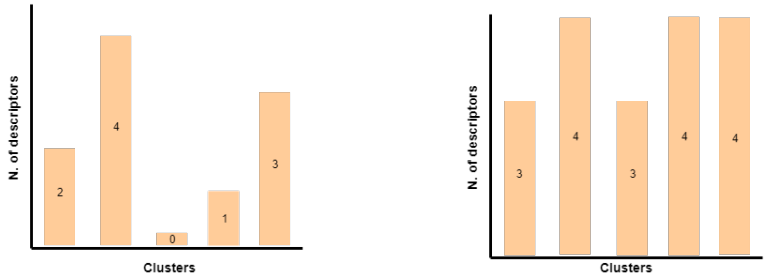


Figure 7.8: Number of SIFT descriptors for each logo in the test directory

The high number of SIFT descriptors in Figure 7.7a causes problems because the BoW model generates a more scattered histogram than the average. The explanation of this situation with a histogram of 10000 elements (see K-Means in Chapter 5 for more information) is extremely difficult, therefore an example with only five elements is used. Figure 7.9a represents a normal logo (e.g. DHL) where the components are quite good separated (10 SIFT descriptors). Figure 7.9b, on the other hand, represents the components of an image similar to Figure 7.7a (18 SIFT descriptors). In this second histogram there are more descriptors and therefore, the classifier could see it as similar to a type of logo. This does not mean that the processed image contains a searched logo, but only that its histogram is similar to a logo's histogram.



(a) *Normal* histogram (b) *Scattered* histogram

Figure 7.9: The SIFT descriptors problem

In this example with five clusters the problem is not so clear to visualize, but with histograms of 10000 elements, it becomes more visible. A logo (e.g.

DHL) generates in average 140 descriptors which are spread across 10000 elements. These values create a very different histogram for each logo's type. In the other hand, where there are 1200 descriptors, the probability to generate a similar histogram to a specific logo is high. For these different reasons, the problem can be solved by removing all the pieces of an image which have more than 200 SIFT descriptors. In fact, Listing 7 shows that this new version has better results.

```
1 INFO: DHL is present at : 34.8 %
2 INFO: FedEx is present at : 8.2 %
3 INFO: Mastercard is present at : 54.3 %
4 INFO: Paypal is present at : 29.7 %
5 INFO: UPS is present at : 17.5 %
6 INFO: Visa is present at : 16.1 %
```

Listing 7.3: Figure 7.6a classification results

To have a more precise idea of the effectiveness of this algorithm, the system has been tested with 45 images where 27 of them are taken from [10] and do not contain any logo. The results of this test are visible in Table 7.1 where the classification is evaluated with the quality measures seen in Chapter 4. The time required to process these images was around 45 minutes.

Images	**Threshold**	**TPR**	**TNR**	**PPV**	**NPV**	**ACC**
All	70%	66.0	98.6	92.1	92.2	92.2
Only logos	50%	88.7	96.7	95.9	90.8	93.0

Table 7.1: Object recognition evaluation

Unfortunately, the results are not as good as expected. The precision is quite good while the sensitivity is too low. This result is bad because in this case the threshold which decides if an image is a logo is high ($prob > 70\%$). It is set so high because some time images that do not contain any logo, generate probabilities which are higher than 50%. Therefore, to keep the precision at a high value the threshold has been increased. As it can be seen from the second line in Table 7.1, if the application is tested with images that contain only logos and a threshold of 50%, the quality measures are much higher. Even if the precision is enough high in both cases, the sensitivity does not achieve a sufficient value in the first approach. For this reason, a solution to increase the TPR must be found and integrated into the existing prototype. The possible solutions within the mandatory requirement (no ANNs) are a lot. They could be i.e. a simple training optimization with more images, a more complicated training optimization with *hard negatives* [81], an optimization of the K-Means in the BoW model, Histogram of Oriented Gradients [23], Haar Cascade [24], etc.

7.3 Algorithm enhancements

Section 7.2 shows that the objects recognition or, to be more precise the selective search, achieves a sufficient result. Unfortunately, in a real application a sufficient result is not enough and therefore it must be improved. For this reason, this section studies all the possible improvements and their results so that the effective usefulness of this prototype can be assessed.

7.3.1 Training set

The first and easiest improvement to apply to this system is the extension of the training set with additional images. The new figures should not contain logos so that the classifier has an idea of other existing objects. The images added ($\approx$ 400) are taken from [10] and contain a multitude of different objects as shown in Figure 7.10.

Figure 7.10: [10] images example

The BoW model generation is already several hours long and with these additional images the duration increases even more. Sadly, even if additional images could be added, an additional increment is not possible because tests of long duration generate delays in the project realisation.

Type	TPR	TNR	PPV	NPV	ACC
Logos as single images	61.2	100.0	100.0	95.0	95.4
Logos in larger images (0.7)	62.3	100	100	91.7	92.6
Logos in larger images (0.5)	77.4	100	100	94.8	95.6

Table 7.2: Selective search training set improvement

Table 7.2 shows the results of the improvements in the classification of single logos and logos in larger images. These tests have been executed after a training phase with 1350 images (before the images were 640). In this case,

the data generation for the BoW model took more time, meanwhile, the classification process took almost the same amount of time. Fortunately, as it can be seen from the third line of the table, the classifier achieved a higher result. With this enhancement, the sensitivity increased by 11.4% and the precision by 4.1% (see differences between Table 7.1 and 7.2). This is not an enormous increment but it is, in any case, an improvement that enhances the algorithm.

7.3.2 Hard negatives

The second and faster improvement for this system is called hard-negative mining. According to [81], when a model for objects detection is trained, there are a lot of negative examples (up to 10^5). Because it is not possible to consider all the negative examples, the concept of *hard negatives* was invented. Hard negatives are negative examples which are incorrectly classified by the initial classifier (false positives). Figure 7.11b shows all the false positives of Figure 7.11a. In this case, it can be seen that there are a lot of wrong identifications and therefore hard negatives that could be used to improve the classifier.

(a) Original image (b) Hard negatives

Figure 7.11: Image with hard negatives

Therefore, to improve the selective search model and the classifier, the hard-negative mining process must be applied. In this case, the hard negatives are all the bounding boxes identified as false positives. To decide whatever a prediction is a false positive, the system computes the Intersection over Union (IoU) between the true positives and the predicted bounding box. True positives are zones localized by hand which are stored in a plain text file as bounding box coordinates (x,y,w,h). IoU is an evaluation metric used to measure the accuracy of an object detector on a particular dataset (for more information see [11]). In other words, it computes a special percentage of overlapping of two windows and according to [11], an IoU above 50% means a good prediction (see Figure 7.12).

Figure 7.12: IoU example (source [11])

For this implementation first, the system searches for all the positives zones. Then, it checks if they have a superposition with the searched logo. If it the case, the zone is considered as true positive and no more actions are performed. On the other hand, if the tested bounding box has not a superposition with a true positive, it is marked as wrongly identified. Once all the false positives are found, the system trains the classifier by saying that all the wrong identifications should be classified as another class.

To test this improvement, the classifier has been tested only with multiple logos in a single image. The required time to update the classifier was around 1 hour.

IoU	Threshold	TPR	TNR	PPV	NPV	ACC
0.5	0.7	24.5	100.0	100.0	84.4	5.2
0.5	0.5	50.9	99.5	96.4	89.3	90.0
0	0.7	64.2	100.0	100.0	91.9	93.0
0	0.5	71.7	99.5	97.4	93.5	94.1

Table 7.3: Selective search hard negatives test

Unfortunately, as it can be seen from Table 7.3, the hard negatives' enhancement with IoU = 0.5 decreased the results rather than increase them. This decrease happened with both thresholds (see Section 7.2.1) and therefore this improvement with these values can be discarded. Fortunately, if the IoU value is reduced to 0 (there is at least an intersection), the results increase a little. In this case, the best results are achieved with the threshold of 0.5, where the sensitivity has been incremented by 5.7% and the precision by 5.3%. As it has been already said in the Section 7.3.1, this is not a big improvement but it is, in any case, something that can enhance the algorithm.

7.3.3 K-Means improvements

Chapter 5.2.3 explained the BoW model implementation which contains a K-Means clustering. As it has been said, the clustering has been executed

ten times with k random centres and for each run, the algorithm has stopped after ten iterations or if every cluster centre has moved less than 1 unit in the last iteration. These settings are given as default by the OpenCV [46] implementation and therefore, they could not be the best ones. For this reason, a possible enhancement could be to vary these values and see if the classification results get better.

Trees	Iterations	Restarts	TPR	TNR	PPV	NPV	ACC
500	50	50	73.1	99.7	98.2	95.0	95.3
1000	50	50	73.5	99.7	98.2	95.1	95.4

Table 7.4: K-Means settings tests in single logos' recognition

To test out the different settings of the K-Means implementation of OpenCV [46], all its components have been varied. In addition to that, the size of the vocabulary of the BoW model has also been modified. To be more precise, the size represents the number of SIFT descriptors used to build the dictionary. It has been varied between $10000, 20000, 30000, 50000$ and 100000 to see if different values could improve the results. The number of the K-Means iterations and restarts has been varied between $10, 20$ and 50. Unfortunately, the changes in the vocabulary size and in the other parameters did not improve the results. In fact, as it can be seen in Table 7.4, the random forest with 500 and 1000 trees had lower results in recognizing single logos than the normal implementation (see Section 6.2). The only thing that has increased is the time needed to generate the BoW model because from the 4 hours it becomes 18 hours. This model has not been tested with the recognition of multiple logos in an image because its results were already low with single logos and because otherwise, the entire Selective Search prototype should have been modified to work with these new K-Means parameters.

7.3.4 Others approaches

This section discusses other possible improvements to increase the results of the developed classifier. These enhancements do not have their own section because they are not implemented or very easy to understand.

Haar Cascade

Object Detection using Haar feature-based cascade classifiers is an effective object detection method proposed by Paul Viola and Michael Jones in their paper [24] in 2001. [82] The implementation of this model would mean the restart of the project and unfortunately, the time available for this thesis does not allow a big change like that. For this reason, this new algorithm could be developed and tested in a future project.

Histogram of Oriented Gradients

Histogram of Oriented Gradients (HOG) descriptors are used in computer vision to perform object detection. HOG is similar to SIFT but it topically used in a sliding window model for object detection. [83] In the case of HOG, an image of size width (i.e. 64), height (i.e. 128) and 3 channels (colours) will have a feature vector of length 64 = 3780.

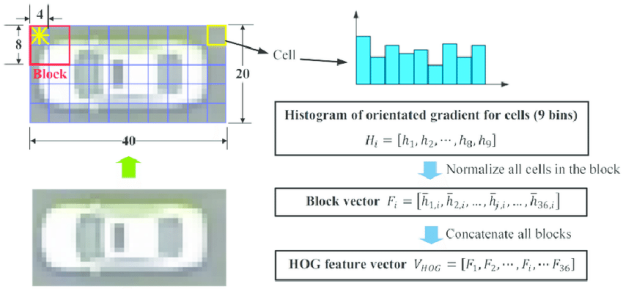


Figure 7.13: HOG work flow (source [12])

Histogram of Oriented Gradients could be a substitute for SIFT and it could be maybe incorporated in a BoW model. Unfortunately, for time constraints in this project is not possible to reimplement the entire system to see if it would give better results. However, it could be a good idea for a future project because this technology could be easily compared with the implementations explained in this document. For more information please refer to [84].

Bounding boxes increment

Section 7.2.1 stated that the number of bounding boxes to detect all the needed logos was 100. Figure 7.4 confirms this statement because it contains all the needed information for the classifier. Because no big improvements to the algorithm have been made, every single idea was tested to see if it would give better results. Fortunately, even if the number of bounding boxes did not seem a problem, it revealed to be the cause of the low results of this classifier.

N. boxes	TPR	TNR	PPV	NPV	ACC
100	66.0	98.6	92.1	92.2	92.2
200	77.4	100	100	94.8	95.6
500	84.9	99.1	95.7	96.4	96.3
1000	86.8	98.6	93.9	96.8	96.3
max	86.8	97.7	90.2	96.8	95.6

Table 7.5: Bounding boxes increment

As it can be seen from Table 7.5, this simple improvement had the highest results until now. In fact, already with 100 additional bounding boxes, the sensitivity has been increased by 11.4% and the precision by 7.9%. The results continue to increase with a higher number of bounding boxes until 500 where the results reach the maximum score of 94.68 (see Section 6.2 for more information about the score). The only problem of this improvement is the time needed to compare such a large number of bounding boxes. In fact, the comparison of 100 bounding boxes for 45 images took around 50 minutes whereas, the comparison of 500 bounding boxes took around 2 hours and 15 minutes. Because this project does not have any quality requirement in sense of time, this problem can be neglected.

7.4 Final outcome

In the previous sections, different improvements have been applied to the classifier to try to achieve better results. These enhancements have been tested singularly and their efficiency has been assessed. Now the only possibility to further increase the results is the combination of these different modifications. Table 7.6 shows these combinations and their scores (see Section 6.2 for more information about the score). To be more precise, the first four rows report the results of the basic improvement whereas, the remaining are the different combinations. For space reasons different abbreviations have been used in fact, in this table BB stays for bounding boxes increase, TS stays for training set improvement, HN stays for hard negatives and Th stays for threshold (see Section 7.2.1).

BB	TS	HN	Th	TPR	TNR	PPV	NPV	ACC	Score
N	N	N	0.7	66.0	89.6	92.1	92.2	92.2	88.86
Y	N	N	0.7	84.9	99.1	95.7	96.4	96.3	94.68
N	Y	N	0.5	77.4	100	100	94.8	95.6	94.63
N	N	Y	0.5	71.7	99.5	97.4	93.5	94.1	92.26
Y	Y	N	0.5	88.7	97.7	90.4	97.2	95.9	93.38
Y	Y	N	0.7	77.4	100.0	100.0	94.8	95.6	94.63
N	Y	Y	0.5	73.6	100	100	93.9	94.8	93.71
N	Y	Y	0.7	62.3	100	100	91.6	92.6	91.08
Y	N	Y	0.5	84.9	97.2	88.2	96.3	94.8	91.6
Y	N	Y	0.7	79.2	100	100	95.2	95.9	95.05
Y	Y	Y	0.5	84.9	97.7	90.7	96.4	95.2	92.36
Y	Y	Y	0.7	75.5	100.0	100.0	94.3	95.2	94.16

Table 7.6: Bag of Words (BoW) final results

Fortunately, the different combinations achieved slightly better results. In fact, the best implementation is the combination of 500 bounding boxes and hard negatives (highlighted row). Therefore, these enhancements will be applied to the final application by default. The different enhancements could be further investigated to improve the results but unfortunately, as stated in the project management document, the objective of this project is to develop an application with almost any improvements because of the short time available for the realisation.

8. Software aspects

In the previous chapters, all the prototypes and their results have been explained in detail. Now, an application which contains all the functionalities must be developed. For this reason, this chapter explains the design of this application, its components and user interfaces.

The software developed must be as modular as possible so that new components (i.e. classifiers, feature algorithms, etc.) can be added without problems.

8.1 Software analysis

This section contains information about the analysis of the software such as use cases and use case diagram.

8.1.1 Use case specifications

This section lists the different use case specifications. These specifications are based on the template shown in [85, p. 68].

View webshop's probability

Section	**Content**
Name	View webshop's probability
Authors	Noli Manzoni
Source	Prof. Dr. Olivier Biberstein (Advisor)
Responsible	Noli Manzoni
Description	The user inserts an URL and the system prints the probability that it is a webshop
Trigger event	The user wants to know the website activity
Actors	User
Pre-condition	An internet connection is available
Post-condition	The user knows the probability that the given website is a webshop
Result	The system prints a probability

Main scenario	1. The user enters a website URL. 2. The system checks if the URL is valid. 3. The system checks if the URL exist. 4. The system downloads the information (images) and analyses it. 5. The system returns the probability that the website is a webshop.
Exception scenarios	2 The system determines that the URL is not valid. A The system displays an error message. B The system waits for a new URL. 3 The system determines that the URL does not exist. A The system displays an error message. B The system waits for a new URL. 4a The system encounters problems with the connection. A The system displays an error message. B The system waits for the connection. 4b The system encounters problems with the analysis. A The system displays an error message. B The system waits for a new URL.

View webshop's services

Section	**Content**
Name	View webshop's services
Authors	Noli Manzoni
Source	Prof. Dr. Olivier Biberstein (Advisor)
Responsible	Noli Manzoni
Description	The user inserts an URL and the system prints the probability that the given website offers the services contained in the system
Trigger event	The user wants to know if the website provides a specific service
Actors	User
Pre-condition	An internet connection is available
Post-condition	The user knows if the given website provides the chosen service
Result	The system prints a list of probabilities

Main scenario	1. The user enters a website URL. 2. The system checks if the URL is valid. 3. The system checks if the URL exist. 4. The system downloads the information (images) and analyses it. 5. The system returns the probability of the different services provided by the website.
Exception scenarios	2 The system determines that the URL is not valid. A The system displays an error message. B The system waits for new URL. 3 The system determines that the URL does not exist. A The system displays an error message. B The system waits for new URL. 4a The system encounters problems with the connection. A The system displays an error message. B The system waits for the connection. 4b The system encounters problems with the analysis. A The system displays an error message. B The system waits for new URL.

Manage Feature or Classifier (CRUD)

Section	**Content**
Name	Manage Feature or Classifier (CRUD)
Authors	Noli Manzoni
Source	Prof. Dr. Olivier Biberstein (Advisor)
Responsible	Noli Manzoni
Description	The user manages a classifier or feature by adding, removing, viewing, setting and training variations (implementations with different parameters)
Trigger event	The user wants to manage a classifier or feature
Actors	User
Pre-condition	At least a classifier's or feature's implementation is available
Post-condition	A classifier or feature variation has been added, removed or modified

Result	The system confirms that the operation has succeeded
Main scenario	1. The user adds, deletes, trains or modifies a classifier's or feature's variation. 2. The system checks if the classifier's or feature's implementation exist. 3. The system checks if the given parameters (variation) respect the implementation. 4. The system performs the command.
Exception scenarios	2 The system determines that the classifier or feature implementation does not exist. A The system displays an error message. B The system waits for a new classifier or feature. 3 The system determines that the given parameters do not respect the implementation. A The system displays an error message. B The system waits for new parameters. 4 The system encounters problems with the command. A The system displays an error message. B The system waits for a new command.

Manage Logos or Logo's types (CRUD)

Section	**Content**
Name	Manage Logos or Logo's types (CRUD)
Authors	Noli Manzoni
Source	Prof. Dr. Olivier Biberstein (Advisor)
Responsible	Noli Manzoni
Description	The user adds, removes, views a logo or logo's type
Trigger event	The user wants to manage logos or logo's types
Actors	User
Pre-condition	At least an image manager's implementation is available
Post-condition	A logo or logo's type has been added, shown or removed
Result	The system confirms that the operation has succeeded

Main scenario	1. The user adds, deletes or shows a logo or logo's type. 2. The system performs the command.
Exception scenarios	2 The system encounters problems with the command. A The system displays an error message. B The system waits for new command.

8.1.2 Use case diagram

The project management document (see attachments) contains all the user stories which have been used to develop the Use Case Diagram (see Figure 8.1).

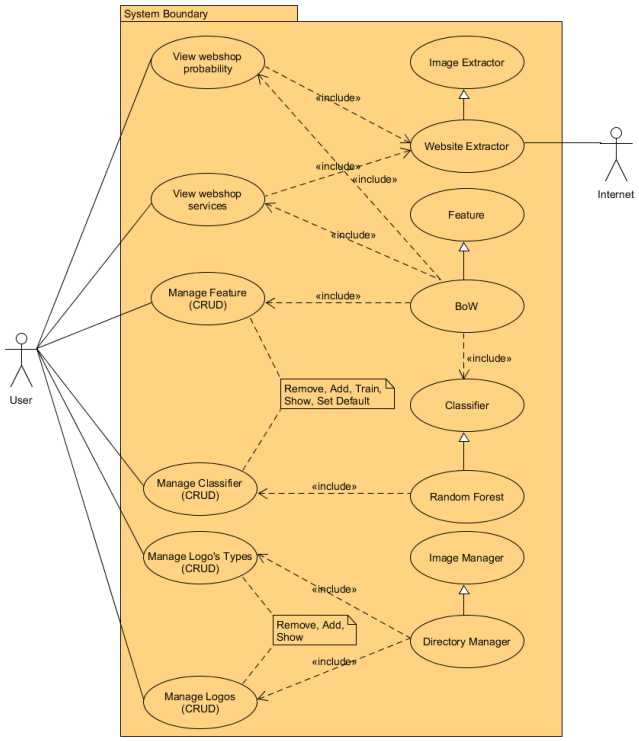


Figure 8.1: Use Case Diagram

This diagram contains all the action which the user can perform (i.e. view webshop's probability, train feature, train classifier, etc.). Moreover, this diagram shows how all these user's actions are subdivided into four different use cases which inherit from four more generic roles i.e. Image Extractor, Feature, Classifier and Image Manager.

Among the user's actions, there is also the possibility to train a feature implementation. This is not a usual thing because the majority of feature implementations do not need a training process. This action is useful for a feature such as Bag of Words because it needs to create a vocabulary of visual words as explained in Chapter 5. Moreover, for the classifier's and feature's related actions, there is the possibility to add different variations of the same algorithm (different settings) and to set the default one for the information retrieval (i.e. view webshop's probability or view webshop's services).

8.2 Software structure

This section explains the design process of the application's development. All the diagrams not present in this section are visible in Appendix A. This section does not contain a domain model because of the nature of the system. In fact, there are not real word entities apart from the logos. In other words, this application contains a series of algorithms and functionalities which cannot be described in a domain model.

8.2.1 Design class diagram

The design class diagram explains in a better way the structure of the application. In this case, it can be seen how each implementation, which contains all the user's actions, inherits its behaviour from a parent class. In addition, the main class (app) contains an instance of each parent class so that the application behaviour can change without problems by swapping an algorithm implementation. To be more precise, this structure represents the strategy design pattern which, according to [86], define a family of algorithms, encapsulate each one, and make them interchangeable and lets the algorithm vary independently from the clients that use it.

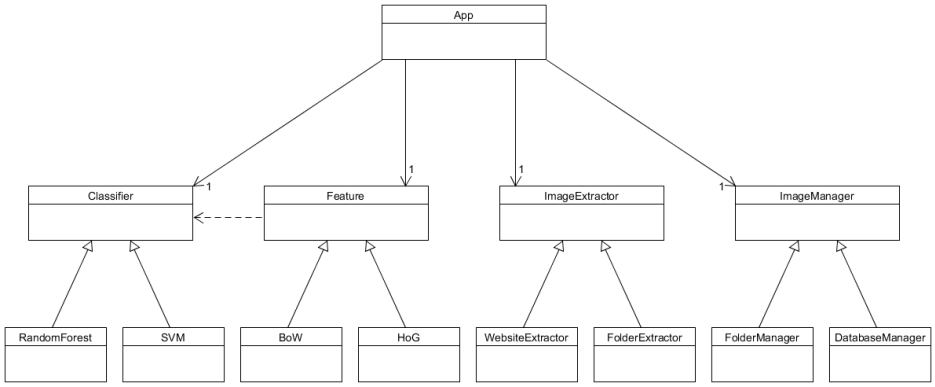


Figure 8.2: Domain Model

This diagram clarifies that the application must be able to support multiple algorithms. Unfortunately, for time constraints and because this project explains only the BoW model and testifies that the Random Forest is the best classifier, only these two implementations are available in the final application. Moreover, for simplicity and because of the great number of images used for the training process, the image manager stores images in a folder (FolderManager). In conclusion, since this project must be able to deal with websites, the image extractor implementation is the WebsiteExtractor which must be able to download images from a given website.

8.2.2 System Sequence Diagrams

The use cases show the events which have been generated by the actor. For this project, there are not important parts in the System Sequence Diagrams (SSDs) because they represent exactly the actions seen in Section 8.1.2. To be more precise, these diagrams show the interaction of the user with the system and in each situation the user has a single interaction with the system. An example of System Sequence Diagram is visible in Figure 8.3 whereas the others can be found in Appendix A.1. This figure shows the action to train the default variation of the classifier by giving the feature name and the classifier name. In this case, the feature's algorithm is necessary because the training process of the classifier requires samples which can be generated only by one of these algorithms.

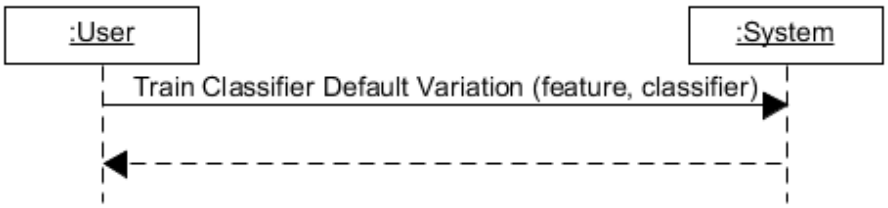


Figure 8.3: Train Classifier Default Variation SSD

8.2.3 Sequence Diagrams

Sequence Diagrams (SDs) show the objects' interaction in a time sequence after the action of the user. Figure 8.4 shows one of the most complicated SD of this project which contains all the important components. The other SDs can be found in Appendix A.2.

This diagram shows the user action to visualize the probabilities of all services offered by the given URLs. In this case, the user gives the websites' URLs, the classifier name, the feature name and different parameter for the images' extractor. Once the User Interface receives the action, it checks

whether the feature and the classifier are present in the system. Then, for each given website, it asks to the main application (*App*) to generate the probabilities of the presence of each service available in the system (logo categories). To be more precise, the *App* object represents an Application Program Interface (API) which allows to use all the functionalities of the system from different user interface implementations (i.e. terminal, Graphical User Interface (GUI), website, etc.). Then, when the application receives the request, it generates the right feature and classifier objects (strategy design pattern). Then, the application asks to the classifier implementation if it has been trained and if the answer is positive, it asks to the image extractor, which checks the website and parameters correctness, to extract the images. Then, the application asks to the feature object to generate the probabilities of the different services. Once this operation is completed, the application asks to the image extractor to delete the downloaded images and then it returns the results to the UI.

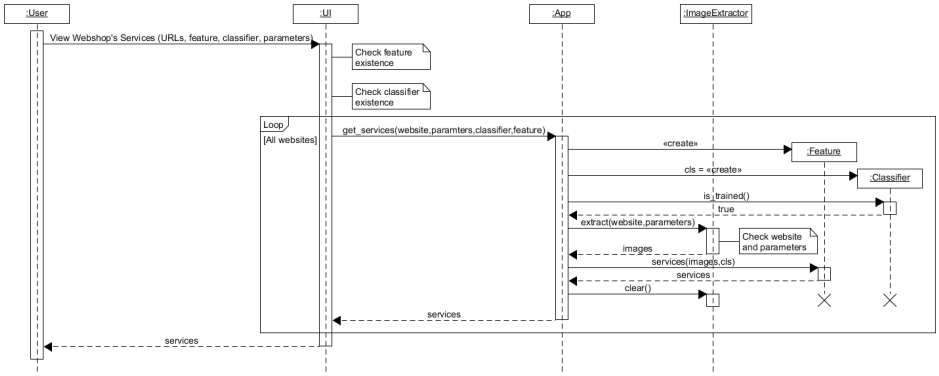


Figure 8.4: View webshop's services SD

8.2.4 Class diagram

With this class diagram (Figure 8.5), the strategy design pattern become clearer because it can be seen how the *app* class uses the interfaces and not the actual classes. Moreover, this shows that the choice of the algorithm is up to the user. Unfortunately, as stated in Section 8.2.1, for this project only one strategy has been developed for each type of algorithm and therefore, the design pattern implementation cannot be fully exploited. This design choice has been made on a view on the future, where a new algorithm could be added to the system to have a performance comparison.

In this diagram, it can be also seen how the *app* class contains a great number of functions which allow interacting with all the components of the system. This centralised control allows using the application from multiples User Interface (UI) without having to rewrite all the necessary code. Because of the presence of different algorithms with different settings, a lot of

methods require a list of parameters. This list is given to the algorithm's implementation and then it is checked to see if it respects the algorithm settings.

In this system, the classifier is used by the feature's implementation because often the classifier training process changes a lot between features. For example, in the case of the BoW model, the classifier training process requires an update called negative mining as seen in Chapter 7.

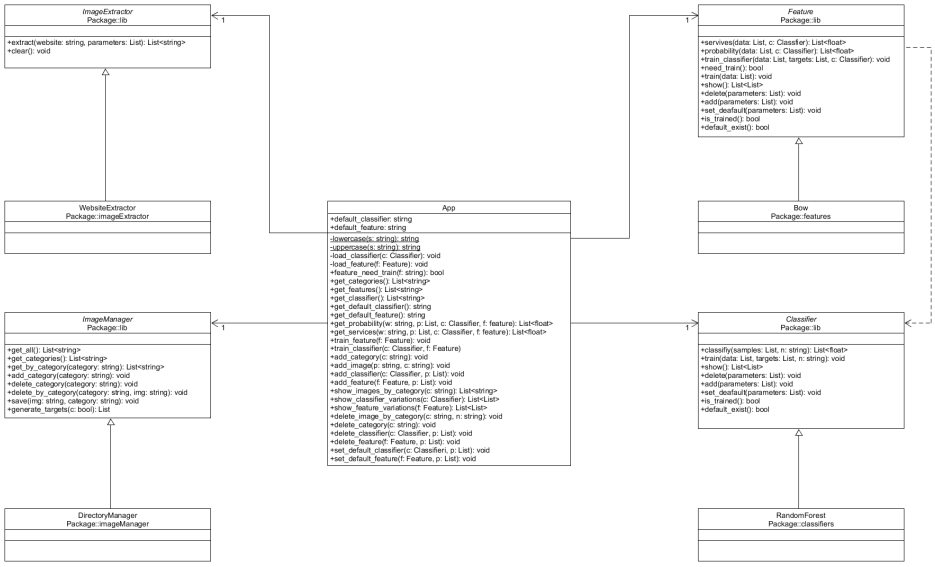


Figure 8.5: Class Diagram

8.2.5 Package diagram

Figure 8.6 shows the arrangement and organization of elements in the system. In this case, all the interfaces which are used by the main application, are stored in the package *lib*. Moreover, different implementations of the same strategy are grouped in a package with the same name of the interface used. This design allows having a clear and tidy structure which has no cycles and which can be extended without a problem.

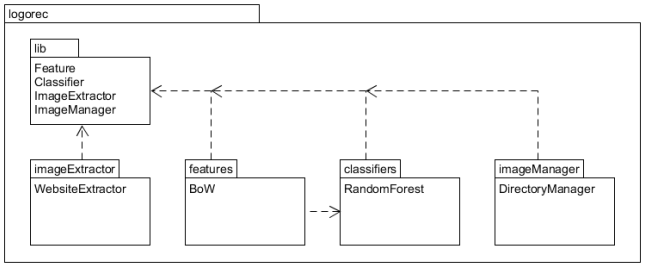


Figure 8.6: Package Diagram

8.3 Implementation

This section explains the main components of the classes developed for this application so that the difficult but important concepts are easily understandable. In addition, the application comes with a good documentation, generated with Sphinx [87], which contains detailed information about the classes methods and arguments. This documentation can be found by following the instructions in the *README* files located in the project directories.

This section describes, in a general way, the four components of the system which have been explained in Section 8.2. In addition, the different strategy's implementations are explained in detail.

8.3.1 Image extractor

The image extractor class is the part of the system which extracts and prepares images for the recognition process. Even if for this project the images should be extracted only from websites, it is important to be able to change strategy without problems. This flexibility allows for example to implement another class which can extract images from a network directory without change the entire system.

How it can be seen from the class diagram in Section 8.2.4 or from Figure 8.7, the image extraction class is really simple. Each implementation must implement two methods i.e. *extract* and *clear*. The former must extract and return a list of local images paths which have been extracted from the given website or i.e. network directory using the given list of parameters. On the other hand, the latter must delete all the images which have been saved locally.

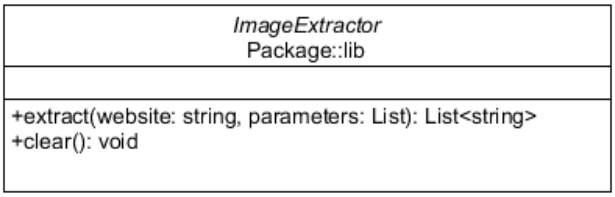


Figure 8.7: Image Extractor

As it has been already stated, for this project a website images' extractor has been developed. This implementation downloads images iteratively by visiting all the internal URLs which point to the same domain. Then, it downloads each image which has not been already downloaded. To limit the number of images extracted, the class uses a parameter called depth. This value tells the system how many pages should be visited. For example with

a depth of one, the implementation downloads only the images contained in the first website page whereas, with a depth of two, the system downloads the images of the first page and of all pages which can be reached by visiting a single URL from it. The images downloaded are stored in a directory so that the method *clear* can easily delete them.

Methods

extract(source, parameters) This method extracts images from the given source (i.e. website) by using the given parameters. It must return a list of local images' paths.

clear() This method deletes all the images downloaded by the *extract* method.

8.3.2 Image manager

Image manager is the class which manages all the images used in the training process. It is important that this part of the system is flexible because there is a larger number of ways to manages and stores images (i.e. database, directories, clouds, etc.).

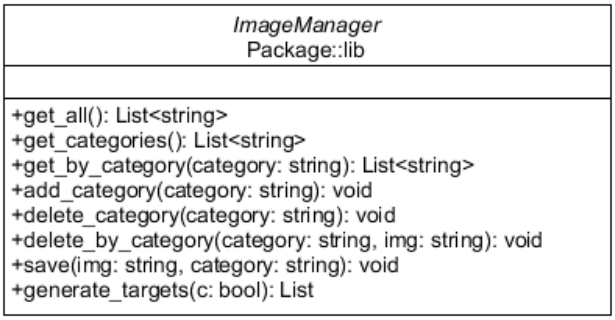


Figure 8.8: Image Manager

How it can be seen from the class diagram in Section 8.2.4 or from Figure 8.8, the image manager contains few more methods than the class seen above. In this case, each implementation must be able to add, retrieve and delete logos' categories (i.e. DHL, PayPal, MasterCard, etc.) and logos. Moreover, it must be able to generate the targets for the training process. The targets are just numbers which represent the category of each logo (i.e. MasterCard = 1, PayPal = 2, etc.). The *generate targets* method has a boolean parameter which says if the targets should be divided in two classes (i.e. MasterCard = 1 and Others = 0) or in a class for each category (see Chapter 6 for more information).

Because of the great number of images needed in the training process (≈ 650), for this project it has been created an image manager which manages the images in directories. This solution allows not to interact with a database which would make the system much more complex. However, the implementation called *directory manager* separates logos' categories in different directories which contains all the relative logos.

Methods

get_all() This method returns a list all images' paths.

get_categories() This method returns a list of all categories' names.

get_by_category(category) This method returns the list of images' names contained in the given category.

add_category(category) This method adds a new category to the system.

delete_category(category) This method deletes the category and all the images related to it.

delete_by_category(category,name) This method deletes the given image from the given category.

save(image,category) This method adds a new image to the given category.

generate_target(mode) This method returns a list of the classes of each logo's type.

8.3.3 Feature

This class represents a feature's implementation such as the Bag of Words model. This is one of the most important parts of the program because it contains most of the software logic. The feature's strategy should be easily interchangeable so that a user can use different techniques with different performance. The most essential characteristic of the feature's implementations is their compatibility with each classifier because otherwise, the system could not compute the different values such as services or probability to be a webshop.

How it can be seen from the class diagram in Section 8.2.4 or from Figure 8.9, it is the feature's and not the classifier's class which computes the final probabilities by using a classifier's instance. This choice is due to the fact that each feature's implementation uses the classifier in a different way and therefore, a standard behaviour cannot be programmed in the classifier. Moreover, the majority of feature's implementations require different parameters which lead to slightly different implementations. For this reason, the system allows the addition of different feature's variations and to set up the default one for the training and information retrieval process. In addition, each feature's implementation contains a method which says if the feature needs a training or not because in some cases (i.e. BoW model) the feature has a preparation phase which can require a lot of time.

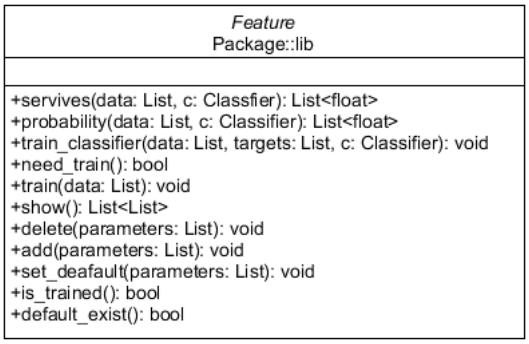


Figure 8.9: Feature

Because this thesis has discussed only a single feature i.e. the Bag of Words (BoW) model, the implementation contains this single feature's strategy. The BoW class is a restructured version of the prototype seen in Chapter 5 and 7 which trains the classifier in the same way of Section 7.4.

Methods

services(data,classifier) This method uses the classifier to compute the probability of the presence of each logo's category in the given data.

probability(data, classifier) This method uses the classifier to compute the probability that the given data represent a webshop.

train_classifier(data, targets, classifier) This method trains the given classifier with the given data and targets.

need_train() This method says if the feature needs a training phase or not.

train(data) This method trains the feature with the given data.

is_trained() This method says if the default variation is trained or not.

show() This method shows all the feature's variations available in the system.

delete(parameters) This method deletes the given feature's variation which is represented by the parameters.

add(parameters) This method adds a new feature's variation with the given parameters.

set_default(parameters) This method sets a feature's variation as default.

default_exist() This method says if a default variation exists.

8.3.4 Classifier

This class represents a classifier's implementation such as SVM, decision tree or random forest. This class contains similar functionalities to the feature's class because it allows adding, deleting and setting classifier's variations, training them and classifying a list of samples. As Section 6.1 has shown, there are numerous classifier's techniques that can classify features. For this reason, it is important to be able to change the classifier's strategy without problems.

How it can be seen from the class diagram in Section 8.2.4 or from Figure 8.10 this class differs from the feature class by a single method i.e. classify. This new method is the centre of the entire system because it classifies the feature's sample so that the probabilities of the presence of the different logos' categories can be returned.

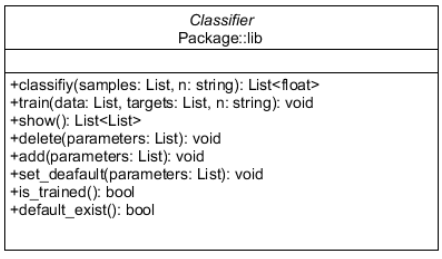


Figure 8.10: Classifier

Section 6.1 has discussed different classification techniques which have been then evaluated in Section 6.2. This evaluation stated that the random forest is the best technology to classify logos which belong to payment and delivery service companies. For this reason and because of limited time, in this project only the random forest has been developed. The implementation is a re-factored version of the prototype developed for the Chapter 6.

Methods

classify(samples, name) This method uses the classifier's variation, which is defined by the name, to classify the feature's samples. It returns a list of probabilities (one for each logo's category).

train(data, targets, name) This method trains the given classifier's variation, which is defined by the name, with the given data and targets.

is_trained() This method says if the default variation is trained or not.

show() This method shows all the classifier's variations available in the system.

delete(parameters) This method deletes the given classifier's variation which is represented by the parameters.

add(parameters) This method adds a new classifier's variation with the given parameters.

set_default(parameters) This method sets a classifier's variation as default.

default_exist() This method says if a default variation exists.

8.4 User Interface (UI)

This section explains the two User Interfaces (UIs) developed to allow the users to use all the functionalities described in the previous section. For this thesis, it is mandatory to develop a command line application and it is optional to develop a Graphical User Interface (GUI). To be able to support different UIs, all the system's functionalities have been implemented in a main class called *app* (see Section 8.2.4). This special class, which can be seen as an Application Program Interface (API), allowed to develop both UIs (optional and mandatory) without being obliged to copy and paste code everywhere.

8.4.1 The command line approach

For the development of the command line application, a Python package has been used. This package allows creating beautiful command line interfaces in a composable way with as little code as necessary. This library is called Click [88] and with few lines, it allows transforming a function in a terminal command. Listing 8.1 shows the implementation of the image manager save method. It can be seen how easy it is to create a command which accepts multiples images and a category among a predefined list of them (type.Choice()). Moreover, the system allows checking on the fly if a given image exists so that the system does not need to deal with file not found errors.

```
@add.command()
@click.argument('category', type=click.Choice(categories_list), nargs=1)
@click.argument('images', required=True, type=click.Path(exists=True), nargs=-1)
def image(category, images):
 """
 Add the given IMAGES to a given CATEGORY set
 """
 for img in images:
  app.add_image(img, category)
```

Listing 8.1: Add image Click example

To execute this example the user should insert a simple command with the relative parameters as shown in Listing 8.2.

```
terminal.py add image DHL C:\images\dhl_01.png C:\images\dhl_02.png
```

Listing 8.2: Add image click example execution

In addition to this simple syntax, this package creates automatically an exhaustive documentation for the user which it is shown when the user asks for help (–help) or when an error occurs (see Figure 8.11).

```
Usage: terminal.py [OPTIONS] COMMAND [ARGS]...

  This application allows to manage logos and logos's categories, to know
  the probability that a website is a webshop and to know the services
  offered by a webshop.

Options:
  --help  Show this message and exit.

Commands:
  add     Add new images, categories, classifiers,...
  delete  Delete images, categories, classifier,...
  get     Retrieve the website's probability to be a...
  set     Change the default classifier' parameters and...
  show    Shows images, classifiers, features, etc.
  train   Train the classifier or the feature model.
```

Figure 8.11: Command line app help instructions

The rest of the system functionalities are implemented in a similar way so that the code is clear and understandable.

8.4.2 A more classic application

Because this program could be used also by inexperienced users which want to know it a certain website offers a certain service, it is important to have a user-friendly interface. For this reason, it has been decided to develop a Graphical User Interface which implements all the functionalities.

To develop this GUI application, it has been decided to use PyQt [89] a Python binding of the cross-platform GUI tool-kit Qt which is an application framework for creating classical and embedded graphical user interfaces.

Thanks to this library, a very simple and easy to use cross-platform (Windows, Linux, macOS) application has been created. This application can be used by inexperienced users because it is really intuitive as shown in Figure 8.12.

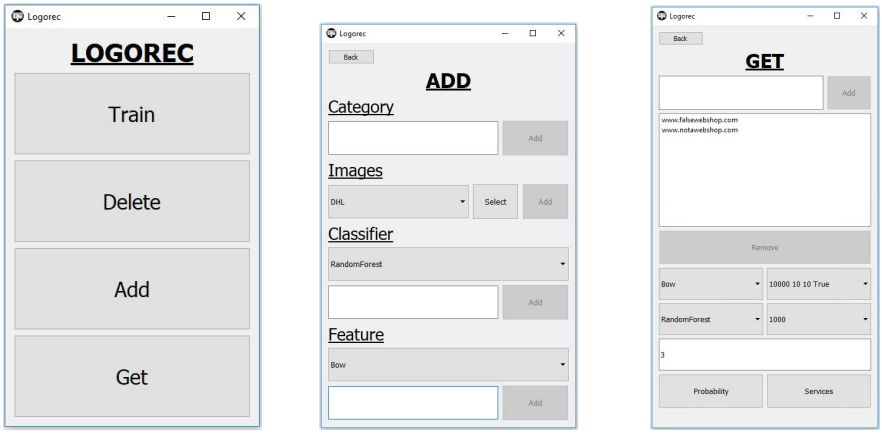


Figure 8.12: GUI application

8.5 Tests

All the system's functionalities have been deeply tested with unittest [90]. The tests have achieved a 91% coverage of the entire application (see Figure 8.13).

Module ↓	statements	missing	excluded	coverage
C:\Dev\git\Logos-Recognition-for-Webshop-Services\software\logorec\logorec\app.py	155	21	0	86%
C:\Dev\git\Logos-Recognition-for-Webshop-Services\software\logorec\logorec\classifiers\randomForest.py	116	9	0	92%
C:\Dev\git\Logos-Recognition-for-Webshop-Services\software\logorec\logorec\features\bow.py	285	17	0	94%
C:\Dev\git\Logos-Recognition-for-Webshop-Services\software\logorec\logorec\imageExtractor\websiteExtractor.py	115	15	0	87%
C:\Dev\git\Logos-Recognition-for-Webshop-Services\software\logorec\logorec\imageManager\directoryManager.py	41	1	0	98%
Total	**1045**	**97**	**0**	**91%**

Figure 8.13: Test coverage

To assure a 100% success rate when using the application, the tests have been integrated into a continuous integration system which checks the integrity of the application each time the changes are committed to GitLab (see Figure 8.14).

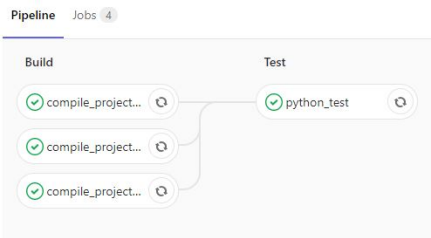


Figure 8.14: GitLab pipeline

In addition to executing the tests, the continuous system compiles the latex's documents so that a history of documents is built up automatically.

8.6 Final results

To test out the final application it has been developed a special image extractor class which allows giving a single image to the system. This new class is important because a single image takes already 5 minutes and therefore, testing a complete website would take hours and the result could not be compared with the processed images. Figure 8.15 shows the excellent results of the application which, given an image with different elements, succeed in recognizing five logos which are available in the system (probability greater than 50%). For other results please refer to Appendix B.

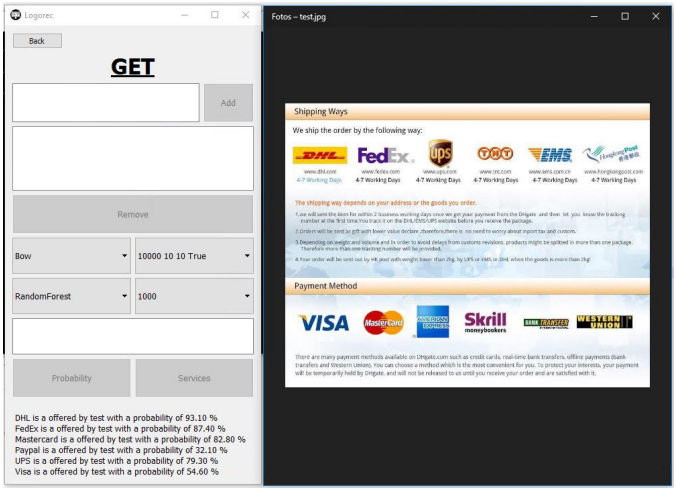


Figure 8.15: Application results

9. Neural Networks

This chapter gives a very high-level overview of the Artificial Neural Networks concept. This small introduction it is a necessity because in Section 6.4 a small implementation which uses this technology has been developed.

9.1 A short introduction

A simple definition of neural networks, that more precisely should be called Artificial Neural Networks (ANNs), is given by the creator of one of the first neurocomputers.

> ...a computing system made up of a number of simple, highly interconnected processing elements, which process information by their dynamic state response to external inputs.
>
> (Dr. Robert Hecht-Nielsen — [91])

Artificial Neural Networks are typically structures organised in different layers. These layers are composed of a series of interconnected nodes which contain an activation function. These layers are divided into three categories (see Figure 9.1) that are: input layer, hidden layer and output layer. The input layer usually represents sensors where the input value is represented by a value between 0.0 and 1.0. The output layer on the other hand, typically returns an answer to a question (yes/no) or a vectorized property. The important part of these ANNs is the hidden layer which is composed of the most complex transformation functions.

These layers, composed of multiple neurons, are a simple feed-forward network where each neuron is connected to each neuron of the previous layer (iterating from left to right). These connections and their activities are represented by parameters called weights. These weights, thanks to the communication between neurons, can change and therefore influence which is the next neuron in the network to be activated. This procedure is managed by the activation function which, in simple words, does nothing but take a value as input and transforms it into another value.

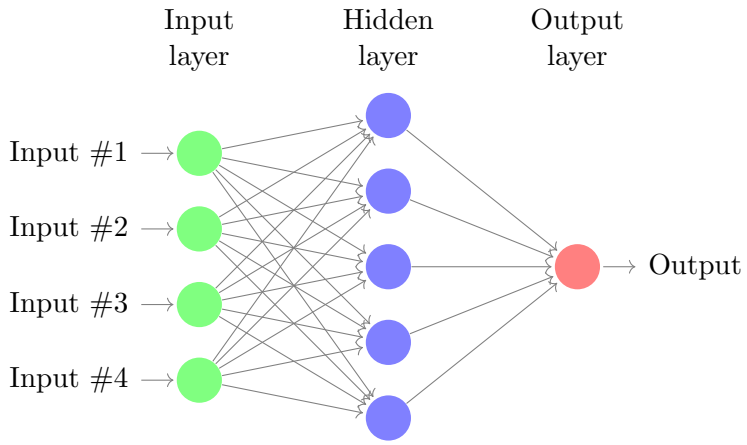


Figure 9.1: Neural Networks Layers (based on [13])

9.1.1 An everyday example

In order to better understand the functioning of these ANNs, let's use a simple example taken from [14], reformulated for this thesis.

Let's think that someone is throwing a party, and he needs to know how much wine to buy. To collect data, he calls three friends who just threw parties, and he asks them how many people they invited and how much wine they bought. With this information, he tries to predict how many wine bottles he needs to buy, but he realises that he cannot choose an appropriate model. This is because he did not get nice linear data, or he could not see any correlation between the size of a party and how much wine was consumed. In fact, to choose an appropriate model, he needs to understand how his inputs relate to the correct answer. To answer this question, he calls back his friends, and he asks them about their parties. It turns out that a lot of factors go into party planning. Of course, some people drink and others do not. But there are other complications. For example, his friend Agatha despise drinkers and therefore if she is invited, everyone will stick to non-alcoholic drinks. On the other hand, his friend Dahlia is the life of the party and so if she is invited, everyone will drink more than usual.

This type of problem, which seems very difficult, can be solved by an Artificial Neural Network without problems because this technology is good at finding relationships between inputs and outputs.

The mathematics behind this process is not so complicated. It can be seen initially as a composition of functions which are called with the output of other functions. A function could be defined in the form of text, as $MATH(x_1, x_2, ..., x_n)$ or, in the form of a diagram, as shown in Figure 9.2.

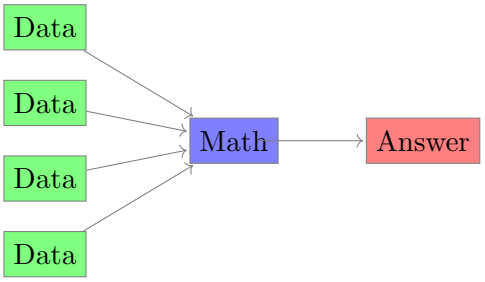


Figure 9.2: ANN with a single function (based on [14])

Now that the definition of a single function is given, an ANN with a composition of functions can be defined. Therefore, an ANN can be defined as $MATH(MATH(x_1, x_2, ..., x_n) + MATH(x_1, x_2, ..., x_n) + ...)$ or in form of a diagram as shown in Figure 9.3.

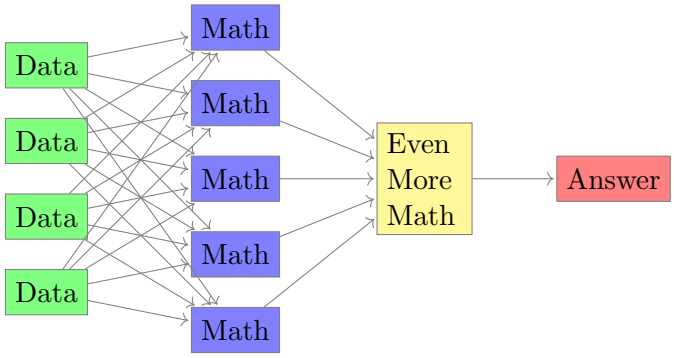


Figure 9.3: ANN with a composition of functions (based on [14])

These maths black boxes in the case of Artificial Neural Networks are called neurons. In this example, neurons in the first layer calculate, directly from the input received, things such as "are people from Friend Group A and Friend Group B present". Then the neurons of the last layer use all the features (each bit of relevant information) to find an answer. As already mentioned, ANNs work very well in this case but why? Everything is due to these neurons because they can extract information about the interactions between inputs, instead of looking for information from one input at a time (as humans do).

Following the example [14] again, let's create an Artificial Neural Network for this party. Suppose there are three people invited: Agatha, Dahlia and Stiffy. As it has been seen, the number of guests does not give enough information so for the input it must be used the entire list of guests. Then, a list is defined as list of numbers where each position represents the presence of a person (0/1) in the following order [Agatha, Dahlia, Stiffy].

Now using this data it must be decided what each neuron of the input layer should calculate. For this example, it can be used a function called Rectified Linear Unit (ReLU) (see Formula 9.1), which is very similar to a linear function.

$$ReLu(n) = \begin{cases} n \; if \; n > 0 \\ 0 \; if \; n \leq 0 \end{cases} \tag{9.1}$$

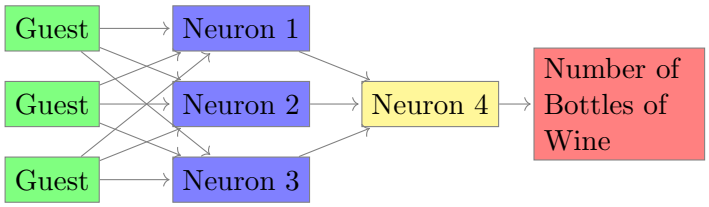


Figure 9.4: ANN for Party Example (based on [14])

As it can be seen from Figure 9.4, this example has two layers of neurons. In order to find the best result through training, this Artificial Neural Network has four parameters, i.e. a, b, c and d, for each neuron. So every neuron is calculated as follows.

$$\text{neuron}_i = ReLu(a_i \times \text{Agatha} + b_i \times \text{Dahlia} + c_i \times \text{Stiffy} + d_i) \tag{9.2}$$

The second layer and its single neuron have, like the others, parameters that must then be adjusted through training. So neuron 4 is calculated as follows.

$$\text{Bottles} = ReLu(w \times \text{neuron}_1 + x \times \text{neuron}_2 + y \times \text{neuron}_3 + z) \tag{9.3}$$

So, from [14], the training process is :

1. Convince friends to give the guest lists for all their parties.
2. Choose random values for every parameter to create an initial model.
3. Run all the guest lists through the model.
4. See whether the model over- or under-estimated how much wine was needed for each guest list.
5. Adjust w, x, y, and z by a little to reduce the error. This means that it must be understood whether each individual neuron tends to overestimate or underestimate.
6. Adjust $a_1, b_1, \ldots, c_3, d_3$ by a little to reduce the error. Re-use the information from step 5 about when individual neurons over- or under-estimate.

7. Repeat steps 3 — 6 until the error is small. In this example, the program automatically runs the model, calculates the error, and adjusts all the parameters hundreds or thousands of times.

Once the ANN has been trained, a possible result could be something like the following Listening.

$$\begin{aligned}\text{Bottles} =& \text{ReLU}(0.5 \times \text{ReLU}(0 \times \text{Agatha} + 0 \times \text{Dahlia} + 1 \times \text{Stiffy}) + \\ & 2 \times \text{ReLU}(0 \times \text{Agatha} + 1.5 \times \text{Dahlia} + 0.4 \times \text{Stiffy} + -0.5) + \\ & -5 \times \text{ReLU}(1 \times \text{Agatha} + 0 \times \text{Dahlia} + 0 \times \text{Stiffy}) + 1)\end{aligned} \tag{9.4}$$

In this model, each neuron encodes something about people's drinking habits:

- Neuron 1 means that, when Stiffy is alone, she will have about half a bottle of wine.
- Neuron 2 means that when Dahlia is around, she will have a couple bottles, and Stiffy will have more than usual.
- Neuron 3 means that when Agatha is present, nobody drinks anything.
- And that +1 at the end, means that if nobody shows up the organizer will have a bottle of wine for himself.

9.2 Convolution Neural Networks

Convolution Neural Networks (CNNs) are in fact artificial neural networks and just like ANNs, in fact, are also made up of neurons connected to each other by means of weighted connections. However, Convolution Neural Networks make the specific assumption that their input has a precise data structure, such as an image in this case. This assumption allows assuming specific properties in the architecture in order to better process such data.

This section is partially based on [15].

9.2.1 Differences between CNN and ANN

Normal ANNs with a fully connected architecture, where each neuron in each layer is connected to all neurons in the previous layer, generally do not scale well as image sizes increase. If one considers, for example, a well-known dataset called CIFAR-10 [92], where 10 indicates the number of categories or classes available, each image has dimensions of 32 x 32 x 3 (width 32, height

32 and 3 colour channels). Therefore, each neuron completely connected to the first hidden layer would have 32*32*3=3,072 weights. This amount would still be manageable, but if the image is 200 x 200 x 3, which is not so large, the same neuron would have 200*200*3=120,000 weight. This is only for a single neuron, but if the whole network is then considered, it will certainly become unmanageable. Unlike normal ANNs, the layers of CNNs have neurons organized into three dimensions: width, height and depth. For example, an input image of CIFAR-10 constitutes an input volume of 32 x 32 x 3. Moreover, in the CNNs, the neurons in a layer are only connected to a small region of the previous layer, rather than to all the neurons as in the ANNs architecture. This is perhaps the main feature that distinguishes a CNN from a normal ANN. For CIFAR-10 the output layer will be 1 x 1 x 10 size because at the end of the network, the starting image will be reduced to a score vector for the 10 classes present in the dataset, organized along the depth size. Figure 9.5 should clarify the above description. In summary, each layer of a CNN transforms a 3D volume input into a 3D volume output where the latter constitutes the set of activations of the neurons of that layer, through a specific activation function.

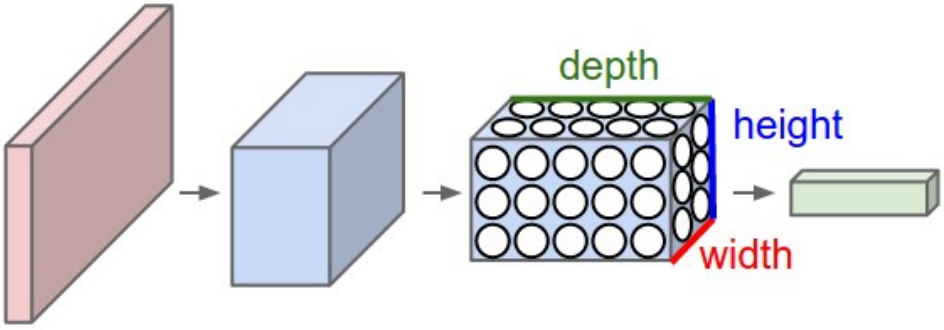


Figure 9.5: CNN representation (source [15])

10. Conclusion and future work

The objective of this thesis was to recognize logos belonging to payment or delivery service companies and to succeed, given a website, in saying whether it is a webshop or not. As it has been shown in this thesis, these objectives have been fully achieved.

As it has been said in the introduction, this thesis is a following of a preliminary work that has found that SIFT descriptors are the best technology for recognizing payment's or delivery's logos. Unfortunately, this technology has a big problem because if it is applied to different images, it returns a different number of descriptors. This fact, which at first glance doesn't seem to be a big problem, makes it difficult to use machine learning techniques which requires data with always the same size. Fortunately, this project has discovered a technique called Bag of Words which allows, given the SIFT descriptors, to create a set of data that has the same size for each image.

This new algorithm, even if it solves the initial problem, does not allow recognizing logos automatically. For this reason, during this project, several classifiers (e.g. SVM, Decision Tree, Random Forest, etc.) were evaluated in order to find the optimal one for this special type of recognition. At the end of this exhaustive comparison, a technology gained first place, reaching a sensitivity of 77 % and a precision of 98%. This new combination, Bag of Words applied with a Decision Tree Classifier, allowed the partial achievement of the thesis' objectives.

Unfortunately, this new model, even if it works well with single logos, does not work completely with images that contain more than one logo. For this reason, an object detection technique has been added to the system. Unfortunately, the Selective Search applied to the previous techniques reached a very low value, i.e. a sensitivity of 66% and a precision of 92%. Fortunately, after some improvements, the combination of these techniques has managed to achieve good results reaching a sensitivity of 80% and a precision of 100%. At this point, the objectives of the thesis were almost completed except for the fact that an application, which contained these techniques and which

could detect whether a website is a webshop, was missing.

In the end, a software was developed. This new program, in addition to allowing the use of the above techniques, has been developed so that new techniques could be added without problems.

This summary allows understanding that the objectives of this thesis have been completely achieved thanks to in-depth research and development of a complete application. Unfortunately, it is not all perfect. In fact, as it has been shown throughout the document, the techniques implemented are very complicated and have a long training and processing time. On the other hand, new techniques such as CNNs, which even if being complicated are easy to use, reached a very good result in the blink of an eye (see Section 6.4). For this reason, in order to have a real rating for the quality of the results obtained here, a similar application implemented with CNNs should be created. In addition to this possibility, there are many other techniques available to try to improve the result obtained in this thesis. Leaving these comments apart, as already said, all the objectives have been fully achieved obtaining results that could be used in a real situation.

In conclusion, I would also like to say that this thesis has allowed me to test different techniques studied during this Bachelor degree and to specialize in some of them. In addition, I had the opportunity to learn a new language (Python) and get to know it in more depth.

11. Acknowledgements

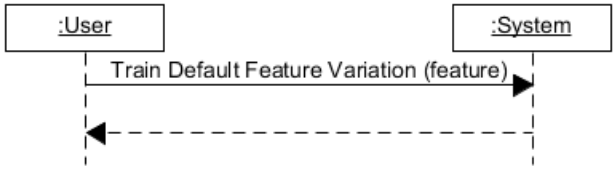
I would like to begin by thanking Prof. Dr. Olivier Biberstein for always supporting, listening and advising me throughout this thesis, dedicating a lot of time and also providing useful tips for improving my work. I also thank my family and my girlfriend for supporting me throughout all these years of study. Moreover, I thank also all the teachers who have supported and helped me improve as a computer science engineer. I thank all my classmates, met during my time at Bern University of Applied Sciences, for all the good moments spent together. Thanks are also due to my friends, for all the evenings that have endured my stories in the course of this work.

A. Diagrams

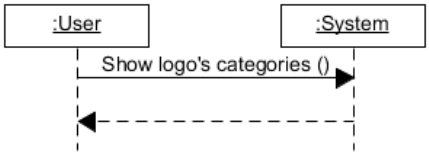
This appendix contains all the diagrams which are not explained in Chapter 8 because they are really similar to each other.

A.1 System Sequence Diagram

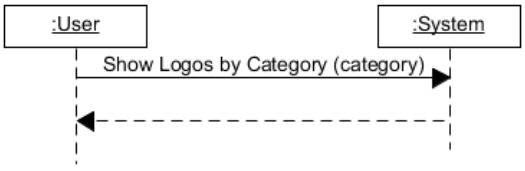
Train default feature variation



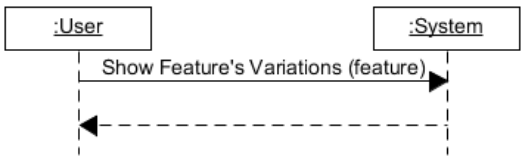
Show logo's types



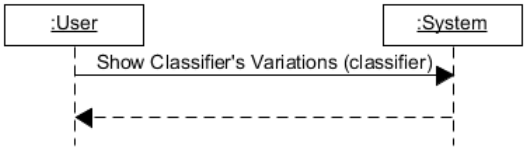
Show logos by category



Show feature's variations



Show classifier's variations

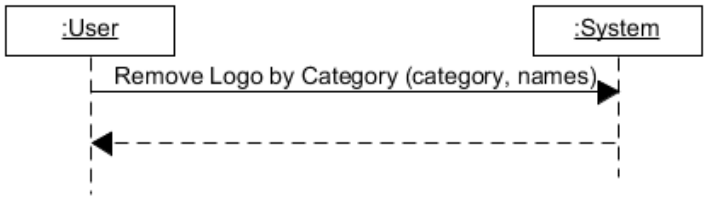


View webshop's services

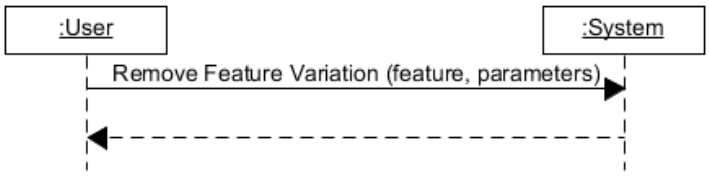
View website's probability

Remove logo category

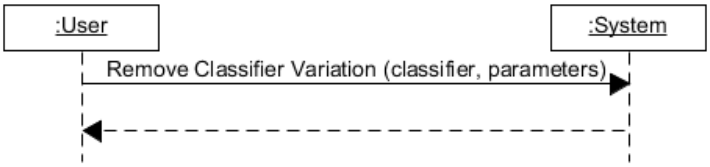
Remove logo by category



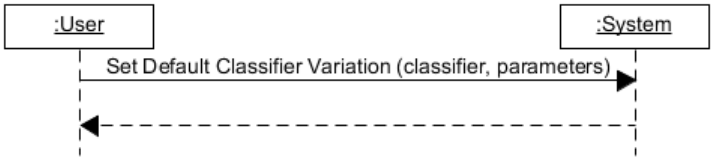
Remove feature variation



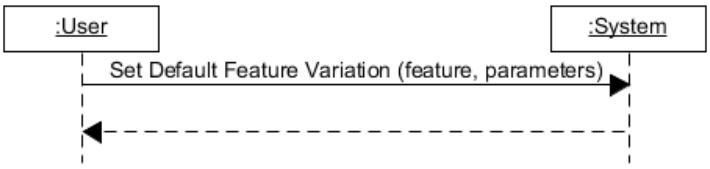
Remove classifier variation



Set default classifier variation

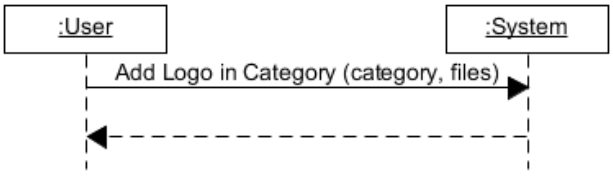


Set default feature variation

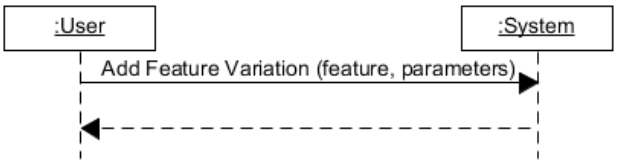


Add logo category

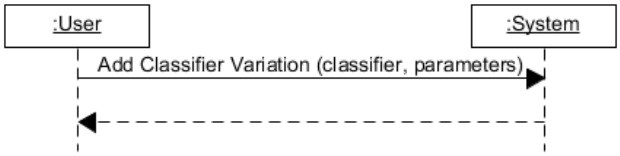
Add logo in category



Add feature variation

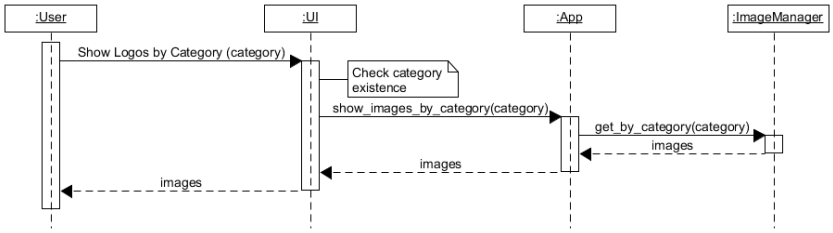


Add classifier variation

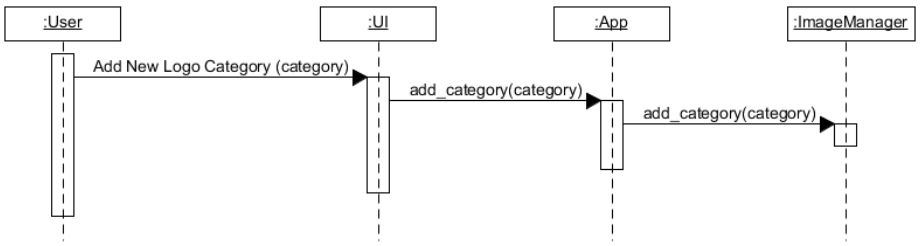


A.2 Sequence Diagram

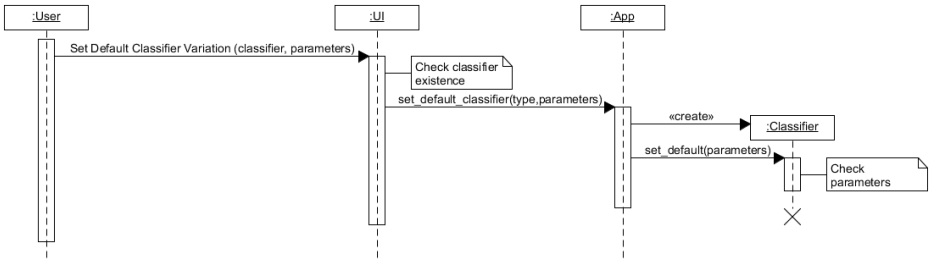
Show logos by category



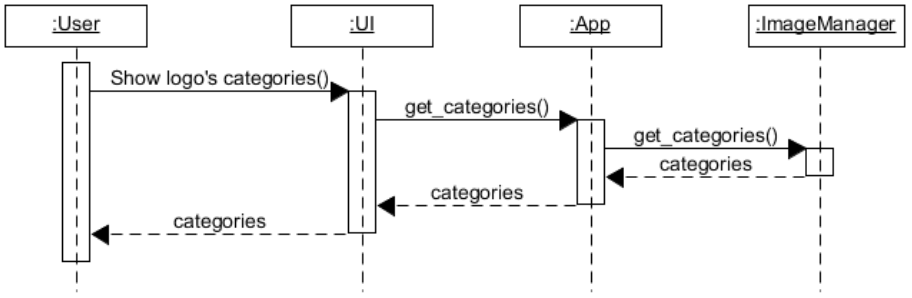
Add logo category



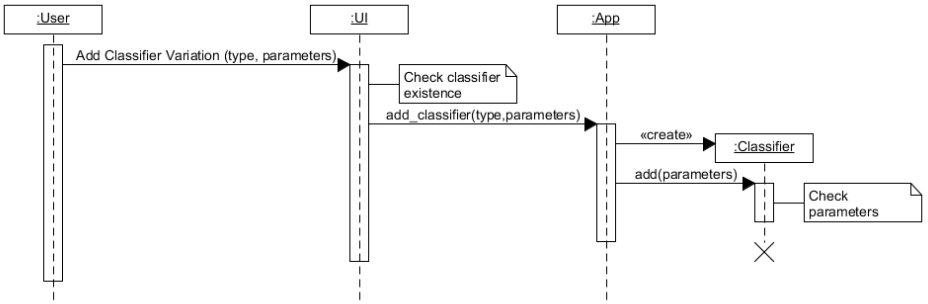
Set default classifier variation



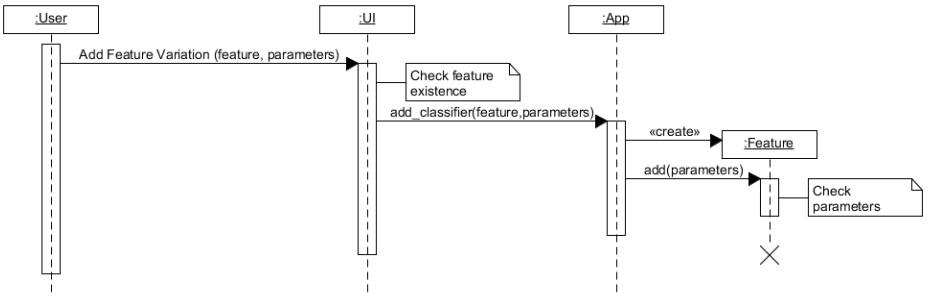
Show logo's categories



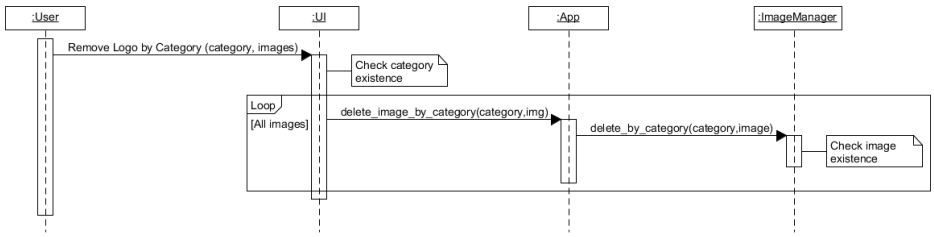
Add classifier variation



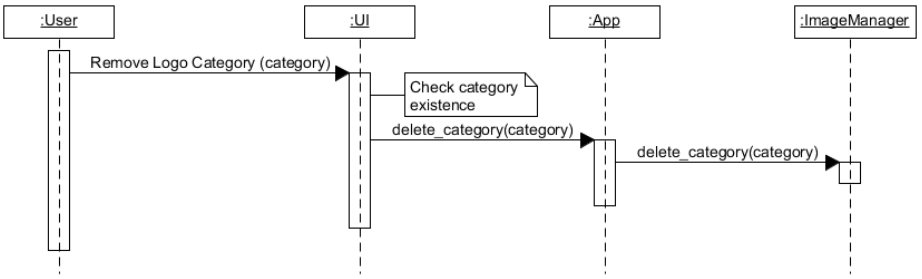
Add feature variation



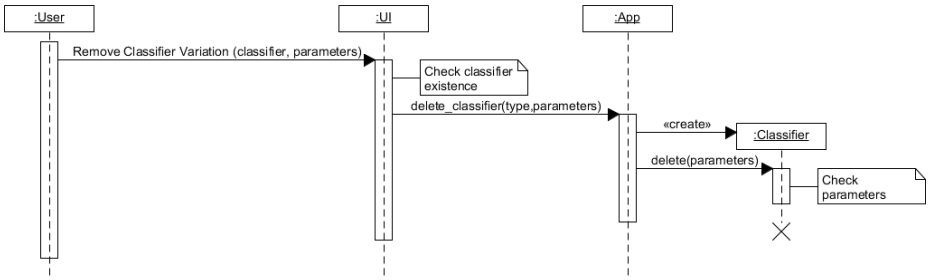
Remove logos by category



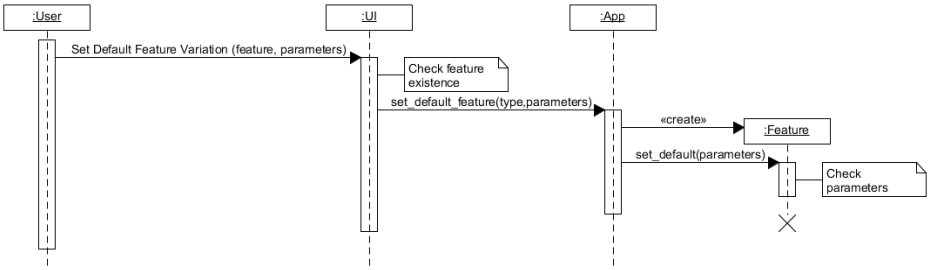
Remove logo category



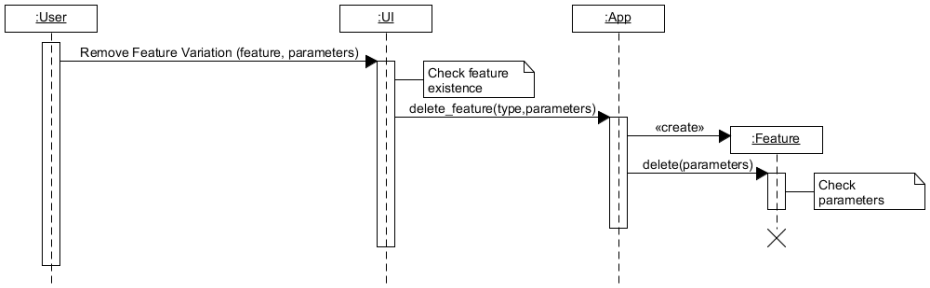
Remove classifier variation



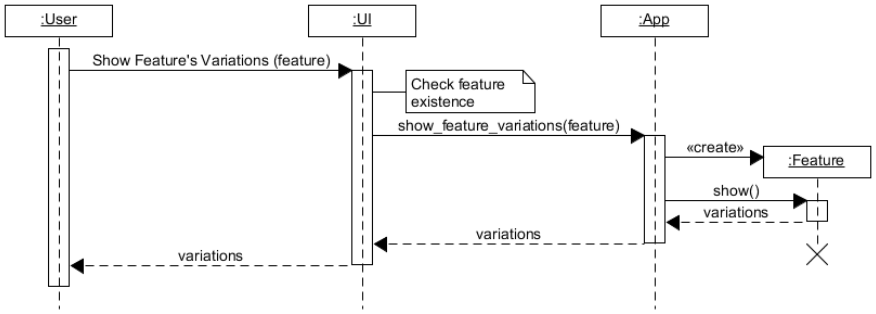
Set default feature variation



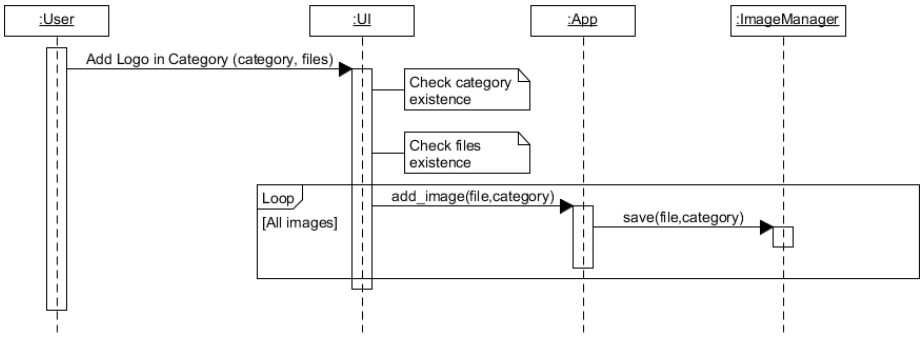
Remove feature variation



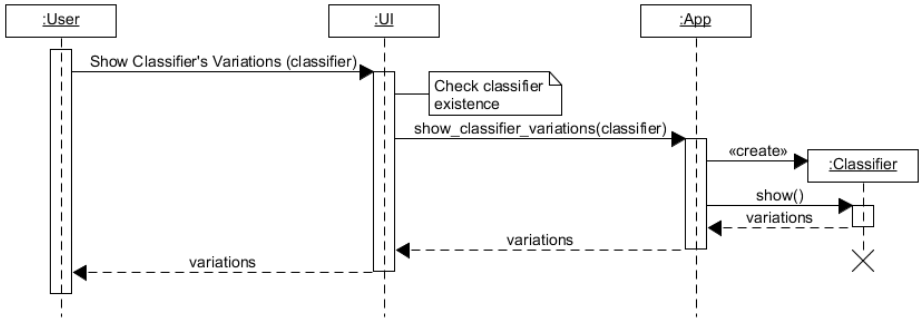
Show feature's variations



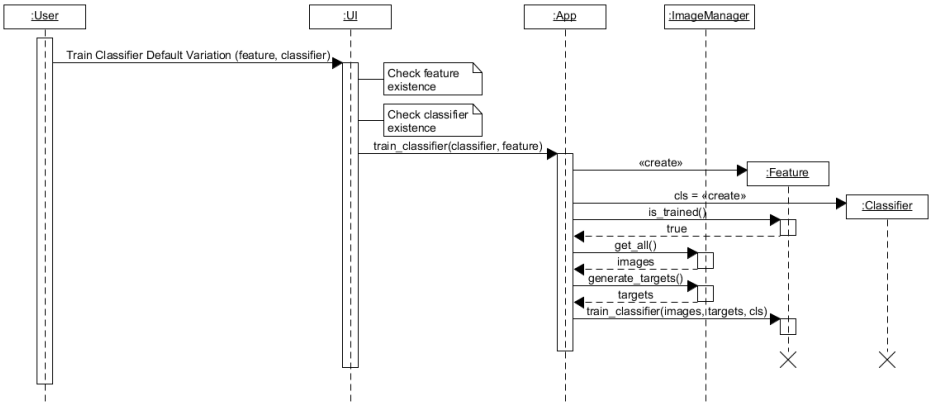
Add logo in category



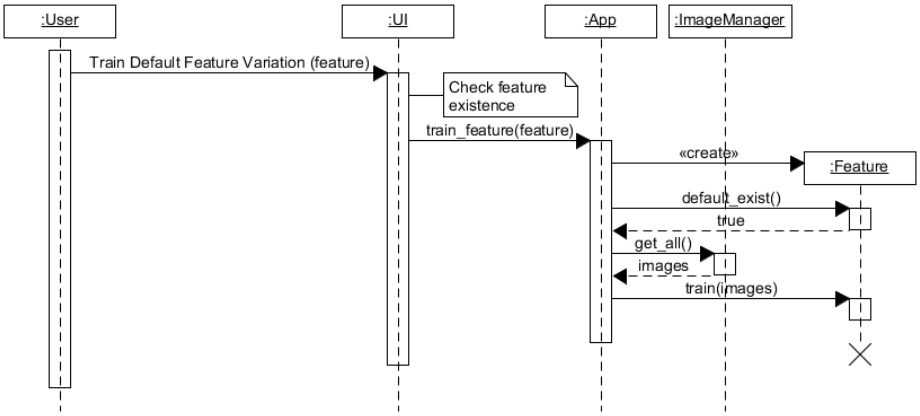
Show classifier's variations



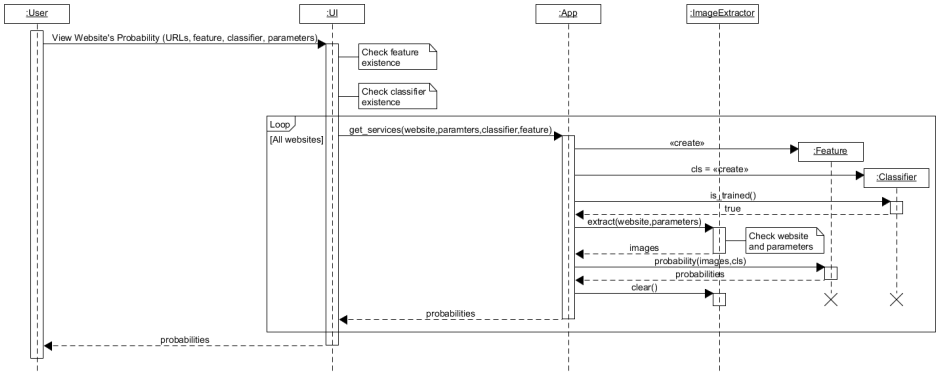
Train classifier default variation



Train feature default variation



View website's probability



B. Results

This appendix contains different images which represent the results of the application in recognizing images with multiples logos. The results are really good but sometime the application does not detect a logo which is indeed in the image. This problem is due to the sensitivity of 80% which does not allow having always perfect results. On the other hand, the precision of 100% makes sure that if the application says that there is a logo, it is really there.

At the moment the system can recognize six different logos i.e. DHL, FedEx, MasterCard, PayPay, UPS and Visa.

Figure B.1 shows a very good result where the application recognize perfectly the logo which is present in the system (i.e. FedEx). This is a very difficult example because especially all the purple logo look really similar.

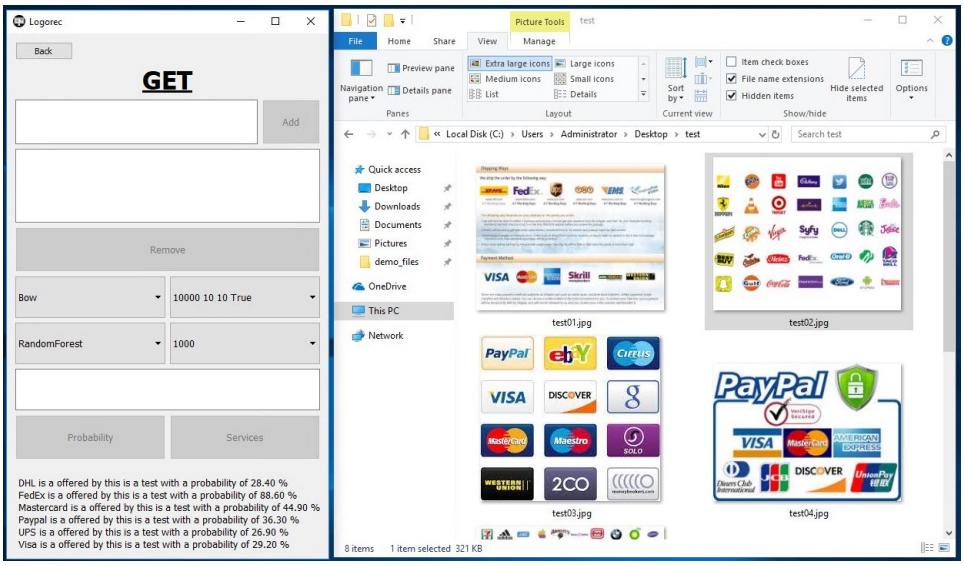


Figure B.1: Recognition of FedEx among similar logos

Figure B.2 shows a situation where the system struggle in recognizing all the logos. In fact, in this case, even if the application recognizes the PayPal's

and MasterCard's logos, the system does not detect the Visa logos. This error is due to the sensitivity of 80%.

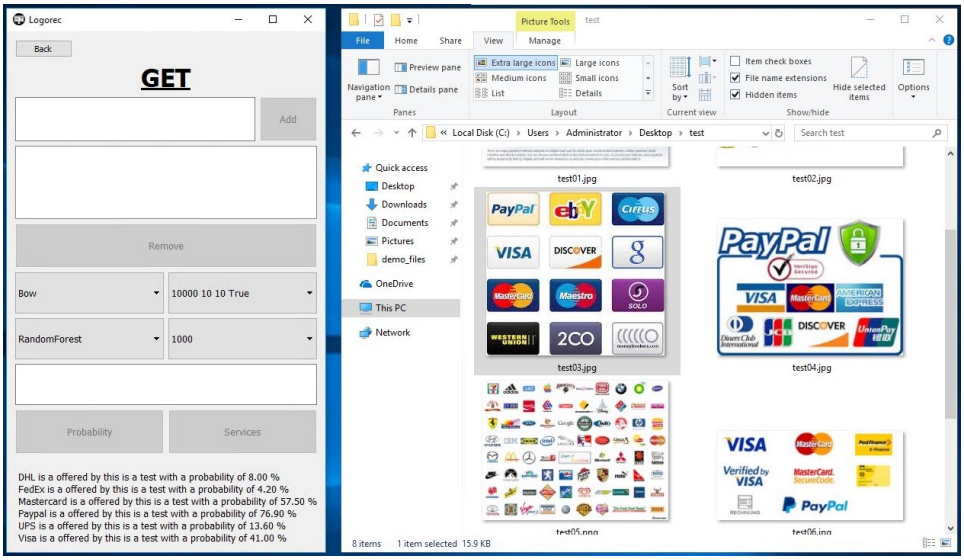


Figure B.2: Recognition of PayPal and MasterCard but not Visa

Figure B.3 shows an example which does not contain any logos. In this case, the system correctly does not find any logos (probabilities below 50%).

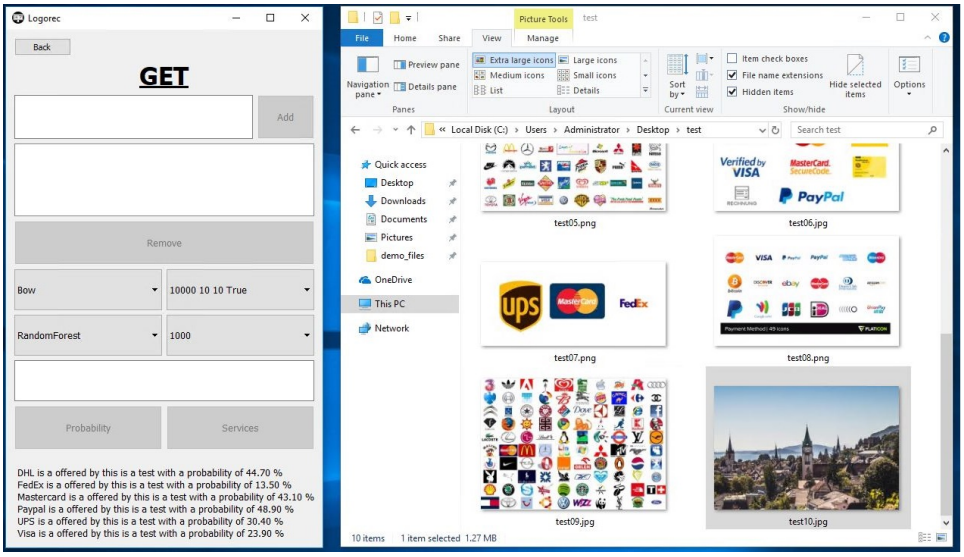


Figure B.3: Recognition of MasterCard among multiples logos

Attachments

Project Management — Logos Recognition for Webshop Services

Meetings Recap — Logos Recognition for Webshop Services

Acronyms

a.k.a. also known as. 4, 8, 10, 11, 13, 14, 16, 19, 21–23, 25, 28, 34, 42, 52, 53

AI Artificial Intelligence. 4

ANN Artificial Neural Network. III, V, VI, 2–6, 9, 11, 29, 49, 57, 83–88

API Application Program Interface. 72, 79

ARFF Attribute Relation File Format. 48

BoW Bag of Words. VII, 5, 7, 19, 22, 23, 25, 26, 28, 29, 34, 37, 40, 44, 46, 49, 51, 56–63, 70, 71, 73, 76, 77, 89

CNN Convolution Neural Network. III, VI, 5, 6, 49, 87, 88, 90

DoG Difference-of-Gaussian. V, 20

e.g. for the sake of example (*exempli gratia*). 17, 30, 56

FN false negative. 14, 15

FP false positive. 14, 15, 17, 18, 34, 52, 55, 56, 59, 60

FP functional programming. 11

GUI Graphical User Interface. VI, 72, 79, 81

HOG Histogram of Oriented Gradients. VI, 5, 57, 62

i.e. that is (*id est*). 3, 5, 11, 14, 21, 26, 28, 30, 36, 38, 45, 49, 51, 57, 62, 65, 69, 70, 72, 74, 75, 77, 78, 86, 100

ILSVRC ImageNet Large Scale Visual Recognition Competition. 5

IoU Intersection over Union. VI, 59, 60

KNN K-Nearest Neighbour. V, VII, 37–40

NPL Natural Language Processing. 22, 23

NPV Negative Predictive Value. 16, 18, 48, 50

OOP Object-Oriented Programming. 11

PPV Positive Predictive Value. 16, 34, 37, 40, 44, 46–48, 50, 55

R-CNN Regional CNN. 5

RBF Radial Basis Function. 32, 34

ReLU Rectified Linear Unit. 86

REPL Read-Eval-Print-Loop. 11

SD Sequence Diagram. III, VI, 71, 72, 95–99

SIFT Scale-invariant feature transform. II, VI, 1, 2, 5–8, 19–22, 25, 26, 28, 55–57, 61, 62

SSD System Sequence Diagram. III, VI, 71, 92–94

SVM Support Vector Machine. V, 8, 12, 30–34, 37, 40, 47, 78, 89

TN true negative. 14, 15, 18

TNR True Negative Rate. 16, 48, 50

TP true positive. 14, 15, 17, 18, 52, 53, 59, 60

TPR True Positive Rate. 16, 34, 44, 48, 50, 57

UI User Interface. III, 71, 72, 79, 80

YOLO You Only Look Once. 5

Glossary

dynamically typed In a dynamically typed language, every variable name is (unless it is null) bound only to an object [93]. 10

functional programming FP is a programming paradigm, a style of building the structure and elements of computer programs, that treats computation as the evaluation of mathematical functions and avoids changing-state and mutable data [94]. 11

GIT Git is a version control system for tracking changes in computer files and coordinating work between multiple people. 3

GitLab GitLab is a web-based git repository that have additional features like wiki, issue tracking, etc. This product support CI Runners that allow the user to execute builds on machines created on demand which once the build is finished, can wait to run the next builds or can be removed. This scaling can be achieved with Docker and its public machines. VI, 82

latex Is a mark up language specially suited for scientific documents. 2

MiKTeX Is an up-to-date implementation of TeX/LaTeX and related programs. 2

Object-Oriented Programming OOP is a programming paradigm based on the concept of "objects", which are data structures that contain data, in the form of fields, often known as attributes; and code, in the form of procedures, often known as methods [94]. 11

pattern matching Pattern matching is a mechanism for checking a value against a pattern. A successful match can also deconstruct a value into its constituent parts. It is a more powerful version of the switch statement in Java and it can likewise be used in place of a series of if/else statements[95]. 11

Read-Eval-Print-Loop REPL a.k.a. interactive toplevel or language shell, is a simple, interactive computer programming environment that takes single user inputs (i.e. single expressions), evaluates them, and returns the result to the user [96]. 11

static typing In a statically typed language, every variable name is bound both to a type (at compile time, by means of a data declaration) or to an object [93]. 10

TeXstudio TeXstudio is an integrated writing environment for creating LaTeX documents. 2

Wafers In electronics, a wafer (also called a slice or substrate) is a thin slice of semiconductor material, such as a silicon crystal, used in the fabrication of integrated circuits and other microdevices[97]. 5

wire-bounding process Wirebonding, or wire bonding, is the process of providing electrical connection between the silicon chip and the external leads of the semiconductor device using very fine bonding wires[98]. 4

Bibliography

[1] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015. V, 6

[2] The process of local extrema detection based on difference-of-gaussian... `https://www.researchgate.net/figure/The-process-of-local-extrema-detection-based-on-Difference-of-Gaussian-DoG_fig7_273897393`. (Accessed on 03/19/2018). V, 20

[3] David G Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110, 2004. V, 5, 19, 20

[4] Sift: Theory and practice: Keypoint orientations - ai shack. `http://aishack.in/tutorials/sift-scale-invariant-feature-transform-keypoint-orientation/`. (Accessed on 03/19/2018). V, 21

[5] Sift feature detection - image processing. `https://ongspxm.github.io/blog/2015/01/sift-feature-detection-image-processing/`. (Accessed on 04/24/2018). V, 22

[6] 1.4. support vector machines — scikit-learn 0.19.1 documentation. `http://scikit-learn.org/stable/modules/svm.html`. (Accessed on 03/08/2018). V, 30, 32

[7] Introduction to knn, k-nearest neighbors : Simplified. `https://www.analyticsvidhya.com/blog/2014/10/introduction-k-neighbours-algorithm-clustering/`. (Accessed on 03/08/2018). V, 37, 39

[8] My pet projects: Constructing decision tree using weka. `https://mypetprojects.blogspot.ch/2010/05/decision-tree-with-weka-part-i.html`. (Accessed on 05/07/2018). V, 48

[9] J. R. R. Uijlings, K. E. A. van de Sande, T. Gevers, and A. W. M. Smeulders. Selective search for object recognition. *International Journal of Computer Vision*, 104(2):154–171, 2013. V, 8, 52

[10] L. Fei-Fei, R. Fergus, and Pietro Perona. Learning generative visual models from few training examples. In *An Incremental Bayesian Approach Tested on 101 Object Categories*, 2004. VI, 57, 58

[11] Intersection over union (iou) for object detection - pyimagesearch. `https://www.pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/`. (Accessed on 04/10/2018). VI, 59, 60

[12] Yongzheng Xu, Guizhen Yu, Yunpeng Wang, Xinkai Wu, and Yalong Ma. A hybrid vehicle detection method based on viola-jones and hog+ svm from uav images. *Sensors*, 16(8):1325, 2016. VI, 62

[13] Neural network — tikz example. `http://www.texample.net/tikz/examples/neural-network/`. (Accessed on 12/22/2017). VI, 84

[14] What is a neural network? `https://norasandler.com/2017/10/19/What-is-a-Neural-Network.html`. (Accessed on 12/07/2017). VI, 84, 85, 86

[15] Cs231n convolutional neural networks for visual recognition. `https://cs231n.github.io/convolutional-networks/`. (Accessed on 06/09/2018). VI, 87, 88

[16] Manzoni Noli. Logos recognition. Technical report, Bern University of Applied Science, 2018. Project 2 report. 1, 2, 6, 7, 8, 10, 13, 19

[17] Latex. Project latex. `https://www.latex-project.org/`, 2010. (Accessed 21.09.2017). 2

[18] Miktexorg - typesetting beautiful documents. `https://miktex.org/`. (Accessed on 02/20/2018). 2

[19] Texstudio - latex made comfortable. `https://www.texstudio.org/`. (Accessed on 02/20/2018). 2

[20] What is object recognition? - definition from whatis.com. `https://whatis.techtarget.com/definition/object-recognition`. (Accessed on 04/18/2018). 4

[21] Object recognition - matlab & simulink. `https://ch.mathworks.com/discovery/object-recognition.html`. (Accessed on 04/18/2018). 4

[22] Alexander Andreopoulos and John K. Tsotsos. 50 years of object recognition: Directions forward. *Computer Vision and Image Understanding*, 117(8):827 – 891, 2013. 4

[23] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 1, pages 886–893. IEEE, 2005. 5, 57

[24] P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. In *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*, volume 1, pages I–511–I–518 vol.1, 2001. 5, 57, 61

[25] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012. 5

[26] The modern history of object recognition — infographic. `https://medium.com/@nikasa1889/the-modern-history-of-object-recognition-infographic-aea18517c318`. (Accessed on 04/19/2018). 5

[27] A brief history of cnns in image segmentation: From r-cnn to mask r-cnn. `https://blog.athelas.com/a-brief-history-of-cnns-in-image-segmentation-from-r-cnn-to-mask-r-cnn-34ea83205de4`. (Accessed on 04/19/2018). 5

[28] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015. 5

[29] Simone Bianco, Marco Buzzelli, Davide Mazzini, and Raimondo Schettini. Deep learning for logo recognition. *CoRR*, abs/1701.02620, 2017. 8

[30] Selective search for object detection (c++ / python) — learn opencv. `https://www.learnopencv.com/selective-search-for-object-detection-cpp-python/`. (Accessed on 02/23/2018). 8, 51, 52

[31] Bogdan Alexe, Thomas Deselaers, and Vittorio Ferrari. Measuring the objectness of image windows. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34/11:2189–2202, 2012. 8

[32] J. Carreira and C. Sminchisescu. Cpmc: Automatic object segmentation using constrained parametric min-cuts. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(7):1312–1328, July 2012. 8

[33] I. Endres and D. Hoiem. Category-independent object proposals with diverse ranking. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(2):222–234, Feb 2014. 8

[34] Santiago Manen, Matthieu Guillaumin, and Luc Van Gool. Prime object proposals with randomized prim's algorithm. In *Computer Vision (ICCV), 2013 IEEE International Conference on*, pages 2536–2543. IEEE, 2013. 8

[35] Html scraping — the hitchhiker's guide to python. `http://docs.python-guide.org/en/latest/scenarios/scrape/`. (Accessed on 02/27/2018). 9

[36] M. R. Henzinger. Hyperlink analysis for the web. *IEEE Internet Computing*, 5(1):45–50, Jan 2001. 9

[37] Python & java: A side-by-side comparison — python conquers the universe. `https://pythonconquerstheuniverse.wordpress.com/2009/10/03/python-java-a-side-by-side-comparison/`. (Accessed on 02/20/2018). 10

[38] Tiobe index — tiobe - the software quality company. `https://www.tiobe.com/tiobe-index/`. (Accessed on 02/20/2018). 10

[39] Java dip - open source libraries. `https://www.tutorialspoint.com/java_dip/open_source_libraries.htm`. (Accessed on 02/20/2018). 10

[40] 9 python libraries which can help you in image processing - data science central. `https://www.datasciencecentral.com/profiles/blogs/9-python-libraries-which-can-help-you-in-image-processing`. (Accessed on 02/20/2018). 10

[41] Curtis T. Rueden, Johannes Schindelin, Mark C. Hiner, Barry E. DeZonia, Alison E. Walter, Ellen T. Arena, and Kevin W. Eliceiri. Imagej2: Imagej for the next generation of scientific image data. *BMC Bioinformatics*, 18(1):529, Nov 2017. 10

[42] Johannes Schindelin, Ignacio Arganda-Carreras, Erwin Frise, Verena Kaynig, Mark Longair, Tobias Pietzsch, Stephan Preibisch, Curtis

Rueden, Stephan Saalfeld, Benjamin Schmid, et al. Fiji: an open-source platform for biological-image analysis. *Nature methods*, 9(7):676, 2012. 10

[43] Pillow - pillow (pil fork). `https://pillow.readthedocs.io/en/latest/index.html`. (Accessed on 02/20/2018). 10

[44] Stéfan van der Walt, Johannes L. Schönberger, Juan Nunez-Iglesias, François Boulogne, Joshua D. Warner, Neil Yager, Emmanuelle Gouillart, Tony Yu, and the scikit-image contributors. scikit-image: image processing in Python. *PeerJ*, 2:e453, 6 2014. 10

[45] Eric Jones, Travis Oliphant, Pearu Peterson, et al. SciPy: Open source scientific tools for Python. http://www.scipy.org/, 2001–. (Accessed on 02/20/2018). 10

[46] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000. 10, 22, 28, 53, 61

[47] Scala vs. java: Another view - dzone java. `https://dzone.com/articles/scala-vs-java-another-view`. (Accessed on 02/20/2018). 10, 11

[48] Github - sksamuel/scrimage: Scala image processing library. `https://github.com/sksamuel/scrimage`. (Accessed on 02/20/2018). 11

[49] Ethem Alpaydin. *Introduction to machine learning*. MIT press, 2014. 12

[50] Hudritsch Marcus. Introduction to image analysis - classification. CPVR course at Bern University of Applied Science, 2017. 12, 13, 15

[51] JINHO KIM[1], Byung-Soo Kim, and Silvio Savarese. Comparing image classification methods: K-nearest-neighbor and support-vector-machines. *Ann Arbor*, 1001:48109–2122, 2012. 22

[52] Vbow pt 2 - image classification in python with visual bag of words (vbow) – ian london's blog. `https://ianlondon.github.io/blog/visual-bag-of-words/`. (Accessed on 03/19/2018). 22, 23, 26

[53] Stephen O'Hara and Bruce A Draper. Introduction to the bag of features paradigm for image classification and retrieval. *arXiv preprint arXiv:1101.3354*, 2011. 22, 23

[54] Bag of visual words model for image classification and recognition - bitsmakemecrazy ¡br¿ kushal vyas's blog. https://kushalvyas.github.io/BOV.html. (Accessed on 03/19/2018). 22, 26

[55] Bag of words (bow) - natural language processing. https://ongspxm.github.io/blog/2014/12/bag-of-words-natural-language-processing/. (Accessed on 03/19/2018). 22

[56] Wikipedia contributors. The canterbury tales — wikipedia, the free encyclopedia. https://en.wikipedia.org/wiki/The_Canterbury_Tales, 2004. (Accessed on 03/19/2018). 23

[57] Introduction to k-means clustering. https://www.datascience.com/blog/k-means-clustering. (Accessed on 03/22/2018). 26

[58] Zhiwu Lu, Liwei Wang, and Ji-Rong Wen. Image classification by visual bag-of-words refinement and reduction. *Neurocomputing*, 173:373–384, 2016. 28

[59] Huilin Gao, Lihua Dou, Wenjie Chen, and Jian Sun. Image classification with bag-of-words model based on improved sift algorithm. In *Control Conference (ASCC), 2013 9th Asian*, pages 1–6. IEEE, 2013. 28

[60] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. The weka data mining software: An update. *SIGKDD Explor. Newsl.*, 11(1):10–18, November 2009. 29

[61] Anrig Dr. Bernhard. Computer perception and artificial intelligence - data mining. CPVR course at Bern University of Applied Science, 2017. 30, 35

[62] Lars Buitinck, Gilles Louppe, Mathieu Blondel, Fabian Pedregosa, Andreas Mueller, Olivier Grisel, Vlad Niculae, Peter Prettenhofer, Alexandre Gramfort, Jaques Grobler, Robert Layton, Jake VanderPlas, Arnaud Joly, Brian Holt, and Gaël Varoquaux. API design for machine learning software: experiences from the scikit-learn project. In *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, pages 108–122, 2013. 30, 32, 36, 38, 39, 42, 43, 44

[63] Chapter 2 : Svm (support vector machine) — theory – machine learning 101 – medium. https://medium.com/machine-learning-101/chapter-2-svm-support-vector-machine-theory-f0812effc72. (Accessed on 03/08/2018). 30, 33

[64] Understanding support vector machine algorithm from examples (along with code). https://www.analyticsvidhya.com/blog/2017/09/understaing-support-vector-machine-example-code/. (Accessed on 03/08/2018). 30, 33

[65] Chapter 2 : Svm (support vector machine) — coding – machine learning 101 – medium. https://medium.com/machine-learning-101/chapter-2-svm-support-vector-machine-coding-edd8f1cf8f2d. (Accessed on 03/08/2018). 33

[66] 6 easy steps to learn naive bayes algorithm (with code in python). https://www.analyticsvidhya.com/blog/2017/09/naive-bayes-explained/. (Accessed on 03/08/2018). 34, 36

[67] 1.9. naive bayes — scikit-learn 0.19.1 documentation. http://scikit-learn.org/stable/modules/naive_bayes.html. (Accessed on 03/08/2018). 34, 36

[68] Naive bayesian. http://www.saedsayad.com/naive_bayesian.htm. (Accessed on 03/08/2018). 34

[69] A quick introduction to k-nearest neighbors algorithm. https://medium.com/@adi.bronshtein/a-quick-introduction-to-k-nearest-neighbors-algorithm-62214cea29c7. (Accessed on 03/08/2018). 37

[70] 1.6. nearest neighbors — scikit-learn 0.19.1 documentation. http://scikit-learn.org/stable/modules/neighbors.html. (Accessed on 03/08/2018). 37, 38, 39, 40

[71] Eamonn Keogh and Abdullah Mueen. *Curse of Dimensionality*, pages 314–315. Springer US, Boston, MA, 2017. 37

[72] A complete tutorial on tree based modeling from scratch (in r & python). https://www.analyticsvidhya.com/blog/2016/04/complete-tutorial-tree-based-modeling-scratch-in-python/. (Accessed on 03/08/2018). 40, 41, 42, 44, 45

[73] 1.10. decision trees — scikit-learn 0.19.1 documentation. http://scikit-learn.org/stable/modules/tree.html. (Accessed on 03/15/2018). 40

[74] Decision trees in machine learning – towards data science. https://towardsdatascience.com/decision-trees-in-machine-learning-641b9c4e8052. (Accessed on 03/08/2018). 41, 42

[75] How random forest algorithm works in machine learning. https://medium.com/@Synced/how-random-forest-algorithm-works-in-machine-learning-3c0fe15b6674. (Accessed on 03/08/2018). 45

[76] I.H. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann series in data management systems. Morgan Kaufman, 2005. 48

[77] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012. 49

[78] MATLAB. *version 9.4.0 (R2018a)*. The MathWorks Inc., Natick, Massachusetts, 2018. 49

[79] Duomo di milano - homepage. https://www.duomomilano.it/en/. (Accessed on 04/04/2018). 55

[80] Share a picture of a tree, plant or anything green you see here! - 7 cups forum. https://www.7cups.com/forum/LaurasOffice_133/ThoughtsIdeasIcebreakersandFun_744/Shareapictureofatreeplantoranythinggreenyouseehere_81849/. (Accessed on 04/04/2018). 55

[81] P. F. Felzenszwalb, R. B. Girshick, D. McAllester, and D. Ramanan. Object detection with discriminatively trained part-based models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(9):1627–1645, Sept 2010. 57, 59

[82] Opencv: Face detection using haar cascades. https://docs.opencv.org/3.4.1/d7/d8b/tutorial_py_face_detection.html. (Accessed on 05/02/2018). 61

[83] Computer vision: What is the difference between hog and sift feature descriptor? - quora. https://www.quora.com/Computer-Vision-What-is-the-difference-between-HOG-and-SIFT-feature-descriptor. (Accessed on 05/02/2018). 62

[84] Histogram of oriented gradients — learn opencv. https://www.learnopencv.com/histogram-of-oriented-gradients/. (Accessed on 05/02/2018). 62

[85] K. Pohl and C. Rupp. *Requirements Engineering Fundamentals: A Study Guide for the Certified Professional for Requirements Engineering Exam - Foundation Level - Ireb Compliant*. Rocky Nook computing. Rocky Nook, 2015. 65

[86] Strategy design pattern. https://sourcemaking.com/design_patterns/strategy. (Accessed on 05/29/2018). 70

[87] Overview — sphinx 1.8.0+ documentation. http://www.sphinx-doc.org/en/master/. (Accessed on 06/04/2018). 74

[88] Welcome to the click documentation — click documentation (5.0). http://click.pocoo.org/5/. (Accessed on 06/04/2018). 80

[89] Pyqt - python wiki. https://wiki.python.org/moin/PyQt. (Accessed on 06/04/2018). 81

[90] 25.3. unittest — unit testing framework — python 2.7.15 documentation. https://docs.python.org/2/library/unittest.html. (Accessed on 06/07/2018). 81

[91] Maureen Caudill. Neural networks primer, part i. *AI Expert*, 2(12):46–52, December 1987. 83

[92] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. *Citeseer*, 2009. 87

[93] Static vs. dynamic typing of programming languages - python conquers the universe. https://pythonconquerstheuniverse.wordpress.com/2009/10/03/static-vs-dynamic-typing-of-programming-languages/. (Accessed on 02/20/2018). 105, 106

[94] Codenewbie. https://www.codenewbie.org/blogs/object-oriented-programming-vs-functional-programming. (Accessed on 02/20/2018). 105

[95] Pattern matching — scala documentation. https://docs.scala-lang.org/tour/pattern-matching.html. (Accessed on 02/20/2018). 105

[96] Wikipedia contributors. Read-eval-print loop (repl) — wikipedia, the free encyclopedia. https://en.wikipedia.org/wiki/Read\OT1\textendasheval\OT1\textendashprint_loop, 2004. (Accessed on 02/20/2018). 106

[97] About wafers: Interesting information about silicon wafers. http://www.addisonengineering.com/about-wafers.html. (Accessed on 04/18/2018). 106

[98] Wire bonding or wirebonding process.
`http://eesemi.com/wirebond.htm`. (Accessed on 04/18/2018). 106