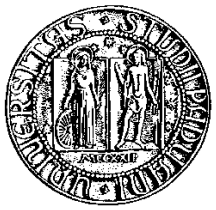

Università degli Studi di Padova

Facoltà di Ingegneria

Corso di Laurea Specialistica in Ingegneria Informatica

tesi di laurea

Logo Detection Application

Extraction, Matching, Segmentation and Classification

Relatore: Massimo Melucci
Correlatore: Roelof van Zwol

Laureando: Michele Trevisiol

13 Aprile, 2010

Aknowledgements

Questa tesi di laurea è stata la conclusione di un'esperienza incredibile che ho vissuto tra Padova e Barcellona, tra un mondo universitario e un ambiente lavorativo di ricerca, un'avventura che mi ha fatto crescere e imparare molto. Tutto questo lo devo al prof. Massimo Melucci dell'Università di Padova, che mi ha permesso di scoprire e di vivere un mondo nuovo.
A Yahoo! sono stato guidato e accompagnato da persone eccezionali: il prof. Ricardo Baeza-Yates di Yahoo! Research Barcelona, che mi ha dato un'opportunità per il mio futuro, il dr. Roelof van Zwol e l'ing. Lluis Garcia Pueyo, con cui ho collaborato, che mi hanno fatto conoscere il mondo della ricerca seguendomi in questi mesi, dandomi fiducia e insegnandomi tanto.

Ringrazio la mia famiglia, mia mamma Daniela, mio papà Maurizio e mia sorella Martina, che non mi hanno mai negato un'opportunita' e che sono sempre stati presenti.

Un ringraziamento speciale va a Marta, che è stata sempre al mio fianco, dandomi un appoggio per non cadere e con la quale spero di vivere ancora tante importanti esperienze. Non posso non citare Nello, Balto, Enrico, Elisa, Matteo, Lally, Fox e ancora Matteo che sono persone su cui posso sempre contare. Silvia, per la sua costante presenza, Alvise, Stefano e Mauro, per essermi stati vicini soprattutto in questi ultimi momenti.

Ringrazio tutti i miei compagni di università, con cui ho condiviso gli studi e gli esami, ma anche molte serate e stupende avventure. I miei amici di Verona e di calcetto con cui ho passato tanti momenti indimenticabili tra emozioni, viaggi e terzi tempi. Ultime ma non meno importanti le nuove amicizie di Barcellona, legami nuovi ma forti che mi hanno accompagnato in questo percorso.

Michele

Contents

Abstract

This thesis describes the research work carried out to fulfil the Master in Computer Science at the University of Padua. The work was performed during a visit at Yahoo! Research in Barcelona and was supervised by Roelof van Zwol and Lluis Garcia Puyeo. The visit was funded by the EU Erasmus Programme. The research work described in this thesis was part of a larger research project which is underway at Yahoo! Research in Barcelona.

The resarch work described in this thesis aimed at addressing the problem of detecting and retrieving all the logos contained in an image given as input. The problem of detecting and retrieving a logo consists of a variety of steps, each of which has different possible solutions. These steps were investigated in this research work. Although logo detection is an important research problem due to the potential industrial impact, a solution has not yet been proposed in the literature to our knowledge.

This thesis is organized in two parts: collection preparation and experimental study. As regards to the former, a collection of logos was designed and implemented to train the classifier, to identify and to extract the logo features which were eventually used for logo detection. The latter regards the detection of logos from an input image. In particular, the experimental study aimed to detect if the input image contains one or more logos and to decide which logos are contained.

Chapter 1

Introduction

Being able to analyze a video, extract frames and trying to figure out what they contain automatically is still an open challenge. The problem of entity detection and recognition in videos can be seen as the analysis of a frames sequence and, therefore, simplified to the analysis of single image. Unfortunately there are no easy ways to identify correctly what a picture contains.

This thesis is a peace of a larger project focused on the analysis of images, to examine whether it contains logos, and if so, to recognize them. What is a logo? It's a name, symbol or trademark designed for an easy recognition. Thus, given a generic image, the goal of the project is to compare its content with all the brands in a database and identify which ones match with input image. Moreover, we will try to locate the logo in the input image, obtaining the correct position.

To better understand how it's structured this work, it's necessary split the main process in two parts: the first one is the data collections preparation, a large set of logos and images for training and testing, and the second one is the real application. This part is resumed in the following steps:

- *Features Extraction:* Extract the key-points of all the logos (and all the test images). These points are very descriptive for the contents of an image, and they are used to compare and to analyze parts of images.

- *Features Segmentation:* Given a large set of key-points, extracted from the logos collection, we have to segment all of them to obtain groups of common features.

For real application we mean the main goal of the project, that is how to retrieve the logo(s) from an image given as input. We can resume this procedure in the following points:

- *Extract the features:* Define the key-points of the input image with the same technique used to extract the key-points from the logos.
- *Classify the features:* Given the features of the image, we want to find the logos segment that match better each key-point. All the logos points in those segments are considered possibles candidates logos.
- *Rank the candidates:* Now we have a list of logos which have in commons some key-points with the input image. We want to rank these points by some characteristics and keep only the most similar ones.
- *Select the logo(s):* After the ranking we have all the information to decide if there is or not a logo inside the image and, in case, which one is contained. Besides in this part we can decide if there are more than one logo.

So, to arrange the assortment of brands, we retrieved all logos from a web site famous to host a large collection of more than 150K logos[1]. This collection constitutes our train set. On the other hand we used 16K images for testing, and we retrieved them from a website that guarantees the presence of logos in its images[2].

The main problem related to the project goal is *how to compare two images*. We don't need to compare if an image is more or less similar to another, but if an image is included into another. The aim is very different and the complexity too. To do this there is only one way in literature: extract the features points of both images and use them for the comparison. The first issue is how to treat these points, because if we want to detect the logos features we need a way to compare all the input image points with all the logos ones: and there is no question to do it point-to-point. In this context it is very important to know what image *features* are: they are the characteristic parts of an image, texture, color, and shape are examples of image features. There are some algorithms that can computes

[1] Brands of the World, http://www.brandsoftheworld.com/

[2] Group "Brands, Marks & Logos" at Flickr, http://www.flickr.com/groups/famousbrands/

them as vectors, usually they are structured with 64 or 128 dimensions of floating values. It is necessary clarify the difference from *global* and *local* features. The first one describes the whole image, or a big part of that, as the brightness or the contrast of a photo. The second one is more specific, it describes a point, or a group of close points that are very characteristics for the image, but for example, the brightness in a small part can be very different from the global value. This project use the local features because it compares parts of images and not the entire ones. The main problem then is how to extract features from images and how to match them with logos in the training set. We extracted them obtaining a unique signature for every interesting point of the image. The choice of these points depends on the approach used: usually one feature is described by one vector called *local descriptor*. There are several methods to extract these points; in this thesis we'll discuss about the main ones and we'll explain our decision.
It is important to consider the working environment of this project, because the automatic recognition of logos implies a storage of a large number of data: images of logos, information on the label, features extracted, etc. Accordingly one of the problems is the computation time to compare the image with all the logos, we used 175K brands even if the number is expected to grow day by day.

For the features extraction, the problem is not in the image but in the brand, because a logo may consist of symbols, words or anything else. A common and problematic condition is that it could be very simple without some particular details than helps to a good detection; in conclusion it may be hard define a good features extractor and a good comparison metrics. After a brief research in literature, we decided to use the SURF descriptor. This decision was taken for two main reasons: precision and fast computation speed. Given the input image and its calculated features, we need to match its characteristics with the logos images in the collection. To avoid the traversal of the entire collection with one-to-one comparisons, we use some techniques to segment the features in sub-groups by common properties. A numerical example to understand better the situation should be this: a logos (200x200) has an average of 100 features points and our collection contains 175K logos so they will be 17Milion of points; instead an image (500x375) has in average 440 features points: so, a set of comparisons point-to-point should be about 7500 million operations and this is unacceptable.
We analyzed different methods to resolve the problem: K-Means clustering, Random Vectors, Balanced Decision Tree, etc. The idea is to split

the points collection in groups using one of these techniques, and retrieve a *representative vector* for each of them. So, once computed the vectors of the input image, they will be matched only with these vectors and not with all the collection. By this we can reduce the number of comparisons and consequently the computation time. To compare two vectors in a multidimensional space there are many way: we used the *Pearson Correlation* that measures the similarity calculating the cosine of the angle between them. It is a technique often used in Text Mining and in many other fields.

It's important to define how is structured this thesis because, as we said, this is a part of a more complex project. This is a Master Thesis in Computer Science and contains in details only the work developed by the undersigned. The section that regards parts of different authors are correctly highlighted. For that reason there are not all the experimentation and all the techniques used, but only the main ones, nevertheless the thesis is wrote to give a full idea of the project and it is structured as follow:

Chapter two: *"Related Works"*, It's essential specified the sources that have allowed us to study, understand and resolve the various problems encountered in the work.

Chapter three: *"Test Collection"*, In this chapter we'll describe the two collections used: one with the brands logos and the other one with Flickr's photos. The first one as logos source and the second one as testing images. Will be discuss the reasons of how it was possible to obtain these collections.

Chapter four: *"Features Extraction"*, Will be discuss about available features and its problems. There are different descriptors, we'll see the better one for the project after a short comparison with other methods. Furthermore there will be a brief some applicable effects to the images to try to detect the features in a more clearly way. In the end we'll describe how to match the points with related problems and solutions.

Chapter five: *"Collection Segmentation"*, In this chapter we'll analyze how to segment the collection. We'll describe the techniques trying to highlight the advantages and disadvantages of each methods with a short description of the implementation code.

Chapter six: *"Classification and Ranking "*, This is the *real application* process, it describes all the procedure from the input image to the

logos retrieved. It contains the classifications of the features extracted and the selection between the most representative logos segments. Moreover it contains the techniques to rank the logos candidates.

Chapter seven: *"Conclusions and Future Work"*, This is the final discussion about the work, with comments and observations and moreover with suggestions and future steps or goals to pursue.

Chapter 2

Related Work

In this chapter we will principally describe the state-of-the-art which will be the basis of our work: features descriptor, segmentation, classification and clustering. There will be a short description of many techniques already implemented and used in certain cases which are closer to our needs. There are different evaluation points to analyze, and some texts and papers were very useful for our decision.

When we talk about logo detection we must consider different meanings: the main one is the identification of the logo into a document [1]. In this case the detection is made easier because of the white background. In [2] is commonly used *geometric invariants* to prune the logos database and local invariants to have a more refined match; on the other hand, in [3] they used *spatial density* to involve a page segmentation computing. Another spread field is the removal of the channel brand from a television program. This issue is analyzed in many works by Zhu [4] [5] or by Zhang [6]. There are different ways used to solve these topic, but they are all based on some facts: in television the logo is commonly in a corner in the same position and, moreover, the background images change continuously, instead, the logo's pixels that remain fixed [7]. This is useful for commercial development: in the transmissions recorder where it's required an automatic removal of the advertising. It's interesting to notice that during a television program they keep the tv logo but not during the advertisements.

However the original goal of this project is to detect a logo in a video, the main issue is to manage and retrieve the multimedia data [8]. A brief but complete introduction of this area is described in [9]. The article was published in 2002 and gives the clue about a typical scheme of video-

content analysis structured on the following points: feature extraction, structure analysis, abstraction, and indexing. This is broadly the procedure adopted by our project, but each process poses many challenging research problems. In this area we cannot avoid mentioning the Sivic and Zisserman's Video Google [10, 11], these two works are done respectively in 2003 and in 2006 in which they created a strong connection between video and text retrieval. They compared a video frame to a document and they considered a visual interest point as a word, applying the techniques of Text Retrieval. Again, in 2005, they published a technical report for MIT [12] with unsupervised method. A peculiar aspect is the comparison with a semi-supervised approach [13]. In almost all these works they used the *SIFT* Descriptor [14]. In this project we will use a different kind of feature extractor algorithm with similar but faster performance, the so called *SURF* [15, 16]. There are other ways to extract local features from an image such as [17, 18, 19, 20], this does not depend on the presence of edges or corners in the image but it is purely intensity-based which is invariant to full affine transformations. The Harris detector [21] instead, based on the predecessor presented by Moravec [22] in his PhD thesis, has the aim to select the corners characteristics. On the other hand the *Maximally Stable Extremal Regions* [23] is another *blob detection* method which extracts some area of interest, similar to a image segmentation; there is additionally an extended version with colour support [24]. There are many debates and discussions about which is the best descriptor in [25, 26, 27, 28] for example, but it continues to be an issue without solution.

The main problem of logo detection in images is to find the best way to compare the features extracted from the images to the ones collected in the logo database. If we do it point-to-point a huge number of comparison will occur. For this reason we need some expedients. Further aid to our scope it is the segmentation of the image. Segmenting an image into regions it's very helpful because the area to extract the features is already segmented and we can compare the features only for determinate surfaces. MIT [29] and Berkeley [30] built some applications useful to test their segmentation techniques. The issue of these methods is the hard compatibility with the features extractor algorithm. Indeed as SIFT or SURF detect a lot of points near the corners or close the borders, computing them only in one region can lead to loss of information. Another way to reduce operations and computational time is to segment in groups all the set of vectors, and to analyze the image features to define which is the best group for every point obtained. To do this a segmentation method results necessary: this

is the way chosen by this project.

In this thesis, to split the vectors collection of logos' features, we use a *decision tree*, specifically a binary decision tree. This is a method commonly used in decision analysis to help identification. Indeed it's used to find the logo into a document too, for example in [31] to reduce the false positive from the candidates.

Chapter 3

Test Collection

The collection is the starting point for testing our hypothesis and making the tests. It's necessary to look for a complete logos resource, where there are not only images, but all the related *meta-data* as brand name, URL, tags and many other details. These information are important for the project, because we want to know all it is possible on the brands: when a logo is detected, the goal is also to show all the data we have of that brand.

One of the biggest collection available on line is *Brands Of The World* with more than 170K logos, with an average of 1.8K logos uploaded per month. So, implementing some applications to find automatically all the logos and the meta-data on the website, download and store them in the database. We did the same procedure for the training set, making sure that each images contains at least one brand. It is possible to use the *Group "Brands, Marks & Logos"* group as an images set already built.

In this chapter we will discuss how it was possible to get the brands and the Flickr images, with the problems encountered and the relative solutions. Later, will compare the problem of how to handle all these images. We implemented applications to search and show immediately the images and the relative information for all our sets.

3.1 Logo Detection Database

We implemented two different databases, the first one contains a collection of logos and it is describe below, the second one contains a collection of images and it is built to perform tests.

For manage a Large Data Collection where there is the necessity to adding, removing or in general changing the information, and it's required a good computation speed, it's usually works with a database as MySQL[1] in our situation.

The database, with the petty name *logodetection*, was initially very simple, with only a couple of table, one with brands and the other with famousbrands. Subsequently was necessary to work with different data, as splitting the famousbrands collection in two parts, the first with the images that does not contains any logos stored in the database, and the second with only images that contains that logos. To do this automatically, we were work with the tabs of the Flickr images to know what the image contains, and for the logos we normalized the brands names. We'll discuss better this point later, still in this chapter. Another important observation regards the images. It's not saved the single image as blob file in the database, but in there, there is only the path of the server where the image are located. This choose was did for keep the db most lightly, and because later there will be the possibility to modify on the images, with conversion, filters and any other effects.

The **brands table** contains all the logos with other information like:

```
[SQL command for build the brands table]
  create table brands
  {
    brandid integer auto_increment,
    code varchar(8) not null,
    name varchar (255) not null,
    url_logo varchar(255) not null,
    url_img varchar(255) not null,
    path img varchar(255),
    info text,
    primary key( brandid )
  }
```

- *code* is the same value used by the source website to identified the logo (8 digits),
- *name* of the brand,
- the *url of the logo brand* is the page where are extracted the information,

[1] MySQL 5.1.24 on RedHat Linux

- the *url of the image* is the direct link to the image,
- *path* is the local position of the downloaded image,
- other brand *information* saved as html text (Country name, website, ..)

The **famousbrands** table contains all the photo, enclose in the group *"Brands, Marks & Logos"* located in Flicr[2], and accessible by the Flickr API with the code: 75819424@N00. Later on we will describe it better. Below there is a brief description of the table structure:

```
[SQL command for build the famousbrands table]
  create table famousbrands
  {
    photoid integer auto_increment,
    flickrid bigint not null,
    photo_title varchar(255) not null,
    url_photo varchar(255) not null,
    path_photo varchar(255) not null,
    description text,
    tags text,
    primary key( photoid )
  }
```

- *flickrid* is the same code used by the source website (8 digits),
- *url_photo* is the web link where there are all the information (description, tags, owner, ..)
- *path_photo* is the local path where the photo is stored
- *description* is a short text wrote by the user owner
- *tags* are wrote by other users, quite important for our tests

These tables were fill in with a couple of python scripts. For the brands there were some problems be due to the website, which asks a confirmation before show up the logo url, instead, for the Flickrs images, was very easy using the API and a free key. Everything are explain in details in the next sections.

When we started some tests, we found the necessity to have a different kind of test set. So we analyze better the Flickr collection to be able

[2] http://wwww.flickr.com/groups/famousbrands/

to extract more information and to obtain more then one sub-collection. In the next section is going to be describe what we needed and how we obtained it.

3.2 Logos Collection

To download all the logos stored on the website, was necessary implement a program to works automatically. But we encountered few problems doing this. The first one is the kind of character-encoding scheme, so, for some logos originally from different country where the encoding is not the same, there could happen errors during the decoding of the strings. For a large collection of data this is a big problem, because it necessary later try to find all the wrong decoding and fix them. To partially resolve this problem, we used a kind of *brute force* substitution, between the download and the writting on the database, for the most commons character code. It is not an elegant solution, but for our scope it works fine and fast.

Source	http://www.brandsoftheworld.com/logo/
Server	media1.barcelona.corp.yahoo.com
Number of Logos	173012
Size	3.7G
Resolution	200 x 200 @ GIF
SURF Vectors	100 (average)

Table 3.1. Retrieved Logos Collection

In the table 3.1, is possible to see a short number of information about the collection. All the logos are in GIF format, this will be a problem for some feature extraction algorithm as OpenSURF, because normally they could work only with JPEG files. The resolution is quite normal for a logo, we made different tests with other resolution, but all the images in this collection keep the same. The average number of SURF vectors refers to the algorithm OpenSURF, there are logos with a poor number of vectors retrieved (23/40) and this is a problem for the comparisons, but depends of the structure of the image (background dark, without corners or angles, words or symbols,...).

3.2.1 Developing

To getting all the logos we implemented a python script, the goal of this application is to retrieve logos from the web and store them locally, in order to extract visual features and experiment with them in future steps. Starting from the main directory:

```
http://www.brandsoftheworld.com/logo/
```

moving in all the subdirectory (or section)

```
http://www.brandsoftheworld.com/logo/1/
http://www.brandsoftheworld.com/logo/A/
http://www.brandsoftheworld.com/logo/B/
[...]
```

and analyzing all the pages in each section

```
http://www.brandsoftheworld.com/logo/1/index-1.html
http://www.brandsoftheworld.com/logo/1/index-2.html
http://www.brandsoftheworld.com/logo/1/index-3.html
[...]
```

In this last step the first problem appears. Supposing the first section have only 16 pages, a normal procedure is to increase the counter and to download the "*../1/index-17.html*" page, facing an error. So, it is easy to change section and start the download again. The problem is during the download of a page out of range (for example the number 17), the server returns every time the last page of the section (in this example the number 16) and the script could not understand when he has to change section. To fix this, the script computes, for a new page, the MD5 and compares the value with the last page downloaded, so if the hashing is the same it has to change section, otherwise keeps to increment the counter.

Each script implemented keeps a *logfile* for the possible errors and this is not rare during a long downloading as all the brands or all the Flickr images. For that the scripts after the parsing of all the sections, checks the logfile and try to fix all the entry automatically.

3.3 Flickr Collection

Comparing the table 3.2 with the 3.1, is possible to see some differences. The most interesting are the average number of vectors extracted, in this case is more than 4 time bigger. This happen not only for the resolution much large of these images, but also for the structure of a photo, usually

Source	http://www.flickr.com/group/famousbrands/
Server	media1.barcelona.corp.yahoo.com
Number of Images	15881
Size	2.0G
Image Resolution	500 x 400 (most common) @ JPG
SURF Vectors	440 (average)

Table 3.2. Retrieved Flickr Collection

much more complex than a logo.

Working with a Flickr database involve some advantage, to obtain the photos is possible to use a method much more elegant: the Flickr API[3], simply a set of callable methods. We exploit them in the python script to download all the photos and the relative information, and in Java to acquire automatically the candidates of an input image. To used that API is necessary have an API key, we used a Non-Commercial one according with the Flickr rules. To understand better the use of Flickr API there are several examples below:

```
http://api.flickr.com/services/rest/?
api_key=fcc535b428d18e3df66711ff87dcadd9
&method=flickr.groups.pools.getPhotos
&group_id=75819424@N00
```

Not all the query requires the API key, but usually the structure it is this. The *method* defines which function call, in this case it retrieve an xml file with all the photos include in the group. The *group_ id* is a parameter required by this specific method. An example of output is that:

```
<rsp stat="ok">
  <photos page="1" pages="201" perpage="100" total="20042">
    <photo id="4341371478" owner="29257248@N02" secret="a6e2a802a4"
     server="4024" farm="5" title="Adminsoft logo / final result" ispublic="1"
     isfriend="0" isfamily="0" ownername="Franz Vanek" dateadded="1265651497"/>
    <photo id="4240922072" owner="32156469@N06" secret="62ccb54943"
     server="4011" farm="5" title="GUINNESS 2010" ispublic="1" isfriend="0"
     isfamily="0" ownername="SIMONE RAVERA PHOTOÂ©" dateadded="1265639112"/>
    <photo id="4340851274" owner="88414926@N00" secret="f999f979ab" server="4027"
     farm="5" title="Diet Mt Dew Ultra Violet" ispublic="1" isfriend="0"
     isfamily="0" ownername="Paxton Holley" dateadded="1265638597"/>
  </photos>
</rsp>
```

[3] http://www.flickr.com/services/api/

The XML is an optimal way to work, there are plug-ins to support it in every programming language. We use it in Python, with the packages *xml.dom.minidom* and *xml.parsers.expat*.
In spite of the query result is a good bunch of information, is missing the URL of the images, and so it is necessary to perform another query. The only information required to identify the image is the *id*. The command to find the information, for example for the first photo retrieved, is the follow:

```
http://api.flickr.com/services/rest/?
api_key=fcc535b428d18e3df66711ff87dcadd9
&method=flickr.photos.getInfo
&photo_id=4328326553
```

And in this case the output will be:

```
<rsp stat="ok">
  <photo id="4328326553" secret="7d4daaba0c" server="2789" farm="3" isfavorite="0"
   dateuploaded="1265239245" license="0" rotation="0" originalsecret="f98be762b5"
   originalformat="jpg" views="11" media="photo">
   <owner nsid="99117185@N00" username="mbell1975" realname="" location="Washington,DC,USA"/>
   <title>Leinenkugel's Honey Weiss Beer</title>
   <visibility ispublic="1" isfriend="0" isfamily="0"/>
   <dates posted="1265239245" taken="2010-01-30 16:42:31" takengranularity="0"
    lastupdate="1265240167"/>
   <comments>0</comments>
   <tags>
     <tag id="3505925-4328326553-1049767" author="99117185@N00" raw="Leinenkugel's"
      machine_tag="0">leinenkugels</tag>
     <tag id="3505925-4328326553-13398" author="99117185@N00" raw="Honey"
      machine_tag="0">honey</tag>
   </tags>
   <location latitude="38.843785" longitude="-77.31122" accuracy="12" context="0"
      place_id="GgZmW42bAplSQFyy"    woeid="2355693">
     <neighbourhood place_id="GgZmW42bAplSQFyy" woeid="2355693">Ardmore</neighbourhood>
     <locality place_id="zzcnEYmbBZz15XaJ" woeid="2401348">Fairfax</locality>
     <county place_id="aYS88JuYA5n1ZPU7Hg" woeid="12590344">Fairfax City</county>
     <region place_id="22b1cJWbApjzghQy" woeid="2347605">Virginia</region>
     <country place_id="4KO02SibApitvSBieQ" woeid="23424977">United States</country>
   </location>
   <urls>
     <url type="photopage">http://www.flickr.com/photos/mbell1975/4328326553/</url>
   </urls>
  </photo>
</rsp>
```

This XML contains all the information about the photo, branched in different sub-groups:

- *specific information* tighly connected with the photo, as the format, date of upload, number of view, identification codes and so on..
- *comments*, whether there are
- *tags*, with information of the author and id
- *geographic location* with city, country, region and coordinate position whenever specified.
- *urls*, link of the photo page.. (if there are more than one, they will be appear here)

This two methods calls give us all the information needed, so now it is possible download and store all the data in the database.

3.4 Test and Train Set

The idea is to use imaged in Flickr famous brands collection and than detect which logos appear in those imagesWe are looking for two sets of images, one for training and the other for testing. For *Train Set* we mean a set of 3.2K images as random bunch of famousbrands, and for *Test Set*, 1K images with some additional requirements:

1. intersection on both sets should be empty
2. it must be ensured that every image of the test set contains (at least) one logo in the brands collection.
3. if possible, none oh the tags appearing in the train set should appear in the test set. This property will be checked examining the images' tags.

To know what an image contains, it's important analyze the tags, so we create two more table in the database, *famousbrands_tags* and *famousbrands_train_tags* for this scope. All the tags from Flickr images, were stored as a entry of the table, we parsed that value and split all the tags as singular word. The tables have the same structure, as follow:

```
[SQL command for build the famousbrands_tags table]
  create table famousbrands_tags
  {
    flickrid integer auto_increment,
    tag varchar(255) not null,
    primary key( flickrid, tag )
  }
```

This is thought for a very easy search by tag, to find it immediately without parse every time the big field "tags" of the main table. In fact, to respect to the first and the third point is necessary, after the creation of famousbrands_train and the relative _tags tables, to check not only the flickrid but every tags; in this way we can obtain two sets with images distinct each other and with distinct brands inside. This point is very important for our future tests.

To fill the Test table we use the following sql command:

```
[SQL command to fill in the test set]
  insert into famousbrands_test select * from famousbrands where photoid not in (
   select distinct photoid from famousbrands_tags where tag in (
    select tag from famousbrands_train_tags
     where tag <> 'logo' and tag <> 'sign' and tag <> 'brand' and tag <> ''))
      order by rand() limit 1000;
```

In the previous command there is a kind of filter to exclude some commons tags like *logo*, *sign* and *brand* and for an empty string, maybe obtained during the parsing of the original tags. This tags are so many popular, than it's impossible to extract 1K images from 15K of famousbrands with all the tags differents without filter them.
Another important operation to apply, is on the brands table. The logos are uploaded on the website, our resource, from all the world, so it is usual to encounter characters from different character-encoding scheme, and some time not readable. To resolve this problem you have to normalize all the brands name, obtaining a comparable string useful for a search.

parlare del problema di gestire e ricercare in modo immediato questi loghi: interfaccia php

Chapter 4

Features Extraction

To describe what is the meaning of *Feature Extraction* it is better digress a little bit. The goal of this project and, more in general, of a video retrieval works, is to detect something of unique, or more descriptive, in an image and than use this kind of information to compare it with other images. To do this it is necessary a comparison measure to reduce the number of variables under consideration. These operations require high computational cost when dealing with large data set, so the reducing dimensionality can effectively cut this cost. This process is called indeed *Dimension Reduction* and provides two different solutions:

1. *Feature Selection*, where are selected subset of the original features without transformation

2. *Feature Extraction*, where in function of some transformation the original features are replaced with some new ones. Here you have to consider the computational cost of the transformation.

This project is focused to the second point, and the idea is to extract the most relevant information from the input data, discarding redundancies and irrelevances. However, working with a large data set provides to do some operation more, as collection segmentation to catch a sub-selection of data as cluster, but all of these techniques are following the features extractor.
A different point of view to explain what does it means:

> **Feature-extracting question:**[32]
> *If we are allowed to use only one numerical feature to describe each data object, what should this feature be?*

There are many ways to extract features[27] more or less optimize for different goals. Some are based on the *Edges* or on the *Corners*, other on the *Curvature* or on the *Shapes* as Hough Transform, but we are focusing on the *Scale-invariant feature transform*[14] and on the *Speeded-Up Robust Features*[16].

4.1 SIFT vs SURF

In this section, there is a brief introduction to SIFT and SURF Descriptors, and in the end a short comparison of them with advantages and disadvantages related to this project. However to understand better the steps and the procedures of each method, we recommend to read the concerned references.[14][16]

Moreover we'll introduce some image filters with tests made and results obtained using the SURF Descriptors. There will be some examples and figures to understand better the proves.

In the end there is a description of all the steps needed to perform the matching and, of course, conclusions and observations about it.

4.1.1 SIFT

SIFT, acronym of *Scale-invariant feature transform*, is one of the most famous algorithm to detect and describe the local features in an image, invented by David Lowe in 1999[14]. It is a method for extracting distinctive invariant features from images that can be used to perform reliable matching between different view of an object or scene.

This feature descriptor maintains a good fame for its properties, because it is invariants to scaling, translation, rotation, affine transformation and partially to illumination changes. The goal of these techniques is to convert an image in a set of vectors, and these vectors have the characteristic to describe enough the image. For us, it will be possible work directly to the vectors and make all the operations with them.

A typical image of size 500x500 pixels will give rise of to about 2000 stable features, and the quantity of features is particularly important for object recognition. For image matching and recognition, Lowe used the

Euclidean distance to compare the vectors of the image with all the other stored in the database. The large number of vectors identified, could become a big problem during the comparison and matching, furthermore a SIFT descriptor is structured by 128 dimensions or, in other words, 128 numbers, twice of SURF vector.

The good of the SIFT is the great number of features that can extract, but the problem is the speed for all the computational steps. Many researches have been done in literature about a comparison of the different Features Extraction Algorithms[26] and in each of them there are not a really winner. Each algorithm has good and bad things, for this project the main reason to choose the SURF Descriptor was the fast computation, since the quality of this descriptor is very similar to SIFT.

4.1.2 SURF

SURF, Speeded-Up robust Features, is another image detector and descriptor inspired by SIFT, was first presented by Herbert Bay in the ECCV 2006 conference in Graz, Austria[15]. SURF approximates or even outperforms previously proposed schemes with respect to repeatability, distinctiveness, and robustness, yet can be computed and compared much faster. This is achieved by

- Relying on *integral images* for image convolutions
- Building on the strengths of the leading existing detectors and descriptors (using an *Hessian matrix-based* measure for the detector, and a *distribution-based* descriptor)
- Simplifying this methods to the essential

In this project it is used the OpenSURF instead SURF, it is an open source implementation (C++, Linux) with all the documentation and references necessaries, because the original version is still to be closed source.

The dimension of the descriptor has a direct impact to the storing and to the time taken of each comparison. The SIFT descriptor uses 128 dimensions, whereas the SURF uses only 64 dimensions. However, lower dimensional feature vectors are in general less distinctive than their high-dimensional counterparts.

Explain works and steps of the SURF Descriptors is not the job of this document, for that we refer to the bibliography, but we can discuss some points of comparison between the two descriptors[25]. Both SIFT and SURF belong to the family of *scale invariant feature detectors* so, they trying to analyze the input image at different resolutions in order to repeatably find characteristic independently of their actual size. To this end, they use multiscale detection operation called *scale space representation* of an image. In a second step detected features are assigned a rotation invariant-descriptor computed from the surrounding pixel neighborhood.

SURF builds on the concepts of SIRF but introduces more radical approximations in order to speed up the detection process. Due to the use of integral images the complexity of SURF is greatly reduced, so SURF often achieves superior performance than its predecessor. Another different is that SURF uses the determinant of the Hessian for feature detection in scale space instead of the Laplacian operator. Nevertheless for the goal of this work, the decision to use the SURF instead the SIFT is due to the greater computational speed without significant loss of information.

For the brands collection, discussed in the the Chapter 4, all the images of the logos was saved in GIF format from as are stored in the website, but the OpenSURF code required an input image in JPG. For that it was necessary to convert all the brands. Below there are some example of SURF detected.

Name	parmalogo	yahoo	juventus	NYyahoo
Type	logo	logo	photo	photo
Computational Time	0.114s	0.083s	0.487s	0.381s
Number Of Vectors	121	72	567	451
Resolution	200 x 200	200 x 200	500 x 495	500 x 333

Table 4.1. Example of SURF computations information

In the table 4.1 there are a few examples to see the computational time of SURF Descriptor for logos and images. In the figure 4.1 and 4.2 there are the images with plotted surfs.

Code Updates

To make the code more suitable for the needs of this project, it was necessary to make some change to the code. First of all the type of input,

Figure 4.1. Examples of SURF computations images

instead support only one image per time, an option has been added to read each values from standard input and compute all the SURF vectors immediately writing on the standard output. A lot of scripts in this project work directly with *standard input* and *output*, because it is much more convenient to manage input and output data and moreover it is possible to work in a concatenated way.

However the most interesting change concerns the introduction of a bounding box. The full name is *Logo Bounding Box* and it has to describe all the rectangular box where is contained the logo. In the OpenSURF code, given all detected SURF points, the maximum and the minimum values are kept, and this data is added to the descriptor information. So far, a vector looks like:

```
float x, y;
float bboxX, bboxY, bboxWidth, bboxHeight;
float scale;
float orientation;
float descriptor[ 64 ];
```

All the code in this project support the bounding box, see figure 4.3, to

Figure 4.2. Examples of SURF computations images

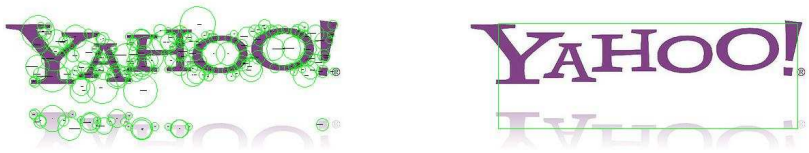

Figure 4.3. An example of Bounding Box

be more precise, there is the possibility to work with or without it, depends of the kind of test. One of the thing that can help to the detection of the logo, is in fact the *origin recognition*. The origin is the first point of the logo, indicates by the bounding box as origin. It is used during the matching of a brand and an image, trying to identify the origin of the logo by the matched points of the other image. This topic is discuss with more details in the Section 4.2.3.

4.2 Matching Points

Once obtained the features descriptors of an input image, using Open SURF in our case, you must find a way to compare them with all the ones in the brands collection. With the SURF, a descriptor is constituted with 64 dimensions, to be more precise by 64 floating point values. To com-

pare vectors as these, there are different ways, we made some comparison between: Euclidean Distance, Correlation and Cosine Similarity.

The *Euclidean Distance* is based on a repetition of the Pythagorean theorem is one of the most commons, used by Lowe with his SIFT. En example below, of this computation:

Given two points $u = (x_1, y_1, z_1)$ and $v = (x_2, y_2, z_2)$
the Euclidian Distance is

$$\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2}$$

The second method tested is the *Correlation*, it's commonly used in Text Mining to compare two vectors of n dimensions. It is obtained by dividing the covariance of the two variables by the product of their standard deviations. The formula is this:

$$cos(\theta) = \frac{\mathbf{uv}}{\|\mathbf{u}\|\|\mathbf{v}\|}$$

The last one is the *Cosine Similarity*, this measures the cosine of the angle between the two vectors.

$$P_{x,y} = \frac{cov(x,y)}{\sigma_x \sigma_y}$$

4.2.1 Matching Technique Comparison

Write here how we compare the matching techniques: Correlation, Cosine Similarity and Euclidean Distance.
With Graphs and numerical values.

4.2.2 Correlation

Given two input vectors, computing the correlation, the output will be a value that represents their similarity. This value has a range from -1 to 1, where -1 indicates that the vectors are opposite, and 1 that the vectors are the same. All the values in the the middle can be valuated by the user, but in this project we are looking for the similarity, so it is necessary to consider only the positive values greater than a certain threshold. To make sure this comparison method is working well, in our case, we tested it in many situations, with a final manually evaluation. Have been implemented

a few applications to do this, described in more details in the Section ??.

The idea is that the trend followed by the values of two different SURF points is similar if the points correspond to the same background shape. See examples of matching and non matching points in the following images, which show two matching points with a correlation threshold of 0.985:

Figure 4.4. Correlation with threshold of 0.985

Figure 4.5. Correlation with threshold of 0.95

Once computed all the SURF points of both images, they are compared using the Correlation and are discards all the matching less than a certain threshold. We made different tests to define the threshold, of course elevated values are more discriminant but more precise. In the figures 4.4 and 4.5 there are a couple of examples with two different elevated thresholds.

4.2.3 Bounding Box

In this section we'll discuss with more details the Bounding Box. The idea is, once detected the correct logo and the logo position in the image, to select the logo region by plotting its bounding box. We try to identify the origin point through a voting: from each feature extracted by SURF Descriptor.

With all these information, the proposed method will be:

- Given a Target and a Source image, see how many points are highly correlated
- For each highly correlated pair of points, we will try to transpose the origin of the Bounding Box of one point in the other image.
- This procedure will give us a guessed bounding box, that will tell where the logo appers in the Target image.

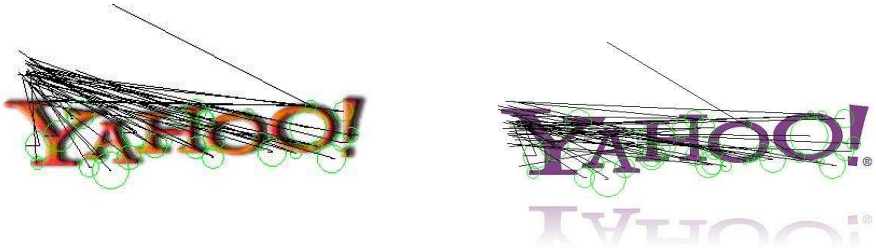

Figure 4.6. Transposed bounding box of the matching points (Threshold 0.95) without BBox plotted

In the figures 4.6 and 4.7 there are a selection of candidates origins from all the matched points. As it is possible to see, not all of those are correct, but with a ranking of the proposed points we can select the best candidate as origin. An implementation should be merge the value of the correlation in this ranking, in other words, use a kind of weight-ranking by correlation to give more or less weight to each vote. The matched points with a big value of correlation are more creditable than the others because the vectors have to be more close.

In the code of OpenSURF, during the computation of the descriptors, we compute and store more information4.8. As we just said, the code compute the bounding box simply considering the extracted points with smallest and biggest value of x and y so it can find the boundaries. Of course the box could not cover the entire logo shape, but it is what SURF can detect and it is everything we need.

Each logo SURF vector contains the origin and the dimension of the bounding box, so, once obtain the candidates logos from the input image, it is possible to start a voting for the origin. In this case, keeping the vectors that vote in a common area we can filter all the other, discarding the false positive candidates points.

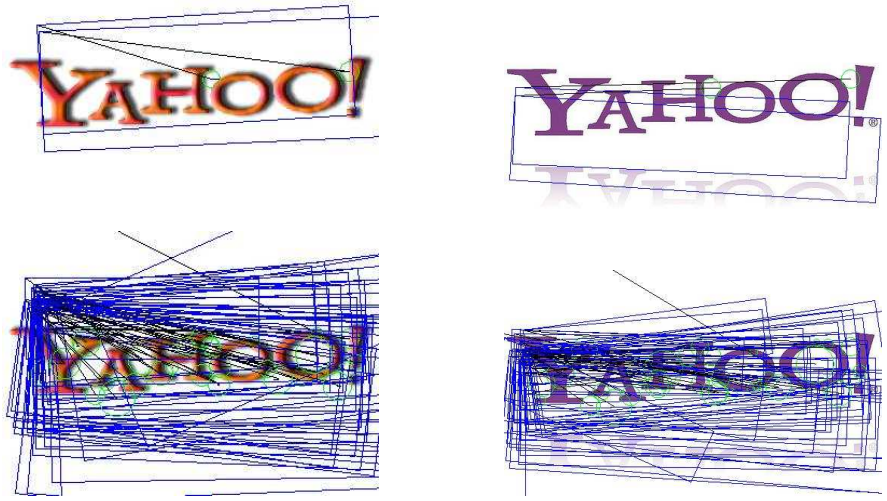

Figure 4.7. Transposed bounding box of the matching points (Threshold 0.985 and 0.95) with voting of all the candidates features.

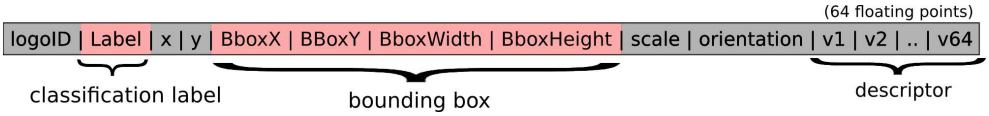

Figure 4.8. Structure of SURF Vectors, the colored parts are added, for this project, to store more information.

4.3 Image Filtering

The following section presents ways to improve the matching ratio by applying transformations both to target and source images in order to isolate matching points and increase correlation. This matter appeared during the matching tests, there were a few images with the same logo in more than one positions, but with different backgrounds, like the figure 4.9. After a comparison with the correct brand, the matching results were very different.

The following examples concern only the correlation matching, without any classification, filtering or segmentation. So, all the surf vectors from the input image were compared with all the vectors of the logos, and are selected the vectors with the bigger match.
The different value of the threshold is important to understand what is discarded and what is not, as the fig4.10. In others tests there were applied some image filter to check the performance change, in the fig4.11 show the

Figure 4.9. The same logo appears twice but with inverted background) without BBox plotted

results with the image converted in gray-scale, in the fig4.12 was applied the blur filter, too.

Figure 4.10. Correlation with threshold equal to 0.80 (with 0.90 there was not common points, with 0.85 not so many)

Figure 4.11. Correlation with threshold equal to 0.80 (with 0.90 there was not common points, with 0.85 not so many)

Figure 4.12. Correlation with threshold equal to 0.80 (with 0.90 there was not common points, with 0.85 not so many)

4.4 Problems and Conclusions

This point of the project was very useful to figure out how the SURF Vectors works, and how it is possible to obtain the better performance without loose information.
We have got different kind of problem, and some filter clarified the limit to use Correlation with the SURF: applying a invert filter to the image, in other words, inverting all the value (in a b/n image) the points extracted before the filter are totally different.

In the image 4.13 this case is very clear: there are two possible matching of the logo, one in the top of the image with dark color and blank back-

Figure 4.13. Correlation with the b/n image: it is detect only the logos located on the top and the other does not have common points.

ground, and the other in the bottom with blank color on dark background. The first one is detected successfully: so the correlation finds similar the vectors of the logo and of that part of image, but the second one does not have a single common point. Applying the filter, explained below, on the image we'll find an inverted situation: the first area does not have common points and the second one is well detected.

Figure 4.14. Correlation with the same image but with inverted values: the logo detected is changed.

If the matching between the interest points of the image and the logos points does not have an high value of correlation, it means that the vectors are not so similar. So or we could change the metric technique or we could change the descriptor algorithm. We tried both the solutions, but with the Euclidean Distance or using the SIFT Descriptor the problem

are the same. Another possible solution is to apply the invert filter to all the logos collection to discard the possibility to loose information in this way, but the collection increases twofold and the computation times as well.

However these are tests to understand with what we are working, the solutions of these problems are not the goal of these project but it's important to know the limits of the used algorithms and techniques.

Matching vector-by-vector the source image and the logos, in the pairs of points with an high value of correlation there are the correct ones, but it is not possible compare with correlation (or Euclidean distance) all the vectors. So the main conclusion is the requirement to process the surf vectors before do the direct matching, or, to try to find a good way to segment all the logos surf in similarity groups. In this case we have to compare not each extracted vector of the input image to all the others, but only with the representative vector of each group, because if all the points in the same group are similar it is possible define a representative one that it is similar with all the other. We can obtain better performance in computational time and in results quality.

Chapter 5

Collection Segmentation

The Collection Segmentation is a key point of this project, because, at this step, all the logos SURF descriptors are computed, and now we have a huge number of vectors. Obviously we can't compare each point from the input image to all the logos points, the computation time is exaggerate and more, using only one matching technique should be a lot of false positive. For these reasons it is necessary to segment all the logos SURF vectors and classified them by some common feature.

In this projects were tested a lot of different techniques for the segmentation. The main idea is to split all the vectors in different groups, and obtain for each group a representative vector. So, to compute the candidates of an image, it only necessary compare its SURF points with these representative ones.

What is a *representative vector*? Think about the current goal, it is to reduce the number of comparison between the points extracted from the input image and all the stored logos points. To do this we segment the logos points collection splitting the not-similar vectors in the different groups, so each group contains only points similar to each other. Computing a vector that represents a group it means to obtain a vector that is similar to all the other in that group. With this one we reduce the number of comparison for each image point to the number of groups. How to create the representative vectors and how many groups generate depends of the used techniques.

The first technique discuss in this chapter it's the *Balanced Decision Tree* where, for each level of the tree and in each of the two nodes, it will be selected only one dimension of all the 64 of the SURF vector, and by this value the descriptors are split. Another method is to use a *K-Means Clus-*

ter, where all the closest points are grouped and one point for each group is computed as centroid to represent all the other. Other ways should be the *Random Vectors* or the *Semi Random Vectors*. For these two methods a set of vectors are created with the same structure of the SURF one, but in a random and in a semi-random mode.

Once segmented all the vectors, we obtain a *model* that contains all the representative vectors used to segment the collection and a *classified* file that contains simply all the logos vectors classified. In all the classified vectors there is one value more in the vectors that contain the *label* (or *signature*) of the relative group. To compute the candidates of an image, we need this model and the relative logos classified.

Each vector of the input image should be matched to each representative vectors and the best match is selected. This selection return the label of the group with the best similarity and to obtain immediately all the logos points inside this group, it is necessary a Random Access to the classified file. To do this it will be created an index to store for each label: the file location, the starting byte and the length of the data included. We'll see in details this step in the next chapter.

5.1 Variance-Balanced Decision Tree

The object is to create a balanced decision tree, where given a set of (SURF) interest points the decision tree produces a balanced set of leaves. Each leave node will contain approximately the same number of interest points.

The tree below 5.1 illustrates how a set of interest points is split in two equal subset, based on the median value that can be computed for a single dimension that is selected from the SURF feature vector.

How to select a dimension at a particular point in the tree? There are several solutions:

1. *At random*: the algorithm selects at random a dimension, Eg a dimension $D = [1..64]$.

2. *Max variance*: the algorithm computes the variance of the feature values at each of the 64 dimensions, and produces a ranked list of

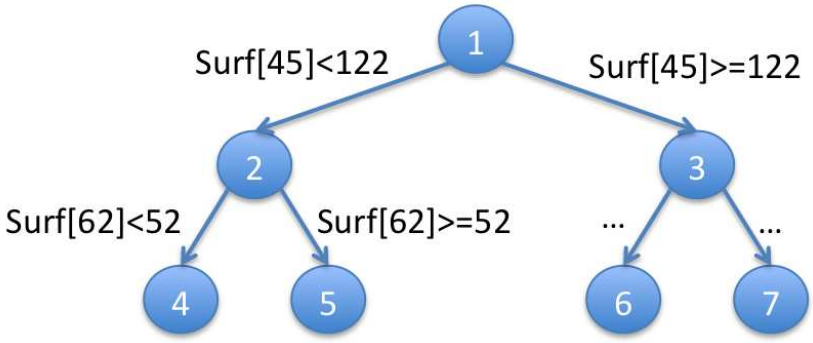

Figure 5.1. In each node the collection is split in two equal subset. With SURF[45] it means that each vector during the segmentation has to check the value in the dimension 45, and if it is bigger go to the right node otherwise to the left one.

dimensions D where the dimensions are ordered in descending variance. When constructing the tree, at depth i, the $i-th$ element of the list of dimensions D is selected. Eg. nodes 2 and 4, corresponding to depth 2 would both use the same dimension for splitting the data, but most likely use a different threshold. The advantage of this approach would be that the decision to split a set of interest points is based on those dimensions that are most discriminating.

3. *Local max variance*: In this variant, the dimension for splitting is selected based on the maximum variance, which is re-computed for every node, with the corresponding set of key-points.

The current proposal is to go with the third variant: the advantage would be that in each node the decision to split is based on the dimension that has the biggest value of variance at that point, in other word, the most discriminating dimension for that level of the tree.

5.1.1 Implementation

Given the SURF vectors for all logos to be structured in the tree, the following steps are needed:

1. Compute a vector with the same dimensionality as the SURF vectors, with the Variance of each dimensionality. To do this, all the collection should be traversed.

2. Take the dimension with maximum variance and select it as a decision dimensionality.

3. Calculate the median for the selected dimensionality and split the collection in two parts, taking the median as a threshold.

So, with the previous steps we obtain a pair of values: the selected dimensionality and the relative value where check and split the collection. For each level of tree these steps are carry out to obtain for each node the pair of values. When a surf vector is processed is required to keep a signature of the chooses in each steps, the common way to do this is to sign with a 0 the left choise and with the 1 the right one. So, for each level there will be a binary value, and in the end of the tree there will be the signature or *leafID*. Look the figure 5.2 where the dimensionality choise procedure is illustrate in steps.

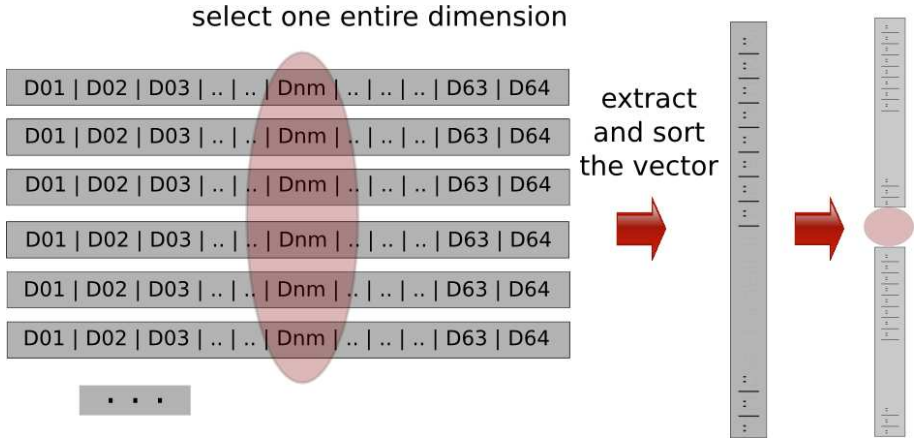

Figure 5.2. Tree creation: compute the variance for each dimension of the collection and select the dimension with the bigger value. Once obtained this new vector, sort it in ascending mode and take the value in the middle: this is the *median*. This value it will be use to split the collection in the current node.

Following there will be the pseudo-code to explain the code.

```
function createDecision ( Collection c, DecisionTreeNode t )
  Vector v = calculateVariances( c );
  double max_variance_index = getMaxIndex( v );
  double median = getMedian( c, max_variance_index );

  t->dimension = max_variance_index;
  t->median = median;
```

```
t->left = new DecisionTreeNode();
t->right = new DecisionTreeNode();

<c1, c2> = splitCollection( c, max_variance_index, median );
createDecision( c1, t->left );
createDecision( c2, t->right );
```

Once created the tree, it should be serialized and stored as a model[1]. This is another convenience to using a tree to store the information. A tree is easily store as an array preserving the random access to a node: if you are looking for the children of the node in position K, you simply have to keep the values in the positions $2 * K$ and $2 * K + 1$ for the left and the right child. In the figure 5.3

Node	1	2	3	4	5	6	7	8
Dimension	45	62	32	8	5	16	62	4
Median	122	52	166	44	110	90	103	89

Figure 5.3. (Model Tree) It's a two-dimensional vector with a pair of value for each node: the dimension index and the dimension value to split.

The height of the tree set the length of the signature (or leafID), than in function of the collection it is possible to segment it with different distribution. Our goal is not to do an hard segmentation, but to keep a large number of points in each leaf and in succession to do a ranking of these vectors. So we are sure to not loose information: it is better to keep the *false positive*, and to discard them later in the ranking step.

5.1.2 Experiments

The experiments describe in the following sub-section are built for the project and not only for this thesis, so the following consideration is taken from the documentation done by the entire working team. The initial tree was create with a set of training images, and with the properties shown in Table 5.1.

[1] http://webdocs.cs.ualb erta.ca/ holte/T26/tree-as-array.html

Property	Value
Number of training images	25000
Largest dimension of the image	500px
Number of SURF descriptors per image	419
Number of SURF descriptors extracted	>10M, of which 2.5M used as training data
Size of the decision tree model	$2^15 - 1 = 32767$ tuples (dimension, median)

Table 5.1. These are the tree implemented by a set of training images.

To define the **height** of the tree we made the following *hypothesis*:
The stability of the tree at a given depth depends on the number of SURF features that have been used to train the decision tree. When enough features are used, the difference between two models will be small.

To test this hypothesis we built a set of pairs of decision trees using the same number of features, but the features should come from different set of images. The "different" between a pair of decision trees will decrease when the number of features used will increase. We're looking for the point where the δ difference between two pairs of decision trees comes below a predefined threshold, with the number of features used being doubled at every step. We need a good metric to measure the difference between two variance-balanced decision trees.
Intuition: As the construction of the tree is based on variance, the dimensions selected at the higher level (low depth) should stay more stable than the dimensions selected deeper in the tree.

Experimental Design

The stability of a Variance-Balanced Decision Tree depends on two parameters:

1. The *number of feature vector* used to train the decision tree
2. The *depth* of the decision tree

For this experiment, we have retrieve 25000 images at random from flickr, the data are shown in the table 5.1.

We estimate the stability of the decision trees by repeatedly doing pairwise similarity comparisons. The **Level Stability Detection metric (LSD)** is adopted to measure how stable the decisions are at each

node in the tree. The LSD metric is explained in detail below. For the experiment we are interested in comparing the stability of the Variance-Balanced Decision Trees given a variable number of feature vectors to train with, but also to investigate the stability at various depths in the tree. For the evaluation we'll compare the stability at depths *5*, *10*, and *15*.

At depth 15, we'll have a tree with 32K leave nodes. As we want to repeatedly measure the stability of the tree we have split the 10M SURF descriptors ($400 * 25K$ images) in 4 equals sets, each containing 2.5M feature vectors. This allows us to do a 6-fold pairwise comparison of the stability. We selected at random X feature vectors from a set to construct a tree, with X ranging from $200K - 2M$ feature vectors. In addition, we trained a decision tree for each set, using all 2.5M feature vectors.

Level Stability Detection metric

The intuitions is that the stable trees will make similar decisions at the same node in a tree. LSD estimates the stability of two trees, by inspecting the agreement in selected dimensions at each level in the tree. The LSD score ranges from $[0 - level]$ where two similar trees will have a score equal to the number of the levels (depth of the tree).

To be able to compare the stability of trees across levels, we can normalize the LSD (nLSD) scores through dividing the score by the number of levels.

5.2 Random Vectors

This techniques provides to split the input vectors computing a *signature* using a defined number of random vectors. The idea is to split the collection by random vectors that, generated in random mode, could segment it in a distribute mode. These are structured as the SURF ones, but the values in each dimension are: first generated in a randomly way with floating values in a range -1 to 1, second normalized.

This method works quite different from the other, to be more precise each random vectors is not use directly to split, but there will be compute a signature after a comparison of all the random vectors. Let's see the following pseudo-code:

```
ImageSURF = read( input_image.surf )
RandomVectors = read( randomVectors.16.class )
signature = ""

for ( current_surf in ImageSURF ) {
  for ( current_rv in RandomVectors ) {
    correlation = pearsonCorrelation( current_surf, current_rv )
    if ( correlation > 0 )
      signature.append( "1" )
    else
      signature.append( "0" )
  }
  store.add( current_surf, signature )
}
```

How is possible to see, each surf descriptor of the image is compared with all the random vectors, and for each comparison it's assign a "1" or a "0". So, if we use 8 random vectors, we obtain a signature of length 8, and 2^8 possibility. We tested different length of signature: 8 (256 possibility), 12 (4K), 15 (32K), 16 (65K), 32 (2^{32}), 64 (2^{64}) and 128 (2^{128}). Obviously with 64 and 128 it will be a huge number of vectors, and the it is outside of the idea of segmentation because the SURF vectors will be not split so well.

So, once computed the random vectors we use them as model, and now we have to segment the logos points collection as it's shown in 5.4 to obtain the classified logos. Apply the algorithm discuss before, all the SURFs obtain a relative signature, that is the label of this group, and now you have to segment by signature all the vectors. Furthemore this is a large file with huge dimension (12G), and for this reason it necessary to build an index that we can store it entirely in memory and we can access directly only to the required data of the classified file. To get the logos

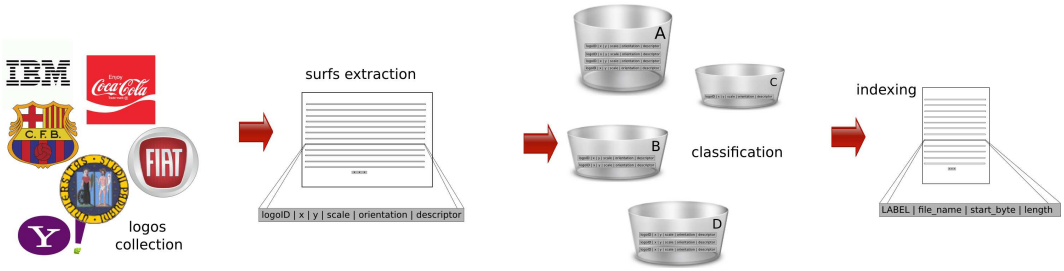

Figure 5.4. Random vectors segmentation procedure.

candidates we make the following steps:

- Get the input image and compute the surf vectors
- For each vector, compute the relative signature
- For each signature, retrieve all the points candidates
- Rank with these points (retrieve the image with more point matched, ..)

5.2.1 Semi Random Vectors

The Semi Random Vectors method works as the Random Vectors, the only difference it's how to create them. In this case are not so random: the dimensions of the descriptors are analyzed, and split in three balanced subgroup. The idea is similar to the Decision Binary Tree, but the values are split in a much more simply way.

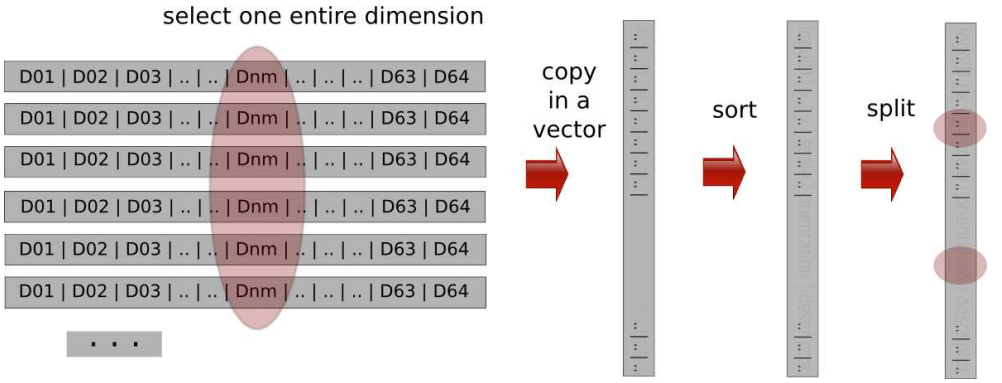

Figure 5.5. Semi random vectors creation.

Let's see the image 5.5. For each dimension it copy the values in a new array and sort it to have all the avlues ina descending order. In the end this array is split in three equal parts, storing the two values used for the split. So, for each dimension, we keep these values and we use them to create the semi-randomvectors.

5.2.2 Signatures Distribution

For the Random and Semi-Random Vectors the distribution is not balanced as for the Variance-Balanced Decision Tree, so it is possible to find some picks that is some signature very commons that represent a lot of

points. These points are taken as noise and a signature with a huge number of vectors is not used in the ranking.

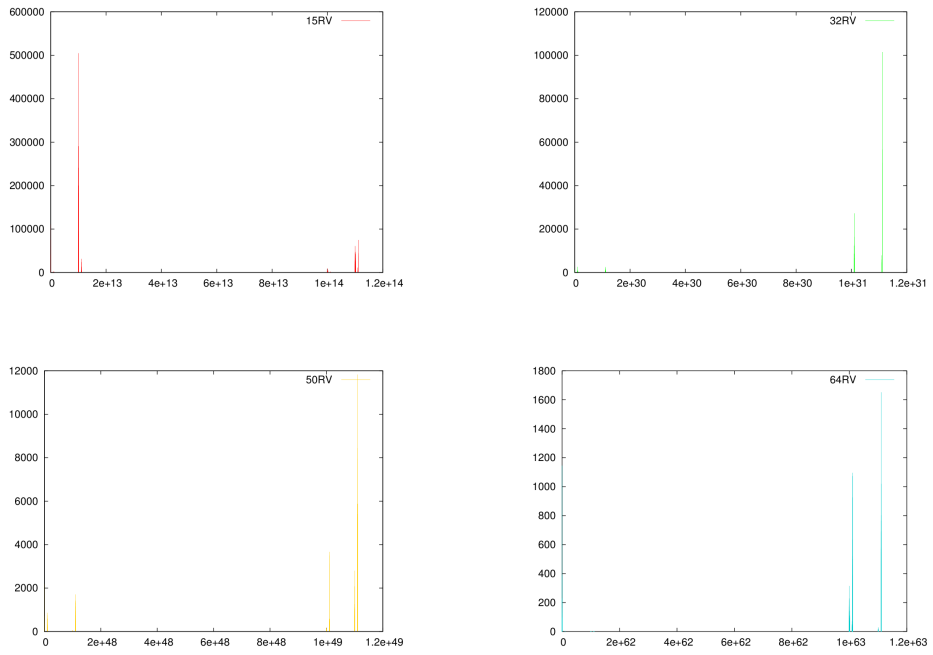

Figure 5.6. These graphs explain the distribution of the signatures after the classification of *brands.surf*. The first one describes about the random vectors with length 15, the other of lengths: 32, 50 and 64. It's possible to see some commons picks in all the graphs.

To tests the Random Vectors to define a good length of the signature (analogous for Semi-Random) we create different number of vectors. In this section we discuss the distribution of signature with lengths 15, 32, 50 and 64. The classified collection is the full one, called *brands.surf* with 175K of logos and about 17M of surf points. In the figure 5.6 there are the distribution graphs in function of 17.5 millions of vectors. The goal is to have a constants values, it should be describe as an horizontal line in the graph. Unfortunately it is not so plane and there are some picks: this means that there are some points very common with the same signature. All these ones are discarded during the candidates selection.

In the table 5.2 there is a summary of the four different Random Vectors here reported. In this table we toke in exam the RV20 and RV40, because

they are more similar to the version 15 and 32 and the comparison is more interesting.

Number of RVs	RV15	RV20	RV32	RV40
# Total sgn	32768	1048576	4294967296	1099511627776
# Empty sgn	23224	935263	4293990258	1099510699539
# Not-Empty sgn	9544	113313	977038	928237
(# Empty sgn)/Total	0.708740	0.891936	0.999773	0.999999
(# Not-Empty sgn)/Total	0.291260	0.108064	0.000227	0.000001

Table 5.2. This table contains information for each different number of Random Vectors: number of total signatures, empty and used ones and proportions of efficiency.

The distribution signature/vectors is not plane as the expectation, but it is interesting see how much it changes when increase the number of RV. For the RV15 there are 32K of total signatures and segmenting the *brands.surf* with 17M of vectors there are used only 9K with more then 70% empties leaves. Using the RV20 the are more than ten times of vectors used, but in proportion with the total number of vectors there are even 89% empties signatures. With the RV32 ad RV40 the situation is worst, indeed getting the signatures from an input image, should be the possibility to find signature without any brands vectors, so empty ones not useful for retrieve the candidates.

Analyzing these values look clear to use a little number of vectors: RV15 is well. We make the same test with RV14 with similar results, and finally we did the experiments with these two sets of vectors. During the experiments, specially for the Random Vectors (not for the Semi-Random) appears necessary to generate a set of RV15 so to test the *random factor* and to see if the results are stable or not. In Section **??** there will be more details.

5.3 Other Techniques

To segment a large collection there are many ways, this chapter toke in exam the Variance-Balanced Decision Tree and the (Semi-)Random Vectors techniques, but during this work there were implemented and tested more methods. All the techniques have the aim to segment the large data set composed of logos vectors, so the possibility to do this step are many,

we had only analyze some of them to choose the good one for our project. The other techniques are K-Means Clustering, a very common method to split huge collection, and Spline, where we compare the vectors shown as functions. However there is a brief discuss about the K-Means only, and not in-depth because it was not developed by myself, but I think it's important to give a general knowledge about it to have another comparison as segmentation technique.

5.3.1 K-Means Clustering

K-Means Clustering is a method of *cluster analysis*, it is very popular for the partition problem where given a set of n data points the aim is to split the data collection in k groups, finding k points centers. In our situation it had to partition the brands collection in k groups of similarity, it means find k vectors, called centers or centroids, that can minimize the mean squared distance from each data vector to its nearest center. To compute these centers the method makes a define number of iterations, where the self centroids could change position to obtain a better distribution of the points, so at each loop the groups are more balanced.

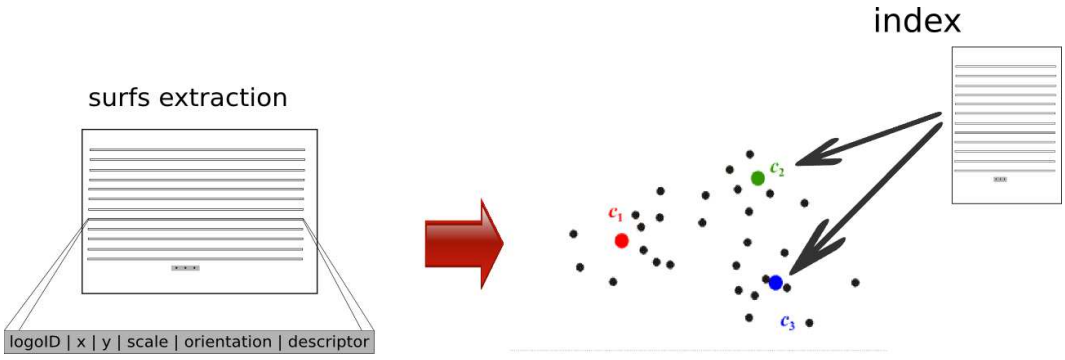

Figure 5.7. KMeans segmentation procedure: acquiring of SURF vectors from the collection, detecting the centroids and indexing the centers.

The procedure is explain in the figure 5.7 where given the large collection of SURF vectors, it find the centroids after a number of iteration, where it compares each times the distances from each vectors to the centroids. Once computed the centroids vectors, they become the *kmeans model*, that is a simply list of vectors as a *signature* or a *leafID* for a Decision Tree. In this case the procedure is similar to the Random Vectors, instead of through the tree to obtain the leafID, the SURF vectors has

only to compute its distance of all the centroids and select the most close.

To compute the centers, we used this time the Flickr Collection, to train the model with more points, not only relative to a logo. This Flickr Collection is constitute with a million of random Flickr image, so it doesn't care if they contain or not logos, because they are choose randomly. Once computed the centroids, the *brands.surf* collection is segmented, so for each SURF vector there is a corresponding centroid. As the other techniques, to have a direct access to each segment (centroid group) we made an index to the centers and to all the relative logos vectors.

Chapter 6

Classification and Ranking

During the Segmentation steps, it was created a *model* and with this it was segmented all the logos collection. After this, we obtain the classes, or groups, that contain the logos split by similarity. This mean that in each group there are the SURF vectors of the logos that are similar for the comparison techniques used in the segmentation. The next step concern about the input image: we have to extract from it all the SURF vectors, and define which is the best group for each of them. With best group we intend the group where there are the elements more similar to the current SURF vector.

Remember that: with *segmentation* we means to split a collection in undefined segments by some commons properties, instead with *classification* we mean to split a collection in classes yet built. For each image vector we have to obtain a class of logos vectors, so we have to classify all the SURF image vectors.

Once obtain these classes, we have a large list of logos points that are similar to the image ones for some properties: these vectors are called *vectors candidates*. The next ad final step is to work on this list, in other words, to apply some ranking to these candidates to understand which are the points more similar and, in the end, which are the images with more similar commons points.

This chapter describes which are the steps to the classification for each techniques sow in the segmentation, and what it's mean with ranking of the candidates, how it work and which are the new techniques.

6.1 Classification Methodology

Classify, as we said, means to define which is the best group o section for an object. In this case, which is the best group or cluster for an input vector. *Ho to do this?* Depends about the technique used to the segmentation: obviously to split the image vectors into the groups created from the logos collection the methodology has to be the same, or similar. In this section there are described the classification steps for the Variance-Balanced Decision Tree and for the (Semi-)Random Vectors, two different procedures to give a better idea of the aim of this section.

Nevertheless, only for a general knowledge, with the other techniques as K-Means or Spline the procedures are pretty similar to the Random Vectors, the idea is to use the centroids, for example, to compute the signature to retrieve the candidates, instead to use directly the random vectors.

6.1.1 Variance-Balanced Decision Tree

The Variance-Balanced Decision Tree technique discussed in the Section 5.1 portended to create a model, defined before an height, with the Flickr Collection vectors. With the *tree model* we were able to pass through it with all the logos vectors, so obtaining a *leafID* for each SURF points of the brands. Looking the figure 6.1 there are in the end of the leaves a list of vectors, these ones are the logos points located in the relative leafID, so each of these bins of vectors is a class of candidates.

Given the input image SURF points, the procedure is analogous to the segmentation, because it has to go through the model tree to catch its leafID or, more in general, its signature. To do this the steps are the same, in each node there are stored two value: the number of the dimension and the split value. So, to each node the SURF vector compare its value of the relative dimension and if is bigger it has to continue to the right side, otherwise on the left one. This procedure has to be done until the last level, where there are the leaves.

So from a list of image SURF vectors, we obtain a signature for each of them and we have to take all the candidates of these signature, that is all the points in the lists with the leafID computed from the SURF input points. Look the figure 6.2 to see a graphic explanation.

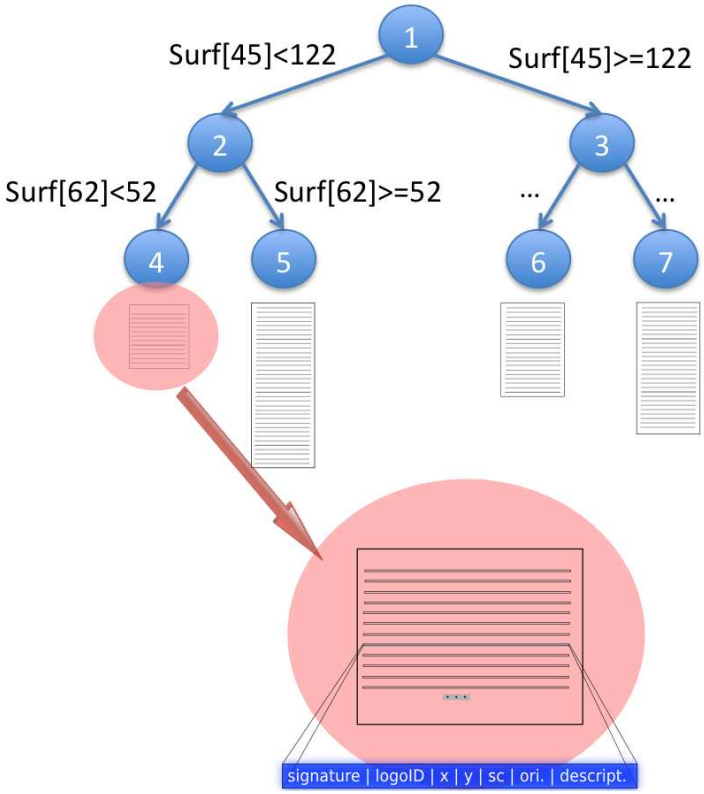

Figure 6.1. This is a model tree through that are segmented the logos brands. In the leaves there are the lists of the points with the relative leafID.

6.1.2 (Semi-)Random Vectors

With Random Vectors the procedure to classify the input SURF vectors is more simple. For this technique the model is just a file with the random vectors. So, obviously using the same model used in the segmentation, we have to compute the signature for the input image points. Looking the figure 6.3, for each input vectors it has to compute the signature simply with matching between the input point and all the random ones, storing a 1 or a 0 in function of the value of matching metric, if it is positive or negative. Once we give the signature we take all the logos points that have been stored with the same signature during the segmentation, and save them as candidates attending the ranking.

To classify the input SURF with the Semi-Random Vectors the procedure is exactly the same. As we said the difference is only in the construction of these vectors.

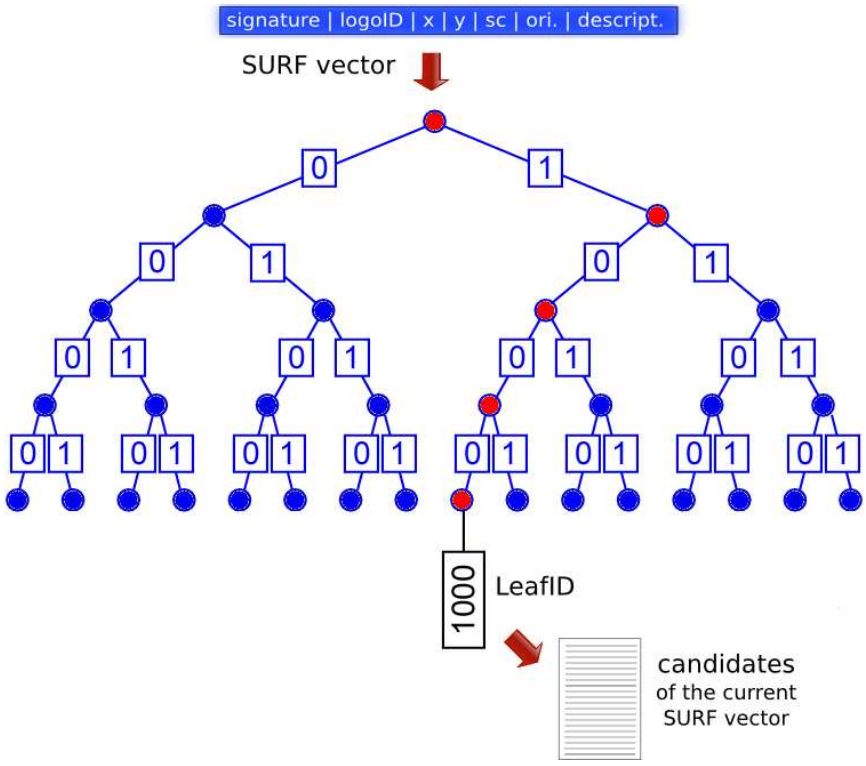

Figure 6.2. This is the procedure to classify a SURF vector extracted from an input image. Through a tree model it keeps a leafID (or signature) and with this, it retrieves all the logos in the same leaf as candidates.

6.1.3 Other Techniques

The other techniques describes in the Section 5.3 have a similar methodology for the classification. The aim is the same: use the model and the relative segmented logos to classify the input image vectors.

Using the K-Means Clustering described in the Section 5.3.1, the model is a list of the centroids computed after the application of the K-Means technique to the logos collection. Once segmented the logos vectors, it computes the centroids, that are simply the representative vector for each group. So, as illustrated in the figure 6.4, we match each SURF points extracted from the input image, to all the vectors in the kmeans model, in other words to all the centroids, keeping the most similar one with the highest value of match.

As the other technique, all the candidates logos points of the obtained

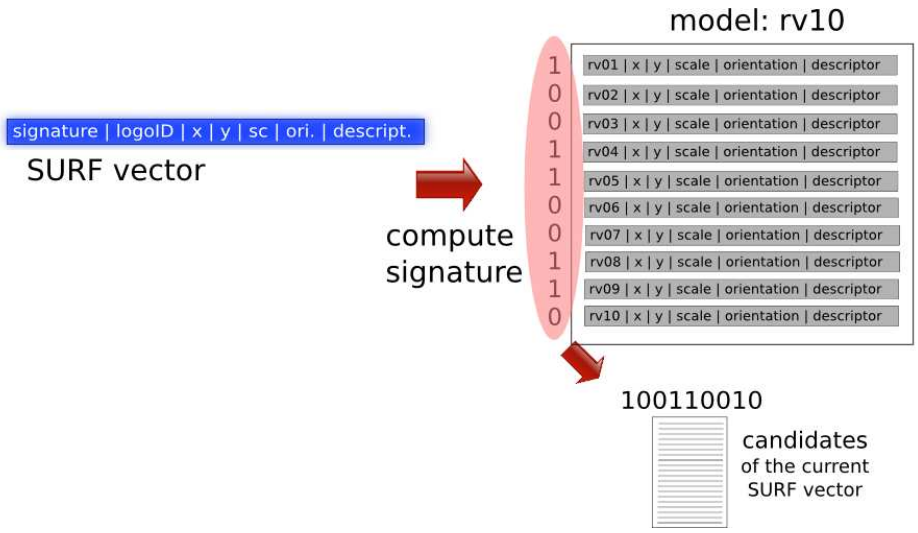

Figure 6.3. Given a SURF vector, compare it with the model of Random Vectors to obtain a signature: with a matching between the input vector and a random vector, if the value is positive return 1 otherwise 0, and this for each random vectors. After that, you have to retrieve all the logos segmented with the same signature.

centroid are taken as candidates. The next step is commonly with all the technique: is the ranking of the candidates.

6.2 Ranking of the Candidates

At this moment, we have an indefinite number of candidates points retrieved in the image classification. The goal is that all these vectors have to be analyzed, with the aim of discarding the *false positive* keeping only the correct candidates. *How we can achieve this?*

It's required some how to order these points: the last ones are going to be discarded, but to understand which are good and which not, we need to rank all of them. The goal is to understand which vectors, of the candidates, have a really similarity with the image ones.

There are many ways to do this, the easiest one is simply to count how many candidates points are in each logo, so we can order the brands logos by a number of comparison: the first logo candidate is the logo with more candidates points. This is a fast method, but it's too much naive because it had some problems. The segmentation creates many groups, and some-

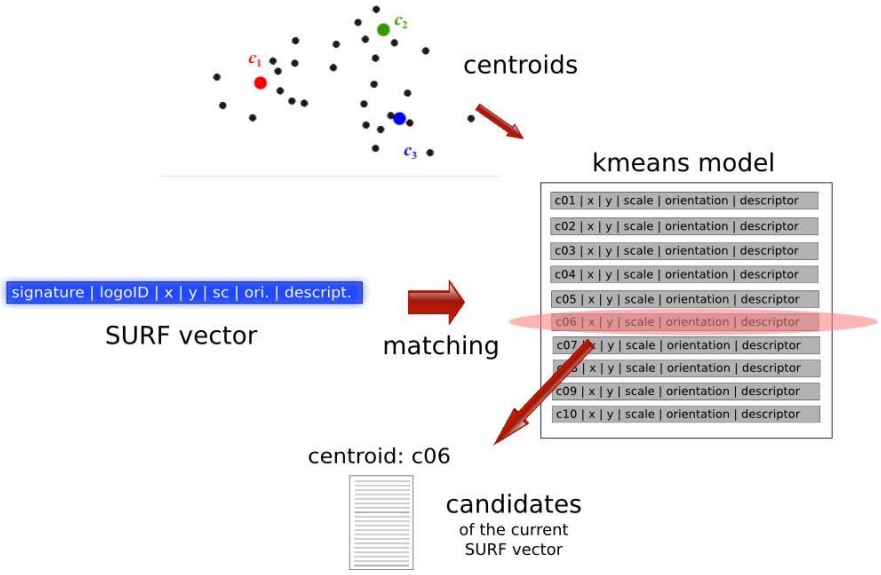

Figure 6.4. Computed the centers of the logos collection, we obtain a list of centroids vectors, we have to match the input SURF with all of these and to choose the best one. The candidates are all the logos with grouped with this centroid.

times there are some vectors that are not so unique but are more similar to *noise*. For the segmentation idea all these points have to store in the same segment. Making an example we are talking about points useless to the pattern matching: as the white points, or with black background or in the corner of the image.. points not good for the detection. So, all the bins have a constant number of candidates, but these group have a huge number of commons vectors: these are not useful to define the correct candidates: we have to *filter them*.

In this section there are described two techniques, the first one is a variation of the Naive Technique but with a weight technique used in *text mining*: *TF-IDF*. The second one is a innovative idea based on the distance point-to-point of each SURF Vectors. The aim is to compare the distances value of the logos candidates to the distances of the points in the input image: if a set of points have the same distance in an image and in a logo, than it's possible that the image contains that logo. To understand better let's see the next two section.

6.2.1 TF-IDF

The *TF-IDF* is a weight technique used in *information retrieval* and *text mining*. It's a statistic method to misure the importance of a word in a document. Technically it's the fusion of two different measurement:

- **TF**: (Term-Frequency) The importance of a text increases proportionally to the number of times a word appears. Usually this value has to be normalized.
- **IDF**: (Inverse Document Frequency) The terms that appear in many documents are less important. (i.e. "the", "at", ..)

Let's see how it works: call N the total number of documents, k_i the term (or word), n_i the number of documents that contain the i-term. The weight given from the TF-IDF will be:

$$w_i = tf_i * \log \frac{N}{n_i}$$

To apply this technique to this project, it is necessary to see the thing in a different way. The term k_i, or word in text retrieval, are the *signature* of the points; the text documents n_i are the logos images. So with term frequency in logo detection, it means how many times a logo contains a signature, and it's possible to avoid the commons terms with the IDF.

The advantage of this technique is that it gives more important to the points that are not so commons, assuming that the points more rare are more discriminative to the detection. Besides it's very simple to implement and with a fast computation.

6.2.2 Proportional Technique

This technique is based on the distance between the SURF points. To understand better, look the figure 6.5: there is a logo with the extracted SURF points plotted, we are talking about the set of the distances from a point to another. The idea is to compare not only the SURF vectors but also the distance between them. However, the problem is that it's not possible to compare the distances, because the scale of the image is different and the brands could be in a different orientation, etc...To do this we have to compare the proportions instead of the distances.

Figure 6.5. This is a simple image with some features points plotted. To understand the Proportional Technique you have to image to compute the distance between each of these points. They will be stored in a matrix of distance.

So, first of all it's required to store a matrix of distances of all the points (for each logo): it has to be a distance point-to-point, so if a logo has in average 100 of points, the matrix will be an half-matrix of $100x100$ values.
When we retrieve all the candidates vectors from the image SURF points, we obtain for each logos some commons points. Now we can do two different thing:

- Compute the distances between the retrieved points in the image of a logo, and the relative of the logo. If the proportions are similar than the current logo candidate takes more point in the rank.
- For each logo candidate with some commons points in the image, compare all the distances of the images to all the distances of the logo (yet built) and keep the more similar proportions if present. *Why this variant?* This one is more expensive and it has a bigger computational cost, but assume there are some errors during the classification step, so it means that some very commons SURF points between the image and the correct logo, are split in different classes. They will never be ranked together. With this variant of Proportional Technique, we take in exam all the points of all the candidates logos another time, like a *second chance*, and we compare the proportion to find similar distances between the points. If there are similar values it's probably that this logo is in there.

Chapter 7

Conclusions and Future Developments

This thesis, as we said in the introduction, is a piece of a bigger work of a research project. There are not mentioned all the experiments and the tests results for two reasons: they are still in developments and they are not include directly on my thesis' job. However, there are some observations and conclusions to discuss.

Talking about the Variance-Balanced Decision Tree, described in Section 5.1, there are some points of discussion about the way to divide the vectors. It splits the collection in function of the variance: the aim is to obtain two different sub-trees, sets of points that are more different as possible. So, the dimension is chosen by the maximum variance, and the splitting value is chosen by the median. The problem is there: the median is a precise value and it could divides two vectors even they have this value very close, but with the threshold in the middle. Let make an example to understand: looking the Figure 5.1 the splitting dimension in the second node is the 45th with threshold of 122. If there are two vectors with the value in this dimension of 121 and 123, so very close, they for sure will be divided at this node. The risk is to put in different groups some descriptors instead very similar. For this reason in the project there was implemented another kind of decision tree: a *ternary decision tree*. The aim is the same, but the splitting works in a different manner: there are two threshold values and not only one given by the median. As shown in Figure 7.1 they have chosen to create three subsets with the same dimension.

This new segmentation method has to correct the VBDT problem, but the experimentation and the tests about that are not be done yet.

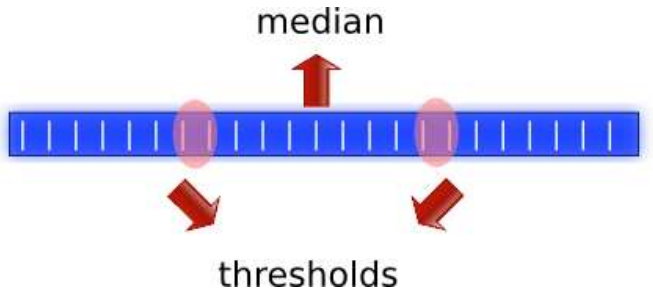

Figure 7.1. This is the idea of the Ternary Tree splitting: instead to split the collection by the median value, it takes two threshold to obtain three subsets of constant dimension.

Using the Random Vectors there are interesting results during the logo detection of an input image. One of the issues, is how to treat the "very common random vectors": those vectors that have a huge number of candidates points. It's possible to assume that they are something like noise or, in other words, they represents some very commons characteristics not useful to the segmentation. In this case a smart solution way should be filter all these candidates to loose a lot of false positives.

About the ranking techniques, there are good results with the *Proportional Rank* but the computational time still be the problem. We compute earlier all the distances for each logo points, storing one matrix for each logo, but to calculate the distances of the image points and make the proportional to each of the candidates, takes a lot of time.
The other method $TF - IDF$ works fine, it's a good compromise to give precision and computational speed.

7.1 Future Developments

There are masses of ways to develop this thesis, and of course this project. In this section there is a description of a bunch of solutions, modifies and future ideas. Some of there are based on papers that could help the goal of this project, other ones are a sets of corrections that could optimize the technique used.

Proportional Rank

There is a simple but interesting development of the Proportional Rank. Instead to count only the matched points between the input image and the

candidates logos, we compute the proportional distance to all the features vectors. So, in this case we can re-consider points that don't have a match but they are a good ones, maybe because they are in a different group or in a filtered one, given to them a *second change*. This is a way to recover lost information or *false negatives*.

The problem is still to be the computational time, because the calculation of the proportion of all the distances between all the points could take a long time. Nevertheless this method could be also a way to evaluate a segmentation/classification technique, comparing only the correctness of them. Than we can understand if the logos retrieved are the best ones or not, and if the ranking technique used works well.

Bag of Visual Words

Bag of Visual Words is a method to represent image, and regards the problem of automatic image categorization[33, 34, 35].

The idea is based to a features collection segmentation like the work done in this project. As explained in [36], treating each cluster as a *visual word*, we can see an image as a *bags of visual word* to be more precise as a vector contained the weighted count of each visual word. The difference is about how manage these words: they apply techniques used in text categorization as term weighting, stop word removal, weighting of visual words, etc...

The main problem of this "concept" is how to build the clusters, because they have to make an hard selection of the vectors: all the group created have to be very similar. The goal is to obtain a *grouping based on the content* like a categories: "landscape", "portrait", "object"... to have a separation of meaning. In my advise the best solution is to divide the collection more in details as: "sun", "sea", "wall", "table", "door"... but in this way the problem is still in the detection of the features. We have to work on each value of the descriptor vectors to be able to understand what every point means. Starting by a vocabulary of this type we can easy understand what is represented in a image. For example, if we found: "chair", "table", "person", "bottle"... we can assume that it contain a scene in a bar, or in a restaurant. Moreover if we add to each visual word a weight, and we count a big number of "chair" and "bottle" the assumption of "bar" is most probable.

This concept have different developments, and it has all the characteristics to become a landmark for the future of image detection.

Image Segmentation

It's not the first time in this thesis that we talk about the image segmentation. The main assumption is that: an image should be a set of different objects and, of course, they should be a set of different surfaces. The idea is to split these surfaces and to analyze all these as a separate image. Why? The hypothesis is simple: a logo has to be in a defined region, not in the middle.
Based on this idea, we can divide the image in image-segments and extract the features of each of them, so we compare a set of points detected in a region with all the logos' points. The comparison should be faster and more precise without many false negative.

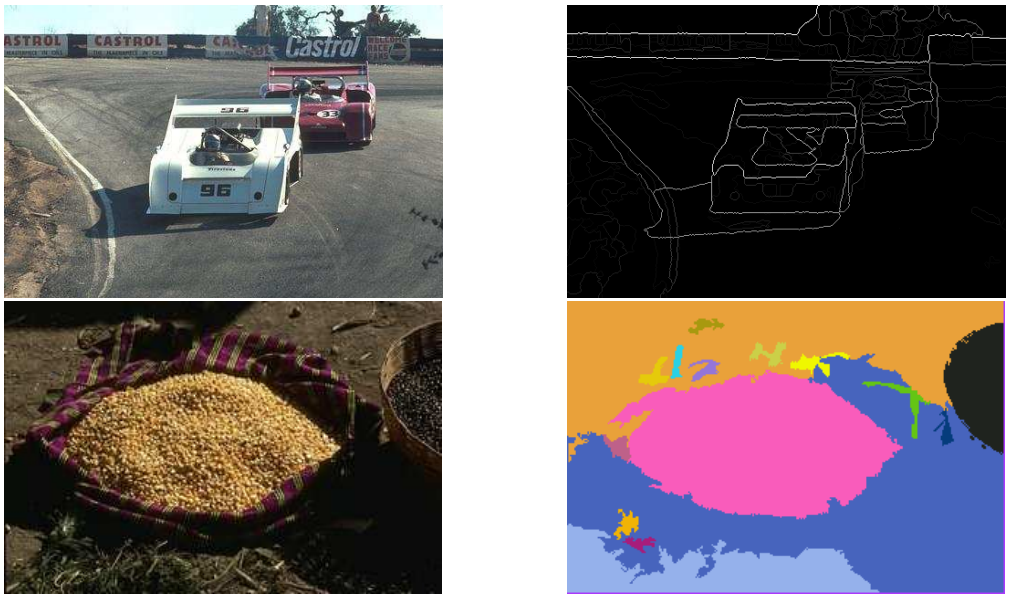

Figure 7.2. Examples of Segmentation techniques, the first one by Berkeley and the second one by MIT.

Now the issue is how to segment an image. There are very good algorithms in literature, personally I have tested [29] and [30], respectively from MIT and Berkeley. In the Figure 7.2 there are posted a couple of examples to understand what we are talking about.
There are different parameters to set: to smooth the input image before

segmenting it, values of thresholds and values of post-processing. By these it's possible to obtain bigger or more precisely regions. For our goal it's better work with big segments, so we can detect a logo inside them. Because the descriptor, as SURF and SIFT, extract a lot of points in the corner or close to borders, so if a logo is selected as a single region, we can't find it. However both these methods are well documented, they are written in C and Java, and the code can be downloaded for free.
So, it will be interesting make a sets of tests with them.

Bibliography

[1] H. Wang and Y. Chen, "Logo detection in document images based on boundary extension of feature rectangles," *Document Analysis and Recognition, International Conference on*, vol. 0, pp. 1335–1339, 2009.

[2] D. Doermann, E. Rivlin, and I. Weiss, "Logo recognition using geometric invariants," in *Document Analysis and Recognition*, pp. 894–897, 1993.

[3] T. D. Pham, "Unconstrained logo detection in document images," *Pattern Recognition*, vol. 36, no. 12, pp. 3023 – 3025, 2003.

[4] G. Zhu and D. Doermann, "Document analysis and recognition," in *Automatic Document Logo Detection*, vol. 2, pp. 864–868, Sept. 2007.

[5] G. Zhu and D. Doermann, "Logo matching for document image retrieval," in *ICDAR '09: Proceedings of the 2009 10th International Conference on Document Analysis and Recognition*, (Washington, DC, USA), pp. 606–610, IEEE Computer Society, 2009.

[6] null Xian-Sheng Hua, null Lie Lu, and null Hong-Jiang Zhang, "Robust learning-based tv commercial detection," *Multimedia and Expo, IEEE International Conference on*, vol. 0, p. 4 pp., 2005.

[7] K. Meisinger, T. Troeger, M. Zeller, and A. Kaup, "Automatic tv logo removal using statistical based logo detection and frequency selective inpainting, presented at the eur," 2005.

[8] S. Rüger, *Multimedia information retrieval*. Synthesis Lectures on Information Concepts, Retrieval and Services, Morgan & Claypool Publishers, 2010.

[9] N. Dimitrova, H.-J. Zhang, B. Shahraray, I. Sezan, T. Huang, and A. Zakhor, "Applications of video-content analysis and retrieval," *IEEE MultiMedia*, vol. 9, no. 3, pp. 42–55, 2002.

[10] J. Sivic and A. Zisserman, "Video google: A text retrieval approach to object matching in videos.," in *ICCV*, pp. 1470–1477, IEEE Computer Society, 2003.

[11] J. Sivic and A. Zisserman, "Video google: Efficient visual search of videos.," in *Toward Category-Level Object Recognition* (J. Ponce, M. Hebert, C. Schmid, and A. Zisserman, eds.), vol. 4170 of *Lecture Notes in Computer Science*, pp. 127–144, Springer, 2006.

[12] J. Sivic, B. C. Russell, A. A. Efros, A. Zisserman, and W. T. Freeman, "Discovering object categories in image collections," in *Proceedings of the International Conference on Computer Vision*, 2005.

[13] R. Fergus, P. Perona, and A. Zisserman, "Object class recognition by unsupervised scale-invariant learning," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, vol. 2, pp. 264–271, June 2003.

[14] D. G. Lowe, "Object recognition from local scale-invariant features," in *Proc. of the International Conference on Computer Vision, Corfu*, 1999.

[15] H. Bay, T. Tuytelaars, and L. J. V. Gool, "Surf: Speeded up robust features.," in *ECCV (1)* (A. Leonardis, H. Bischof, and A. Pinz, eds.), vol. 3951 of *Lecture Notes in Computer Science*, pp. 404–417, Springer, 2006.

[16] H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool, "Speeded-up robust features (surf)," *Comput. Vis. Image Underst.*, vol. 110, no. 3, pp. 346–359, 2008.

[17] A. Baumberg, "Reliable feature matching across widely separated views," 2000.

[18] T. Tuytelaars and L. V. Gool, "Wide baseline stereo matching based on local, affinely invariant regions," in *In Proc. BMVC*, pp. 412–425, 2000.

[19] K. Mikolajczyk and C. Schmid, "An affine invariant interest point detector," in *Proceedings of the 7th European Conference on Computer Vision, Copenhagen, Denmark*, pp. 128–142, Springer, 2002. Copenhagen.

[20] M. Brown and D. Lowe, "Invariant features from interest point groups," in *In British Machine Vision Conference*, pp. 656–665, 2002.

[21] K. G. Derpanis, "The harris corner detector," 2004.

[22] H. P. Moravec, *Obstacle avoidance and navigation in the real world by a seeing robot rover*. PhD thesis, Stanford, CA, USA, 1980.

[23] J. Matas, O. Chum, M. Urban, and T. Pajdla, "Robust wide baseline stereo from maximally stable extremal regions," in *BMVC*, 2002.

[24] P.-E. Forssen, "Maximally stable colour regions for recognition and matching," *Computer Vision and Pattern Recognition, IEEE Computer Society Conference on*, vol. 0, pp. 1–8, 2007.

[25] F. Schweiger, B. Zeisl, P. Georgel, G. Schroth, E. Steinbach, and N. Navab, "Maximum detector response markers for SIFT and SURF," in *Vision, Modeling and Visualization Workshop (VMV)*, (Brunswick, Germany), Nov 2009.

[26] C. Valgren and A. J. Lilienthal, "Sift, surf & seasons: Appearance-based long-term localization in outdoor environments," *Robot. Auton. Syst.*, vol. 58, no. 2, pp. 149–156, 2010.

[27] K. Mikolajczyk, T. Tuytelaars, C. Schmid, A. Zisserman, J. Matas, F. Schaffalitzky, T. Kadir, and L. V. Gool, "A comparison of affine region detectors," *Int. J. Comput. Vision*, vol. 65, no. 1-2, pp. 43–72, 2005.

[28] K. Mikolajczyk and C. Schmid, "A performance evaluation of local descriptors," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 27, no. 10, pp. 1615–1630, 2005.

[29] P. F. Felzenszwalb and D. P. Huttenlocher, "Efficient graph-based image segmentation," *Int. J. Comput. Vision*, vol. 59, no. 2, pp. 167–181, 2004.

[30] P. Arbelaez, M. Maire, C. C. Fowlkes, and J. Malik, "From contours to regions: An empirical evaluation.," in *CVPR*, pp. 2294–2301, IEEE, 2009.

[31] H. Wang and Y. Chen, "Logo detection in document images based on boundary extension of feature rectangles," *Document Analysis and Recognition, International Conference on*, vol. 0, pp. 1335–1339, 2009.

[32] R. Baeza-Yates and B. Ribeiro-Neto, *Modern Information Retrieval*. Addison Wesley, May 1999.

[33] S. Lazebnik, C. Schmid, and J. Ponce, "Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories," in *CVPR '06: Proceedings of the 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, (Washington, DC, USA), pp. 2169–2178, IEEE Computer Society, 2006.

[34] T. Deselaers, L. Pimenidis, and H. Ney, "Bag-of-visual-words models for adult image classification and filtering," in *International Conference on Pattern Recognition*, (Tampa, Florida, USA), Dec. 2008.

[35] E. Nowak, F. Jurie, and B. Triggs, "Sampling strategies for bag-of-features image classification," in *In Proc. ECCV*, pp. 490–503, Springer, 2006.

[36] J. Yang, Y.-G. Jiang, A. G. Hauptmann, and C.-W. Ngo, "Evaluating bag-of-visual-words representations in scene classification," in *MIR '07: Proceedings of the international workshop on Workshop on multimedia information retrieval*, (New York, NY, USA), pp. 197–206, ACM, 2007.