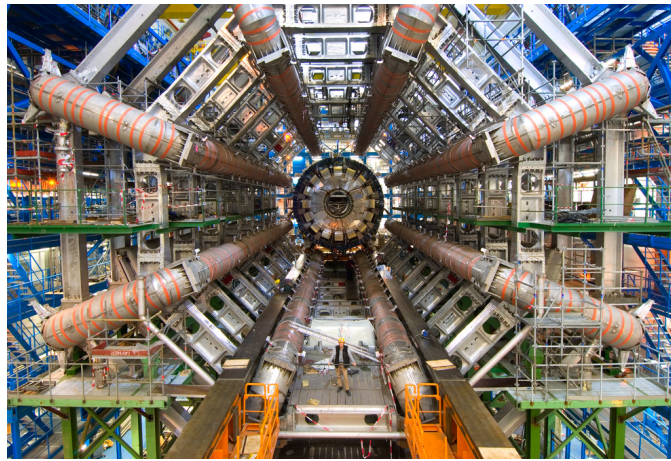


CERN Multimedia Library

Fast and scalable image auto-tagging

MASTER PROJECT
Computer Science section



Author:

Camille FREJAVILLE

Supervisors:

CERN: Ludmila MARIAN

EPFL: Vincent LEPETIT

Contents

1	Introduction	11
1.1	Context	11
1.1.1	CERN	11
1.1.2	Invenio	11
1.1.3	Multimedia	12
1.2	Project	13
1.2.1	Motivations	13
1.2.2	Objectives	14
1.2.3	Implementation choices	14
2	The tagging interface	17
2.1	User's manual tools	17
2.2	Suggestions	17
2.2.1	Face detection	17
2.2.2	Face recognition	19
3	Face detection	21
3.1	Context	21
3.2	The Viola and Jones algorithm	22
3.2.1	Principle	22
3.2.2	Features	22
3.2.3	Cascade training	24
3.3	OpenCV's implementation and data	28
3.4	Cascade training for face profiles	29
3.4.1	Data	29
3.4.2	Method	29
3.5	Results	31
3.5.1	Evaluation framework	31
3.5.2	Results	32
3.6	Improvements	37
3.6.1	On the data	37

3.6.2	On the algorithm	38
4	Face recognition	41
4.1	Introduction	41
4.2	Recognition inside a collection	42
4.2.1	Assumptions	42
4.2.2	Algorithm	43
4.2.3	Results	47
4.3	Recognition from previous tagging	52
4.3.1	Context	52
4.3.2	Algorithms	53
4.3.3	Results	62
4.3.4	Propagation to non frontal faces	63
4.4	Summary of the full process	66
5	Conclusion	71
	Bibliographie	73

List of Figures

1.1	A part of the CDS website welcome page	12
1.2	An image collection contained in CDS	13
1.3	An example of the current way of telling who is in a photo . . .	14
2.1	Area selection for tagging	18
2.2	Title's form for the tag	18
2.3	Tagged image, with all tags displayed at once	19
2.4	Face detection result, with all tags displayed at once	20
2.5	Tag suggestion.	20
3.1	Haar-like features used in OpenCV implementation	23
3.2	(a) The basic LBP operator. (b) The 9×9 MB-LBP operator. In each sub-region, average sum of image intensity is computed. These average sums are then thresholded by that of the center block. MB-LBP is then obtained.	24
3.3	The AdaBoost algorithm. T weak classifiers, each based on only one feature, are chosen for having the lowest error. They are combined in one strong classifier. The samples used for learning are re-weighted at each iteration depending on their difficulty to be classified.	26
3.4	An example of the result of a face detection using OpenCV's already trained classifier for profile faces.	28
3.5	An example of pictures taken from the AFLW database	30
3.6	(a) Pose of the head as described in the AFLW database. (b) AFLW's 21 landmarks	31
3.7	Number of faces used for training and testing for each head orientation	31
3.8	Detection results for the LBP features, the Gentle AdaBoost algorithm and a varying sample size (from 20×20 to 60×60). The bigger the sample size is, the lower the false positive and true positive rates are.	33

3.9	Detection results for the LBP features and the Discrete AdaBoost algorithm and a varying sample size (from 20×20 to 60×60). The bigger the sample size is, the lower the false positive and true positive rates are.	34
3.10	Detection results for the LBP features and the Real AdaBoost algorithm and a varying sample size (from 20×20 to 60×60). The bigger the sample size is, the lower the false positive and true positive rates are.	34
3.11	Detection results for the LBP features and the LogitBoost algorithm and a varying sample size (from 20×20 to 60×60). The bigger the sample size is, the lower the false positive and true positive rates are.	35
3.12	Detection results for the LBP features and the four algorithms: Gentle AdaBoost, Discrete AdaBoost, Real AdaBoost, LogitBoost. The Gentle AdaBoost and the Discrete AdaBoost algorithms perform better than the two other ones. The algorithm to choose depends on the data it is used on.	36
3.13	Detection results for the LBP features and the Discrete AdaBoost algorithm with a varying number of stages, from 10 to 20. With more stages, less mistakes are made, as the last stages are supposed to deal with the most complex samples, but some faces are also not detected.	36
3.14	Detection results for the LBP features and the Discrete AdaBoost algorithm with a varying minimal hit rate (MHR), from 0.6 to 0.995. A high MHR gives a high true positive rate but also a high false positive rate, as the training algorithm accepts more errors in order to fulfill the MHR requirement.	37
3.15	Detection results for the LBP features and the Discrete AdaBoost algorithm with a varying maximal false alarm rate (MFAR), from 0.6 to 0.995. A low MFAR gives a lower false positive rate but also lowers the true positive rate.	38
4.1	Picture from [16] (a) Spatially weighted mask shown relative to a face detection. (b) Cut out of a person with face detection superimposed. (c) Weight mask superimposed over the image.	47
4.2	First recognition example using clothes. top: photo tagged by hand, bottom: resulting suggestions, retrieved without any thresholding.	48

-
- 4.3 Possibilities for the tag Mark. The model has been taken from figure 4.2 and the ellipses show the 10 best choices for the face color histograms, refined with Camshift. The green ellipse attests for the best choice after clothes comparison. Note that Mark's face wasn't in the 10 best possibilities so the 1st choice is completely irrelevant. It is discarded after thresholding. . . . 49
- 4.4 Possibilities for the tag Joe. The model has been taken from figure 4.2 and the ellipses show the 10 best choices for the face color histograms, refined with Camshift. The green ellipse attests for the best choice after clothes comparison. As for Mark, Joe's face is not among the 10 best choices, thus the best possibility is completely irrelevant and the resulting distance to the model is pretty high. It is discarded. 50
- 4.5 Possibilities for the tag Liviu. The model has been taken from figure 4.2 and the ellipses show the 10 best choices for the face color histograms, refined with Camshift. The green ellipse attests for the best choice after clothes comparison. Here, the 10 best possibilities for the color histograms are not faces. The Camshift algorithm outputs shrunk ellipses, as it cannot converge towards a face position. This suggestion is discarded before retrieving the suggestions to the tagging interface. . . . 50
- 4.6 Possibilities for the tag John. The model has been taken from figure 4.2 and the ellipses show the 10 best choices for the face color histograms, refined with Camshift. The green ellipse attests for the best choice after clothes comparison. John has been recognized. There was two serious candidates: John and David but the clothes comparison allowed to choose the right face. 51
- 4.7 Possibilities for the tag Andrzej. The model has been taken from figure 4.2 and the ellipses show the 10 best choices for the face color histograms, refined with Camshift. The green ellipse attests for the best choice after clothes comparison. There was 4 faces among the 10 best positions and Andrzej's face was not the choice with smallest distance if we only consider the face histograms. Fortunately, the clothes comparison allows us to find the true result. 51

4.8	Possibilities for the tag David. The model has been taken from figure 4.2 and the ellipses show the 10 best choices for the face color histograms, refined with Camshift. The green ellipse at-tests for the best choice after clothes comparison. David has been detected as the most probable face after clothes comparison between the 10 choices and the clothes of the model. . . .	52
4.9	An example of face normalization on the AT&T faces. (a) and (b) are the faces before normalization and (c) and (d) the normalized faces. The eyes have been detected and the faces have been scaled, cropped and rotated so that the eyes are at the same position in each sample.	53
4.10	Example where LDA finds a better subspace than PCA. The blue and red dots represent two different classes. The PCA projects points onto the blue line, which results in a really bad class separation. LDA uses the between classes variance to find the red line and projects the points on it. The two classes are better separated.	57
4.11	A circular neighborhood with 8 points and a radius of 2. The points are bilinearly interpolated if not centered on a pixel. The LBP response is computed as follows: for each point on the circle, we compare its value with the center one. If it is bigger, we output 1 and 0 otherwise. The result is then a binary vector of length 8.	59
4.12	Recognition results for the Eigenfaces and the Fisherfaces methods varying the subspace dimensions. A low number of dimensions results in a higher error rate, as we are losing details. For the Fisherfaces method, a number lower than the number of classes results in an even higher error rate.	64
4.13	Recognition results for the LBPH method varying the radius for the Circular Local Binary Patterns. The optimal radius depends on the sample size. A small radius results in features that are too much describing local variations and a large radius result in features that don't account for local texture anymore.	64
4.14	Recognition results for the LBPH method varying the grid dimensions. Small grid dimensions mean large regions for computing the LBP features histogram and a loss of details. A bigger grid mean more attention being paid to local variation.	65
4.15	Tagging and suggestion process in the case only one image is uploaded	67
4.16	Tagging and suggestion process for a collection	68

Abstract

Inside Invenio, the web-based integrated system for handling digital libraries developed at CERN, there is a media module, enabling users to upload photos and videos. Especially in CDS, the Invenio instance used at CERN, people use this digital library to upload pictures of official events that took place at CERN. However, so far, there was no way of tagging what's inside these photos.

This project is meant to solve the problem of tagging persons in a photo in an easy and fast way. First, by implementing a complete tagging interface that allows the user to square parts of the photo, resize them, move them and give them a name. Second, by running face detection so that squares already appear on faces and the user just has to fill the title field. Finally, by running a face recognition system that learned from previous tags created by users.

In this report, we will show how we implemented the tagging interface, how we improved the existing face detector to make it more efficient, which face detection methods we used and how we combined them to have a fully working framework.

Finally, we will show the results we obtained and the study we did on the parameters to choose for the different algorithms.

Acknowledgments

I would first like to thank all the persons who made this Master Project at CERN possible: Vincent Lepetit for supervising my thesis from EPFL, Jean-Yves Le Meur, for accepting me in his team, and Ludmila Marian for supervising my thesis at CERN and for proposing such an interesting project. Secondly, I would like to express my deepest appreciation to my EPFL supervisor, Vincent Lepetit for all the discussions that we had during the project and for his helpful advises.

Then, I would like to thank my supervisor at CERN, Ludmila Marian, for her availability and for all her help on the various aspects of my project, ranging from technical questions to conception matters and deep discussions about recognition methods.

Finally, I would like to express my gratitude to all Invenio members, who quickly integrated me in the team, and gave me technical support in various areas.

Chapter 1

Introduction

1.1 Context

1.1.1 CERN

The European Organization for Nuclear Research, CERN (Centre Européen pour la Recherche Nucleaire in French) [1] is the world's largest laboratory for particle physics. Founded in 1954, it is located on the border between France and Switzerland, near Geneva, and has now 21 member states. It employs more than 2,300 full-time employees and 1,500 part-time employees and welcomes every year thousands of visiting scientists and engineers. It provides scientists with highly complex scientific instruments that allow them to study the basic constituents of matter.

Among CERN's achievements, we can cite this year Nobel Prize in Physics awarded to Peter Higgs and François Englert for their Higgs Boson which existence was confirmed through CERN's experiments.

But CERN is also known for being the place where the World Wide Web was invented in 1989 and where the first website was created [2].

1.1.2 Invenio

CERN has always been involved, since its beginning, in the cooperation and open release of scientific results. First with the printings distributed for free by the CERN library, then using electronic means after the World Wide Web was designed, until now, with the CERN Document Server (CDS), an instance of the software platform Invenio, developed at CERN.

Invenio, and its instance CDS [3], was built with the goal of sharing and indexing scientific publications. The CERN by itself has millions of records and CDS also harvests records from other sources.

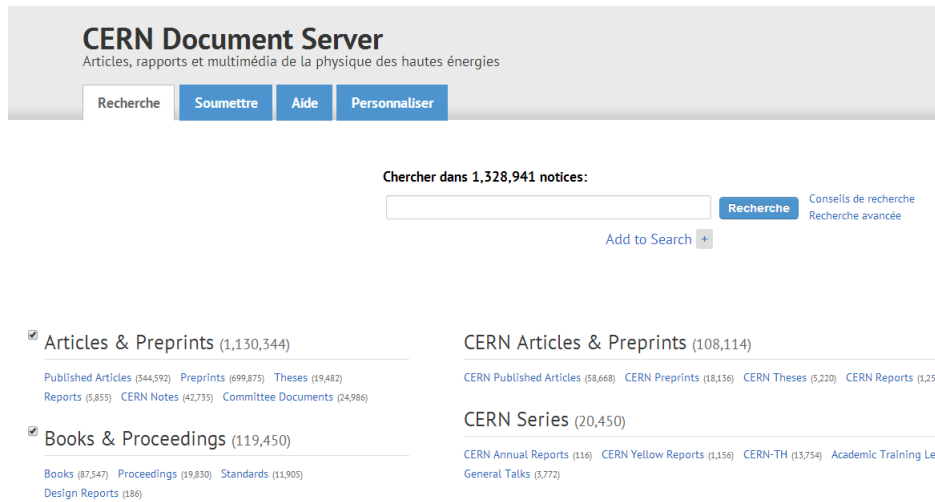


Figure 1.1: A part of the CDS website welcome page

Invenio is a suite of applications which provides the framework and tools for building and managing an autonomous digital library server. It comes with advanced tools for indexing and ranking documents and is well designed for management of document repositories from medium to large size. It is organized in independent modules, each of them implements a specific functionality, and is continuously improved and extended by a highly motivated team. It is accessible via a web based personalized interface [4]. This interface allows the user to browse documents by keywords but also by use of a tree like index. It also includes collaborative tools such as baskets and alerts for new documents.

The Invenio software is free and open source under the terms of the GNU General Public License and is now used by several other institutions across the world, including EPFL [5].

Invenio has been developed mainly in Python.

1.1.3 Multimedia

Invenio also provides the possibility to upload pictures and it is widely used by CERN to post official pictures of events taking place at CERN or involving CERN.

These pictures are uploaded grouped together as collections.

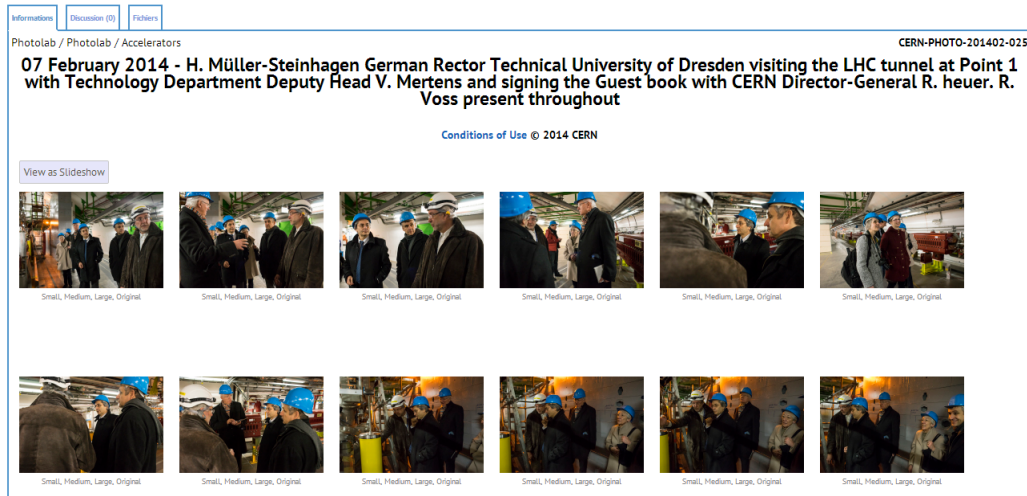


Figure 1.2: An image collection contained in CDS

1.2 Project

1.2.1 Motivations

CDS has now more than 150,000 pictures uploaded and many pictures continue to be uploaded everyday. But so far, the only way to describe the content of the images is to put the information in the title or in the comments. For instance, if someone wants to name the persons in the photo, he has to write something that looks like this: "From left to right: Mr X, Mr Y..." (see figure 1.3). With the increasing popularity of picture sharing and tagging, the Invenio team thought it was time to have a tagging interface that works the same way as the ones you can find in Facebook or in Google's Picasa.

But more than just a tool that allows someone to select parts of a photo and put a name on it, this project is meant to provide helping tools to make the user's task easier by displaying automatic suggestions on top of the image to be tagged. Especially in CDS, users upload several pictures at once, grouped in a collection, that are related to one single event. They sometimes upload up to 200 photos at the same time and thus would have to spend a lot of time to tag each of them. The intent of this project is to automatically use information from previously tagged pictures to suggest new tags to the user. This project focuses on faces, as it will be the main subject of tagging in CDS and Invenio.



Figure 1.3: An example of the current way of telling who is in a photo

1.2.2 Objectives

This project has three main objectives. The first one is to implement the tagging interface. This interface has to allow the user to square regions of the photo, describe them and possibly modify them afterwards. It also displays the tags on top on the picture every time the image appears in Invenio afterwards. The display is done such that a tag is seen when the mouse comes over it and is hidden otherwise.

The second objective is to provide a robust face detection, not only for frontal faces but for all orientations of faces. This detection happens at first intent for a user to tag a photo. In order to make the process of tagging faster for the user, the faces are already squared and the user just has to fill in the name.

The last, most challenging, objective is to recognize already tagged faces using face recognition algorithms. This part uses a continuously augmented database of faces alimented by Invenio's users but also information from the pictures being tagged inside the current collection.

1.2.3 Implementation choices

For the tagging interface, we chose to use JQuery, as it is easy to use and well suited for handling events and displaying overlays on top of elements. For the image processing part, we made the choice to use OpenCV [6] for most of the algorithms developed. The main reason for that choice is that this library, coded in C and C++, has proven to be fast and efficient in its way to represent data, images and operations on them. It also comes with a lot

of features, such as the most basic operations (color conversion, histograms, etc), the most famous computer vision algorithms, a lot of machine learning algorithms, searching and clustering methods, etc. Since it is easy to use and it is open source, OpenCV received a lot of contributions from independent developers and researchers. On top of that, this library has a wrapper for Python, which makes it pretty easy to interface with Invenio.

For some parts of the code, however, we chose to implement everything in C++ using OpenCV and to implement our own wrappers, as it is faster to do all the processing in C++ and then retrieve the result to the Python part. This document will be organized as follows:

In a first time we will talk about face detection, as it is an important part and as we had to improve the existing OpenCV version in order to meet our goal.

The next section will be dedicated to face recognition. We will present two different approaches to the problem and the algorithms we implemented and tested.

Chapter 2

The tagging interface

2.1 User's manual tools

The first tool to be implemented in Invenio is the possibility, for a user that uploaded a photo, to give a name to any part of the picture. The resulting square and its title is then added on top of the image and displayed when the user's mouse comes over it. See figures 2.1, 2.2 and 2.3 to have an overview of how it looks like.

In addition to that, the tags are saved for further utilization. They are stored in a database, individually and all together grouped by photo in a JSON file. The goal, besides providing the possibility to annotate pictures, is also to be capable of performing a search using these tags. So the tags will be indexed by the search engine module together with the other information coming from the picture's title.

2.2 Suggestions

On top of providing to the user a nice interface that enables to tag images, we also want to ease the user's task by pre-processing the images. We focus on faces since it will be the main subject of tagging.

2.2.1 Face detection

The first overlay to be displayed is the face detection. When an image has not already been tagged, we detect faces and square them. That way, the user just has to enter a valid name and confirm. If he leaves the square alone, it won't be saved after he leaves the tagging interface. With this processing, the user can tag quickly the persons he knows but he is not forced to fill in



Figure 2.1: Area selection for tagging

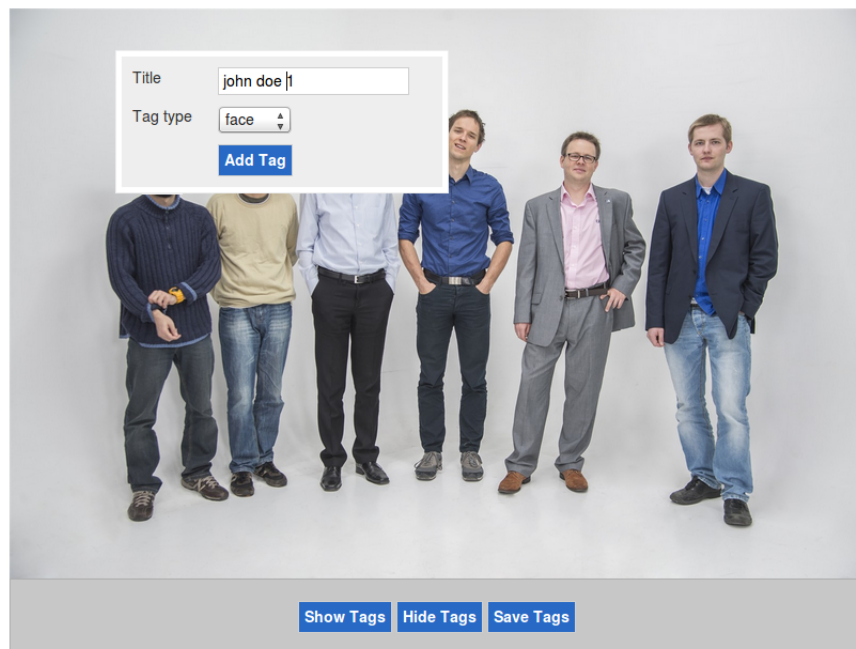


Figure 2.2: Title's form for the tag



Figure 2.3: Tagged image, with all tags displayed at once

everyone's name, especially if it's not relevant. Figure 2.4 shows an example of face detection inside the tagging interface. For that part too, the squares are displayed only when the user's mouse come over it.

2.2.2 Face recognition

The last feature we added to our tagging interface is the face suggestion. That feature uses previously tagged pictures in order to suggest a person's name. The user always has to confirm a suggestion otherwise it will be discarded from the tag list after the tagging interface is closed. See the example figure 2.5.

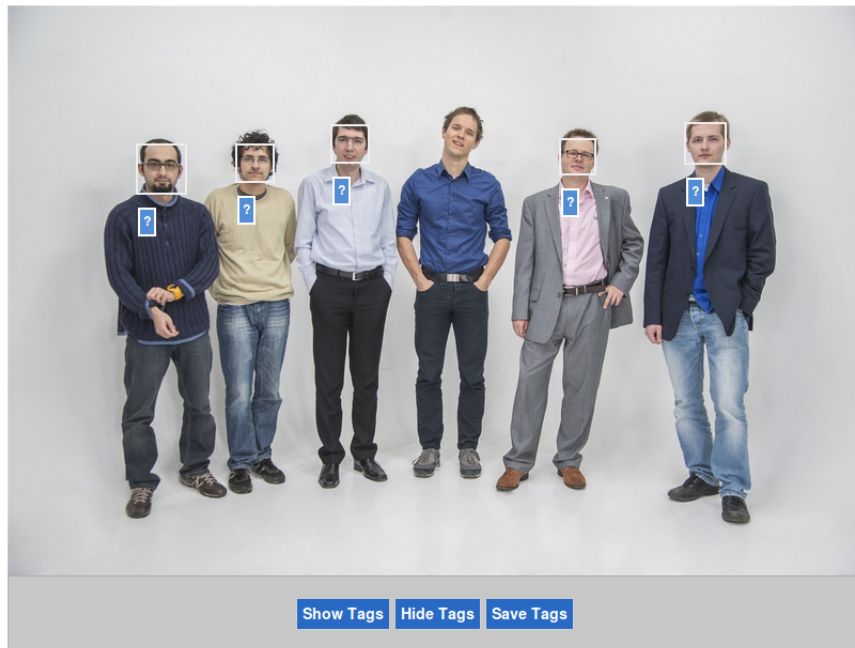


Figure 2.4: Face detection result, with all tags displayed at once

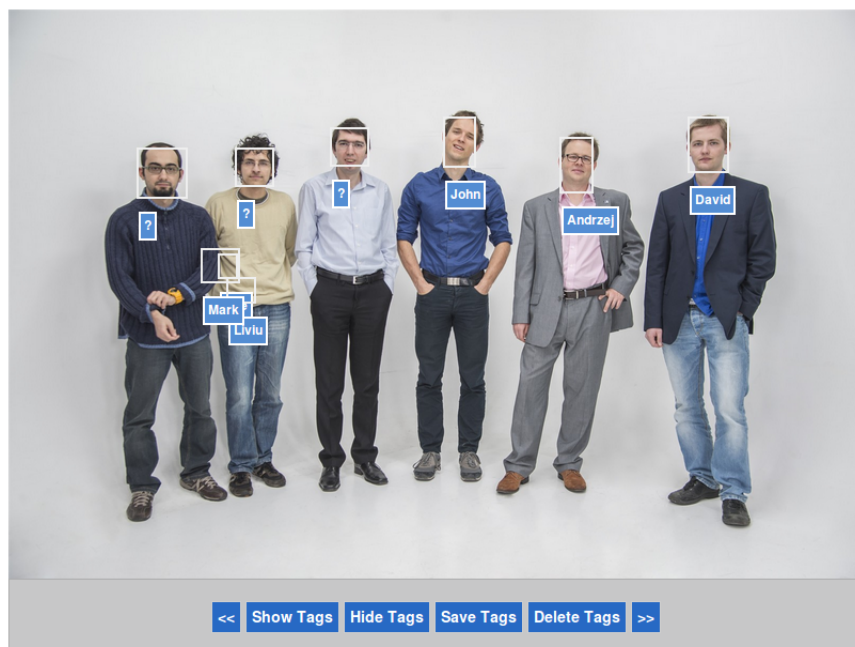


Figure 2.5: Tag suggestion.

Chapter 3

Face detection

The first feature implemented inside the tagging interface is automatic face detection. Although users can tag any part of an image, it's been demonstrated that the main utility of the tagging interface will be for tagging people. So a good start, even before considering recognizing already tagged people, is to at least square every face to make the user's task easier.

We will see that the problem is not trivial, as it requires a lot of work on choosing the training data and on tuning the different parameters. The training process can also take days for training a single classifier.

We will start by describing the context, the typical photos we consider being the ones we will have to deal with. We will then talk about the detection algorithm. We will describe how the learning process is done and how the faces are detected afterwards. We will then present the training that we ran and the results we obtained.

3.1 Context

In CDS and Invenio, there is no strong restriction on the pictures that can be uploaded. The content of the photo vary widely from one picture to another and the task of detecting faces is really challenging in that context because of the whole range of possibilities for a face appearance. The variations include various hair styles, facial hair styles, glasses, expressions changes, illuminations, helmets, and, the most important, face orientation regarding to the camera point of view.

On top of that, we want the face detection to be fast. The face detection is run when a user uploads the image. The picture should be loaded in the tagging interface with the squares on the faces already there. So the face detection has to be real time.

Despite the fact that face detection is now available in a lot of cameras and smartphones, we will see that the problem of detecting a face is not simple, especially if we challenge ourselves with detecting not only frontal faces but also profile faces.

In this chapter, we will present the algorithm commonly used for face detection and its various improvements. We will then present what was the OpenCV already trained model and what we did to improve it. More precisely, we will detail all the choices that can be made during the training process. The main challenge for this part was not to code the whole training and detecting process, as OpenCV already has one available, but it was to find the best training data and the optimal combination of parameters so that our face detector would detect most of the faces contained in the photos, regardless of the background and the orientation of the face.

Building a good testing framework was also important in order to get a good estimation of how the detector would behave in Invenio.

3.2 The Viola and Jones algorithm

A really popular algorithm for face detection is the algorithm first elaborated by Viola and Jones [7] and later improved by Lienhart [8]. This algorithm has the advantage of being fast while achieving good detection rates if trained properly.

3.2.1 Principle

This algorithm is based on supervised machine learning. Examples of positive and negative samples (pictures containing faces and pictures not containing faces) are provided to the learning algorithm. The algorithm extracts features and builds a function, based on this data, that is able to recognize a new sample as being a face or not a face.

3.2.2 Features

A face has to be described somehow. Here, we present two common features that are well suited for faces, as they account for face's characteristics such as eyes, nose, mouth, etc.

Haar-like features

Viola and Jones used features called Haar-like features. Below is a representation of the features used in the initial algorithm and the ones added by

Lienhart afterwards.

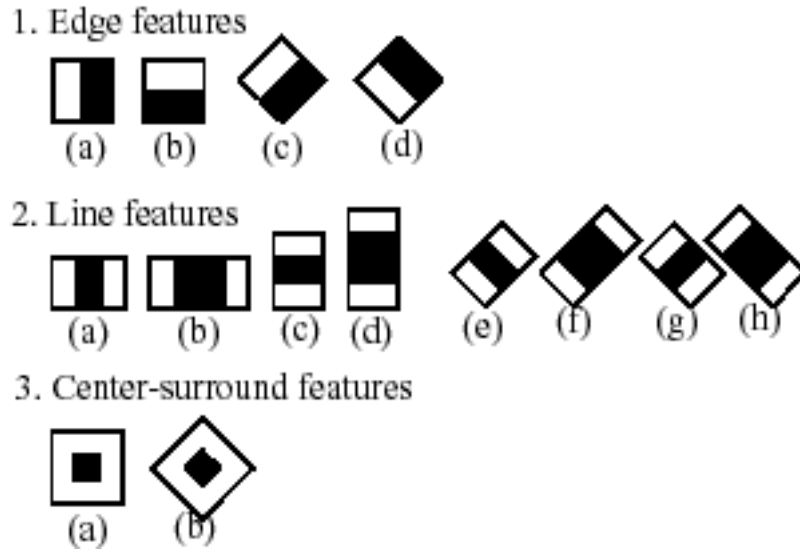


Figure 3.1: Haar-like features used in OpenCV implementation

All the faces used to train the classifier are of the same size. They are matrices of pixels of gray scale intensities and, from each of the masks presented above, a response value is computed. The mask is put on top of the region of the sample. The values of the pixels that are contained in the black part are summed then the response to retrieve for training is the difference between the sum of the pixel values for the whole region and the sum for the black region. For each image sample, Haar-like features are computed at different positions and scales and are stored for training along with their scale and translation information.

Haar-like features are a fast way to obtain interesting information about an image such as lines, borders and isolated dots. This information can attest for the presence of a mouth, a nose or an eye.

To make the computation faster, an integral image is calculated from the sample. An integral image is constructed as follows: each position (x, y) contains the sum of pixel intensities that are above at the left in the original image. Using the integral image, the sum of intensities of any rectangle of pixels in the image can be computed with at most four look-ups at the integral image.

Local Binary Pattern features

Other features that can be used instead of Haar-like features are Local Binary Pattern features (LBP) or, more precisely, Multiscale Block Local Binary Pattern features (MB-LBP) as described in [9]. LBP features are computed as follows: a window of 3×3 pixels is considered and each pixel is compared to the center pixel. When the intensity value for that pixel is bigger than the center one, the response is 1 and 0 otherwise. This results in a binary vector of size 8.

LBP are widely used for texture description but are too local to be robust. In MB-LBP the computation is done on average values of sub-regions instead of single pixels. Here is a representation of the two methods.

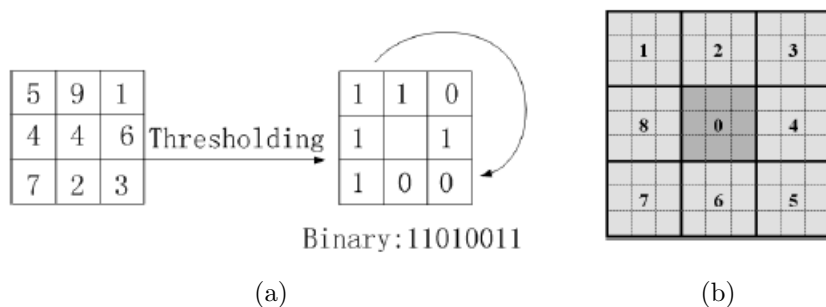


Figure 3.2: (a) The basic LBP operator. (b) The 9×9 MB-LBP operator. In each sub-region, average sum of image intensity is computed. These average sums are then thresholded by that of the center block. MB-LBP is then obtained.

So, using Haar-like features or MB-LBP features, each positive and negative sample is scaled at the same size and represented by a vector containing the features values computed at different positions and scales. All the vectors have the same size and will be provided to a training algorithm that will learn which type of vector should be considered as a face.

3.2.3 Cascade training

Before explaining how the training process works, we will explain briefly how the search for a face in a photo is done.

The search for a face in the image is done in a greedy way. The image is browsed at different positions and scales. For each region of interest in the image, the answer to the question "Is that sample a face?" is retrieved by a cascade of classifiers. The classifiers are organized in cascade such that

each classifier (stage) classifies a candidate as being a face or not and then passes the sample to the next classifier in the cascade. The classifiers at the beginning of the cascade are fast, accept a lot of false positives but allow to discard very quickly the candidates that obviously are not faces. As we go through the cascade, the classifiers are more complicated and are made more precise to deal with more complicated cases. If a candidate passes all the classifiers, it is labeled as a face. Since only a few candidates go through the most computationally expensive stages, this face detection algorithm is fast enough to be run in real time. For a sample that has nothing to do with a face appearance, it is very likely that it won't pass the first classifiers. They are fast and not too precise so that the negative candidates are discarded quickly and that only remain the most interesting cases for the next classifiers.

This is the main reason this algorithm is so famous compared to the others. It computes an exhaustive search for the faces in the image but it is fast enough because of the fact that most of the areas tested are quickly discarded from the list of potential candidates.

The method to build the classifiers is a variant of an algorithm called AdaBoost, meaning adaptive boosting and first formulated by Yoav Freund and Robert Schapire [10]. The algorithm is given positive samples (in our case faces) and negative samples (anything but faces) described by features.

The classifier imagined by Viola and Jones, for each stage of the cascade, is made of T weak classifiers. A weak classifier is based on only one feature. The weak learning algorithm determines the best threshold classification function so that the least number of samples is misclassified. We consider the 1 response as meaning the sample is classified in the face category and the 0 response as meaning the sample is in the not a face.

A weak classifier $h_j(x)$ is described as:

$$h_j(x) = \begin{cases} 1 & \text{if } p_j f_j(x) < p_j \theta_j \\ 0 & \text{otherwise} \end{cases}$$

With f_j the feature, θ_j the threshold and p_j a parity indicating the direction of the equality.

As the number of computed features for a sample is extremely high, AdaBoost is first used to select the features to take into account in the classification. The corresponding weak classifiers are then linearly combined in one strong classifier. The algorithm is summed up figure 3.3.

This algorithm is called Discrete AdaBoost. There are other versions of this algorithm that were invented later, called Real AdaBoost, Gentle AdaBoost and LogitBoost [11].

In the Real AdaBoost algorithm, the weak learner, instead of returning a

- Given example images $(x_1, y_1), \dots, (x_n, y_n)$ where $y_i = 0, 1$ for negative and positive examples respectively.
- Initialize weights $w_{1,i} = \frac{1}{2m}, \frac{1}{2l}$ for $y_i = 0, 1$ respectively, where m and l are the number of negatives and positives respectively.
- For $t = 1, \dots, T$:

1. Normalize the weights,

$$w_{t,i} \leftarrow \frac{w_{t,i}}{\sum_{j=1}^n w_{t,j}}$$

so that w_t is a probability distribution.

2. For each feature, j , train a classifier h_j which is restricted to using a single feature. The error is evaluated with respect to w_t , $\epsilon_j = \sum_i w_i |h_j(x_i) - y_i|$.
3. Choose the classifier, h_t , with the lowest error ϵ_t .
4. Update the weights:

$$w_{t+1,i} = w_{t,i} \beta_t^{1-e_i}$$

where $e_i = 0$ if example x_i is classified correctly, $e_i = 1$ otherwise, and $\beta_t = \frac{\epsilon_t}{1-\epsilon_t}$.

- The final strong classifier is:

$$h(x) = \begin{cases} 1 & \sum_{t=1}^T \alpha_t h_t(x) \geq \frac{1}{2} \sum_{t=1}^T \alpha_t \\ 0 & \text{otherwise} \end{cases}$$

where $\alpha_t = \log \frac{1}{\beta_t}$

Figure 3.3: The AdaBoost algorithm. T weak classifiers, each based on only one feature, are chosen for having the lowest error. They are combined in one strong classifier. The samples used for learning are re-weighted at each iteration depending on their difficulty to be classified.

discrete value (-1 or 1) returns a class probability estimate. $|h_t(x)| = p_t(x) = \hat{P}_w(y = 1|x) \in [0, 1]$. The sign of $h_t(x)$ gives the classification and the probability estimate a measure of confidence about the classification. The strong classifier is then

$$\text{sign} \left(\sum_{t=1}^T \frac{1}{2} \log p_t(x)/(1 - p_t(x)) \right)$$

The Gentle AdaBoost algorithm works the same as the Real AdaBoost algorithm but the resulting strong classifier is

$$\text{sign} \left(\sum_{t=1}^T (P_w(y = 1|x) - P_w(y = -1|x)) \right)$$

This algorithm puts less weight on outlier data points so it is recommended with regression data.

The LogitBoost algorithm is another version based on fitting additive logistic regression models by stage wise optimization of the Bernoulli log-likelihood. For the two classes case, $p(x)$ stands for the probability of $y^* = 1$ ($y^* = (y + 1)/2$ taking values 0, 1) and we have:

$$p(x) = \frac{e^{H(x)}}{e^{H(x)} + e^{-H(x)}}$$

With $H(x) = \sum_{t=1}^T h_t(x)$.

The algorithm starts with $p(x_i) = \frac{1}{2}$ and for each of the T stages, it computes the working responses and weights:

$$z_i = \frac{y_i^* - p(x_i)}{p(x_i)(1 - p(x_i))}$$

$$w_{t,i} = p(x_i)(1 - p(x_i))$$

Then it fits the function $h_t(x)$ using a weighted least squares regression of z_i to x_i using weights $w_{t,i}$.

It updates $H(x)$ with $H(x) + \frac{1}{2}f_t(x)$ and $p(x)$ with $\frac{e^{H(x)}}{e^{H(x)} + e^{-H(x)}}$.

The output classifier is $F(x)$.

As the choice of the algorithm depends on the data to classify, we will test each of these algorithms, along with the parameters that can vary. We will discuss the choices we made in the result section but first, we will talk about the existing classifiers available in the OpenCV data folder.



Figure 3.4: An example of the result of a face detection using OpenCV's already trained classifier for profile faces.

3.3 OpenCV's implementation and data

The OpenCV library [6] has a face detector implementation that works both with haar-like features and LBP features. To use it, one must provide the appropriate trained classifier represented as an XML file. OpenCV source code is provided with several already trained classifiers, among them, classifiers for frontal faces, for profile faces and for different parts of the face (eyes, ears, etc). There are classifiers for both Haar-like features and LBP features. However, only the trained classifier for frontal faces performs well. The face detection rate for profile faces behave pretty bad and the detection rate is lower than 15% (see the results section for more information about the testing set and figure 3.4 for an example).

Fortunately, OpenCV also comes with an API for training classifiers from any data. To build a new classifier, one must provide appropriate negative and positive samples and choose the parameters. Among the parameters that can be chosen, there is:

- the feature type
- the boosting algorithm
- the size of the samples (i.e. all the samples will be scaled at the same size in order to have the same number of features)
- the number of stages (the number of classifiers in the cascade)

- the minimal hit rate for a stage
- the maximal error rate for a stage

3.4 Cascade training for face profiles

As the detection for profile faces using OpenCV's already trained cascades gave pretty poor results, we decided to train our own cascades using a training set close to our context and using the OpenCV API.

3.4.1 Data

For training, we used a database called Annotated Facial Landmarks in the Wild (AFLW) [12] which contains images downloaded from Flickr. These images cover a wide range of face appearances in terms of age, gender, ethnicity, pose, expression, etc and are annotated with up to 21 landmarks per face. Especially, this image collection is well suited for us because it has been taken in the real world by thousands of different users and not in a laboratory under constrained conditions. It pretty well sums up what we expect the user to upload in CDS and Invenio. The data is composed of 25,993 faces contained in 21,997 images.

For the negatives samples, the samples that don't contain any face, we used a mix of several databases. The photos are landscapes, rooms and everyday life objects. There are 3,020 negative photos.

3.4.2 Method

As we said before, the OpenCV's cascades for face profiles got a pretty low detection rate for both Haar-like features and LBP features. As we don't know how these cascades were trained, we can't say what should be improved. However, there are a few possibilities that we explored: the first possibility was that the cascade must have been trained with not enough data or data not representative enough of the real world images. We thus first tried to train a single cascade of classifiers with the data for all the faces that were not frontal faces. The resulting cascade was not giving significantly better results.

A second explanation for the low rate could be that one classifier for all face orientations was too ambitious. The range of appearances for all face poses must have been too large for the classifiers to be able to perform well. The appearance of someone almost facing the camera and the appearance of someone turned 90 degrees from the camera can be considered as highly

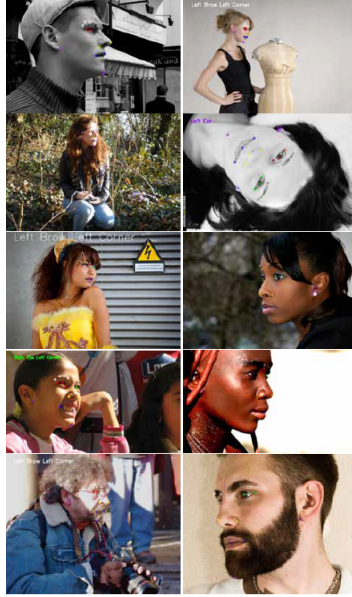


Figure 3.5: An example of pictures taken from the AFLW database

different from the classifiers point of view. Thus, we decided to train six different classifiers for six different ranges of face orientations.

The AFLW database comes with landmarks on all the faces and, from these landmarks, the face's pose can be estimated in terms of three angles (see figure 3.6). The database comes with an API for calculating the pose.

We thus used this angle information sorted the faces per range angle and trained six different classifiers. We dedicated one classifier for a face orientation at every 15 degree for the yaw angle (see figure 3.6). For each face of the AFLW database, we retrieved only the faces with an acceptable range of values for the pitch and roll angles and we sorted the faces in six different folders depending on the yaw angle. We only built detectors for faces turned to the right. Faces turned to the left are mirrored in both training data and testing data. We ran the training several times with different parameters in order to get the higher possible detection rates.

In the next section, we will present our approach for training the classifiers, the different choices that can be made in terms of data and parameters and how we tested our classifiers.

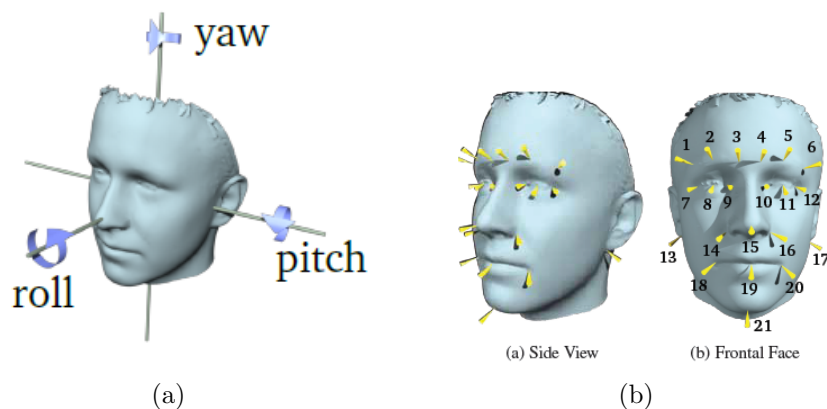


Figure 3.6: (a) Pose of the head as described in the AFLW database. (b) AFLW's 21 landmarks

3.5 Results

Here we describe the results of the different classifiers we trained using the AFLW database. We tuned the different parameters to be chosen in order to get the best detection rates for our case, faces in the wild.

This part of the project required a lot of work on the choice of the data to use and its preparation for training and testing but also for the parameters evaluation and choice and training process which can take several days for only one classifier.

3.5.1 Evaluation framework

To evaluate the efficiency of our cascade of classifiers, we took a subset of the AFLW database to run the tests on. 15% of the faces for each range of orientation have been removed from the training set and used for evaluation. In the figure 3.7 we describe how the data has been sorted, first, between each range of face orientation, second, between training and testing set. The testing images were taken randomly for each orientation.

Database	subset purpose	15°	30°	45°	60°	75°	90°	Total
AFLW	training	4814	3547	2115	1584	1329	990	14289
AFLW	testing	863	632	375	281	235	176	2562

Figure 3.7: Number of faces used for training and testing for each head orientation

3.5.2 Results

The OpenCV's cascade training API comes with a lot of freedom about the choice of parameters, the parameters to take into consideration are:

- The type of feature to extract from the samples: Haar or LBP.
- The boosting algorithm: Discrete AdaBoost, Real AdaBoost, Gentle AdaBoost, LogitBoost.
- The number of stages: the number of classifiers in a cascade, the first one being fast and the last ones being more precise. More stages there is, more precise is the detection but smaller is the number of detected faces.
- The sample size: the size at which all the faces provided for training will be scaled. A bigger size means more features computed, a longer training in practice, but probably a higher detection rate and a lower false detection rate depending on the provided data.
- The minimal hit rate for each stage: the minimal percentage of true faces that should be detected.
- The maximal rate of false alarm: the maximal percentage of mistakes that can be done for each stage.

We will present, first, all the trainings we ran and, for each, the chosen parameters. For each set of parameters, we ran six cascade classifier trainings corresponding to the six face orientations.

Then we will present the results in terms of false positive rate and true positive rate.

The optimal result would be to have 100% of faces detected and no wrong detection. However, this goal cannot be completely achieved and we have two options: either choose the classifier that has the smallest amount of errors, but, on the other hand, that detects less faces or choose the classifier that detects more faces but makes more mistakes. In our context, the best choice is the second one because we want to help the user by squaring the regions of the image that he might want to tag. A wrong detection should have a small impact since the user can just ignore it.

Our goal is to build one face detector using the six detectors, one for each face pose. Thus, for testing, we tested the whole detection rate, the number of faces detected by the combination of the six detectors.

We present our results using ROC curves (Receiver operating characteristic), but for computing the false positive rate, we need to know what the number

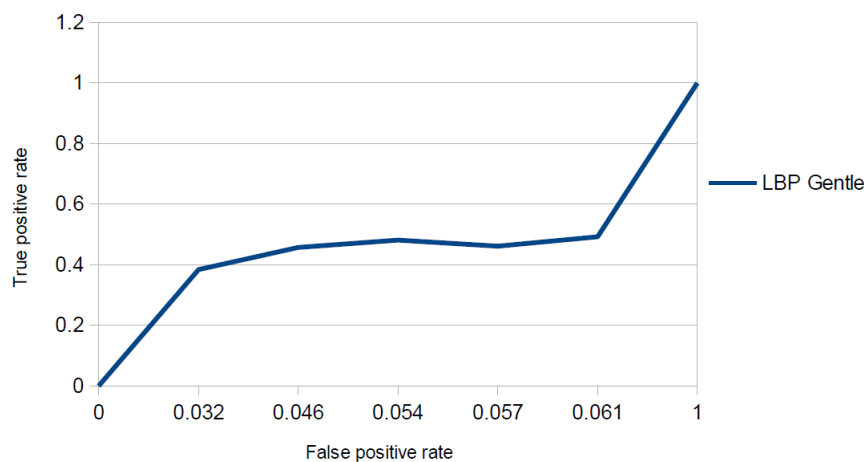


Figure 3.8: Detection results for the LBP features, the Gentle AdaBoost algorithm and a varying sample size (from 20×20 to 60×60). The bigger the sample size is, the lower the false positive and true positive rates are.

of true negatives is. This number is finite but not known as all we know is that the face search considers samples at different scales and translations. So the number of true negatives should be all the sub-windows that have been explored while not classified as a face. We just chose this number, as it is the same for each of the tests we ran and as our goal is to compare the different tests in order to choose the more suitable combination of parameters.

We first trained the classifiers while varying the sample size, from 20×20 to 60×60 . In figures 3.8, 3.9, 3.10 and 3.11, we present the results for the four different boosting algorithms: Gentle AdaBoost, Discrete AdaBoost, Real AdaBoost and LogitBoost. From the results we can say that increasing the sample size decreases the error rate. With a small sample size, the face loses a lot of details, making the detection less precise. On the other hand, the number of faces detected also decreases when we increase the sample size. This is due to the fact that a bigger size and more details lead to a classifier which is more discriminant about what is a face. Another parameter to take into account is the detection time for the classifiers. The classifiers trained on the big sample sizes take more time for the detection than the ones trained on the small size.

In figure 3.12, we made a comparison of the four algorithms using the default parameters and the LBP features (see OpenCV documentation for the default parameters). We conclude that the Gentle and the Discrete AdaBoost methods perform better for the LBP features and our dataset. We will then use these methods for the rest of our tests.

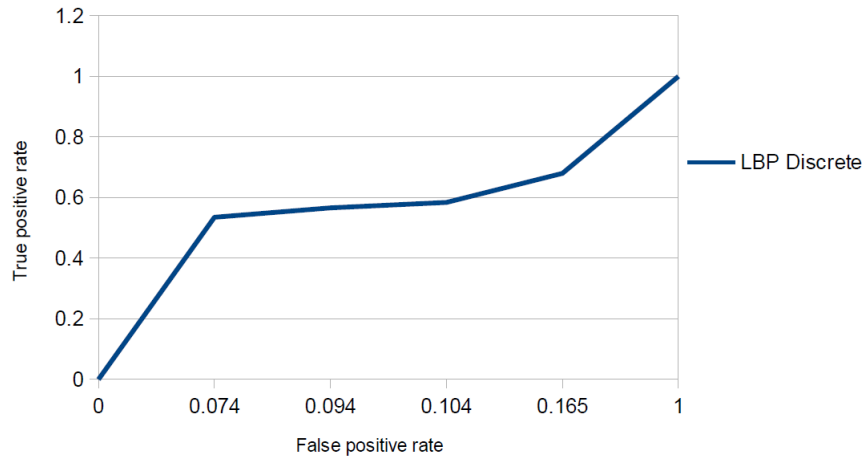


Figure 3.9: Detection results for the LBP features and the Discrete AdaBoost algorithm and a varying sample size (from 20×20 to 60×60). The bigger the sample size is, the lower the false positive and true positive rates are.

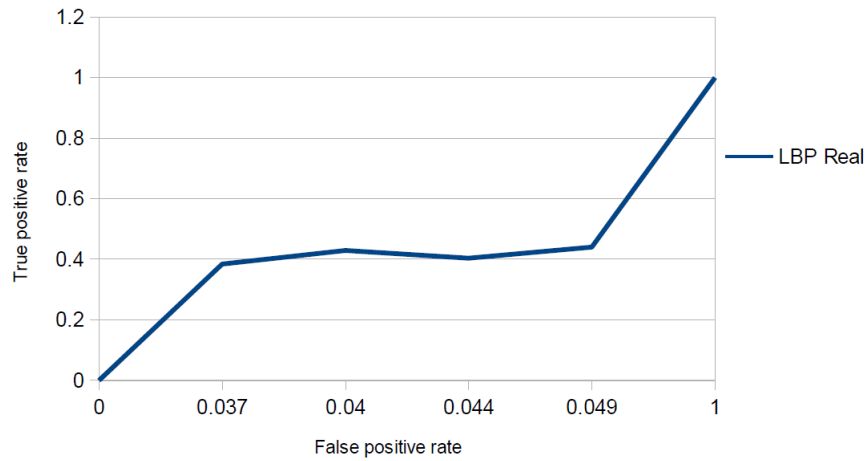


Figure 3.10: Detection results for the LBP features and the Real AdaBoost algorithm and a varying sample size (from 20×20 to 60×60). The bigger the sample size is, the lower the false positive and true positive rates are.

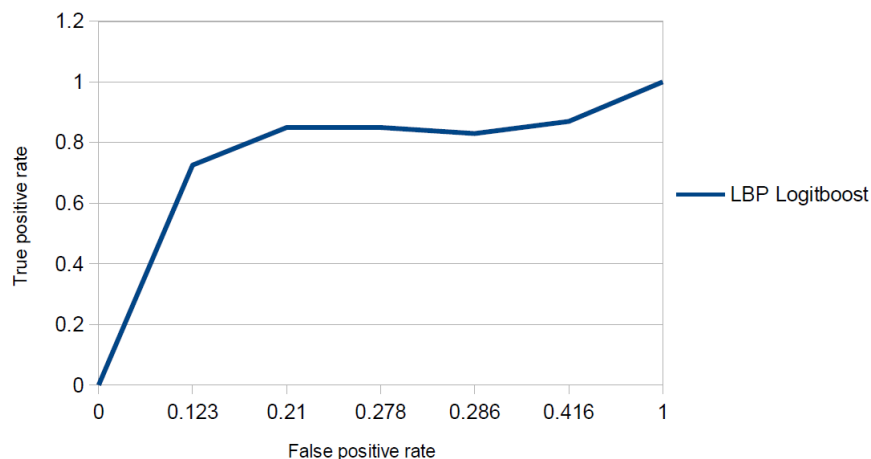


Figure 3.11: Detection results for the LBP features and the LogitBoost algorithm and a varying sample size (from 20×20 to 60×60). The bigger the sample size is, the lower the false positive and true positive rates are.

In figure 3.13, we present a test on the variation of the number of stages with the Discrete algorithm. From this result, we see that more stages means more stability. The classifiers built on less stages (i.e. less classifiers in the cascade), detect more faces but they also make a lot more mistakes (we can almost say that they say yes to everything). Indeed, a cascade classifier built on more stages takes more time to perform the detection. So we should choose a value that is both acceptable in terms of computation time and detection rate.

In figure 3.14, we present a test on varying the minimal hit rate. This parameter is interesting to study as we may ask ourselves if we want a high recognition rate no matter what the cost is. If we put a high minimal hit rate, the training algorithm, in order to be able to fulfill that goal, tends to accept also a lot of false positives. So, putting a smaller value would decrease the number of false positives a lot but would also decrease the number of true faces detected. As we decided to put the priority on detecting a lot of faces, we chose to keep a high minimal hit rate, even though it means getting more errors.

Finally, we present figure 3.15 a test on varying the maximal false alarm rate. As expected, if we lower the maximal false alarm rate, we get less errors. However, we also get less faces detected. Our choice should be a

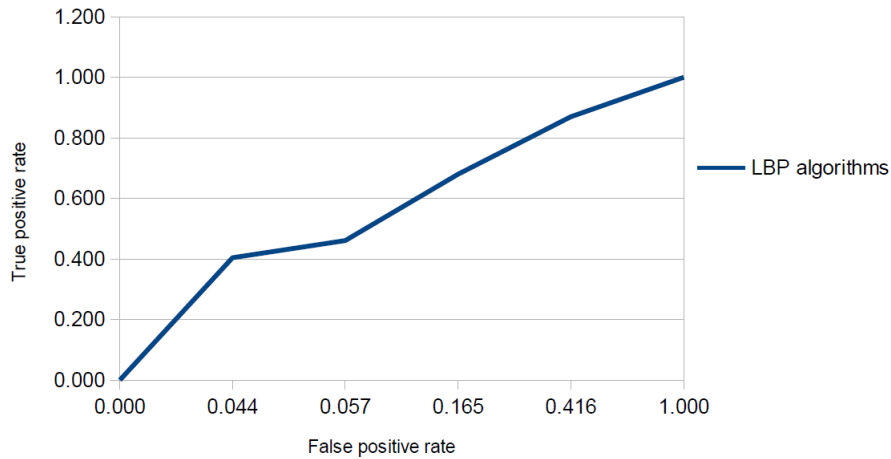


Figure 3.12: Detection results for the LBP features and the four algorithms: Gentle AdaBoost, Discrete AdaBoost, Real AdaBoost, LogitBoost. The Gentle AdaBoost and the Discrete AdaBoost algorithms perform better than the two other ones. The algorithm to choose depends on the data it is used on.

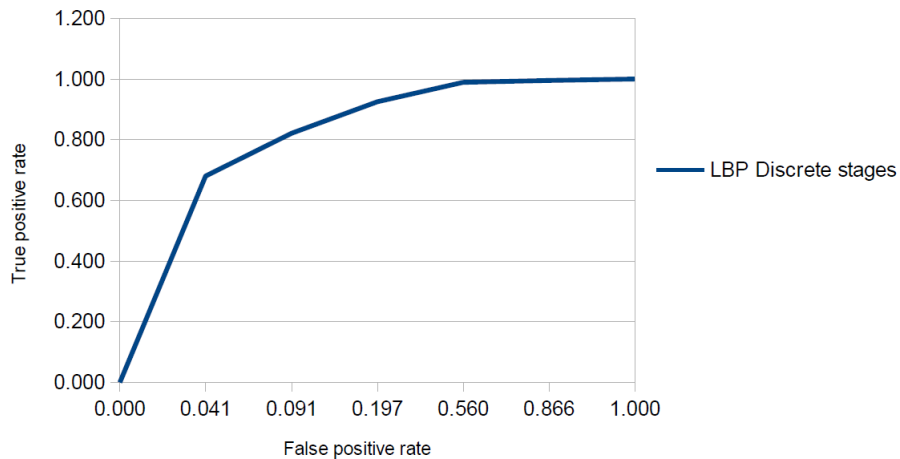


Figure 3.13: Detection results for the LBP features and the Discrete Adaboost algorithm with a varying number of stages, from 10 to 20. With more stages, less mistakes are made, as the last stages are supposed to deal with the most complex samples, but some faces are also not detected.

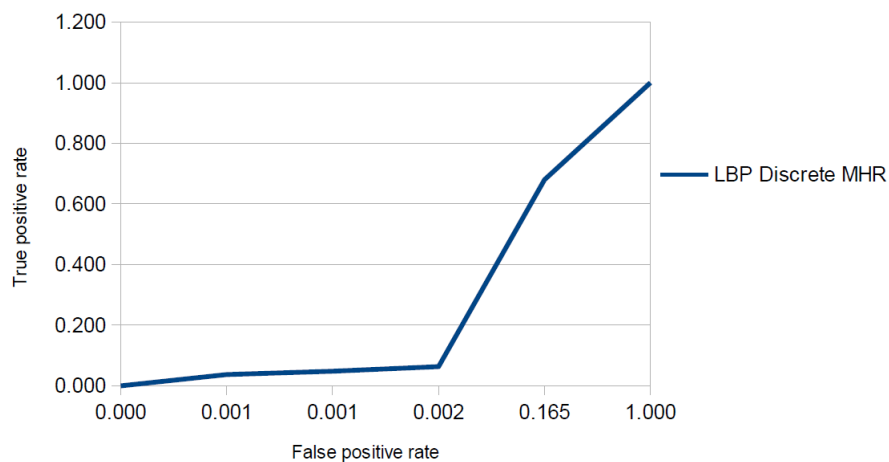


Figure 3.14: Detection results for the LBP features and the Discrete AdaBoost algorithm with a varying minimal hit rate (MHR), from 0.6 to 0.995. A high MHR gives a high true positive rate but also a high false positive rate, as the training algorithm accepts more errors in order to fulfill the MHR requirement.

careful trade off between detection rate and amount of errors we consider to be acceptable.

To conclude, in this result section we saw, through our tests, the role of each of the parameters and we were able to make a choice about optimal values. In this section, we only presented the results for LBP features, as they are faster to train. However, we intend to conduct more tests in the future.

3.6 Improvements

3.6.1 On the data

The AFLW database presents many advantages, as it is a large database and as it contains faces in the wild. This database covers a large range of possible faces in terms of age, gender and ethnicity. It is also annotated and the pose estimation that is computed from the landmarks is pretty accurate.

However, we may think about what type of face we will expect to have in CDS. Although this dataset is probably the closest and the biggest one for our case, we may think about feeding the training set with photos directly coming from tagged pictures in CDS and run the training from time to time in background.

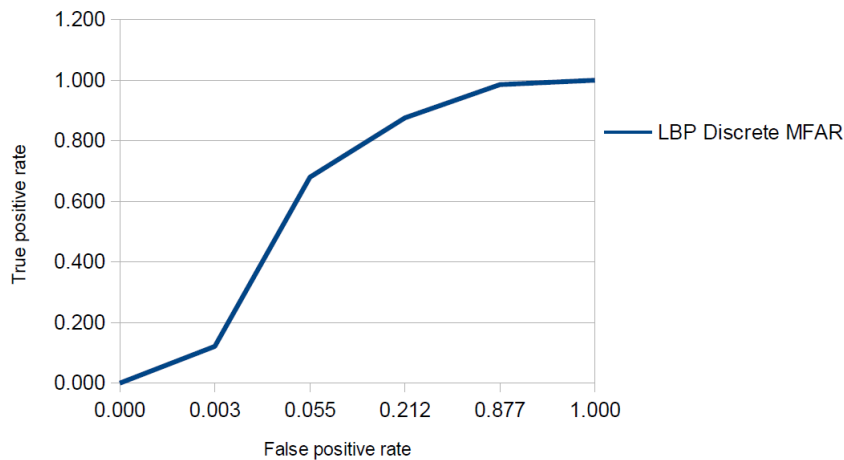


Figure 3.15: Detection results for the LBP features and the Discrete AdaBoost algorithm with a varying maximal false alarm rate (MFAR), from 0.6 to 0.995. A low MFAR gives a lower false positive rate but also lowers the true positive rate.

As the training API in OpenCV only needs a description file with the image paths and face positions, this is pretty easy to set up a task to run from time to time. Especially, the boosting algorithms are fairly resistant to over fitting.

The only problem for this idea is that we need to find a way to get the orientation of the face that has been tagged because we need to train the six classifiers on the right face orientations. A lot of research has been done on pose estimation. Inventive methods such as the one described in [13] could be integrated into our system. When we get the precise orientation of the face, we assign the corresponding face to one of the six classifiers to be learned. The classifiers are then periodically computed again with the new data at a scheduled time.

For the problem of false detections, regions of the image that have been taken for faces but that are not faces, the problem to add them to the next training is more complicated. Integrating misclassified samples to the future training set would be more difficult, as we are not sure about whether the user didn't take into account a given square because he didn't want to fill in the person's name or because the detection didn't point out a person's face. Adding untagged squares to the negative set would be really risky.

3.6.2 On the algorithm

A lot of algorithms for detection exist. They can be:

- Feature based: these algorithms search for relevant features such as eyes, nose and mouth.
- Template based: these algorithms try to fit a given model to a potential face.
- Appearance based: the Viola Jones algorithm comes from this category which is about searching for a face by comparing rectangles at different scales

Some of the detection algorithms are really complex and maybe more precise. However, we had the constraint to be able to run in real time. If we had much more computational power, we could consider other approaches. Some of them more suited in the context of face recognition.

Active Appearance Models, for example, are extremely popular because they can handle large variations in expression and pose. However they require to be initialized not too far from the face to detect in order to converge. Thus, if we have no idea of where the face is, we have to do an exhaustive search and make the model converge, which is quite slow. We tested it at the beginning of our investigation for the best algorithm. This model is interesting because it fits a face shape with landmark points over the face and these points can then be used for face recognition. Sadly, if we don't have a clue of the face position and if we have to do an exhaustive search, this algorithm is not suitable for real time.

The method described in [13] is also interesting as it combines face detection, pose estimation and landmark localization. This could be combined with face recognition assuming higher computational power than we have now.

Chapter 4

Face recognition

4.1 Introduction

In this chapter, we will discuss face recognition from images uploaded and tagged by various users.

The purpose of this face recognition system is to simplify the user's task. It has two goals. The first one is for when a user uploads a large collection of images belonging to the same event. In these pictures, there might be the same persons several times and it could be great if the user only had to enter these person's name for the first image and that the rest of the images get tagged automatically. Of course, this is just a suggestion and the user would have to confirm the proposed tags.

The second purpose of our recognition system is to learn from each tag that is entered by a user and to be able to suggest a name for a new face even if it is the first time it appears in the collection being tagged.

So, what we have to deal with here are completely unconstrained conditions. People might not face the camera, the illumination should vary a lot, their facial expression should vary also and we might only have a small set of faces corresponding to one person.

The most famous algorithms for face recognition only use frontal faces and relatively small variation in illumination. They also need an acceptable number of faces per person in order to achieve a good recognition rate. Another requirement for our purpose is that the recognition has to be done in real time. As soon as the user starts to tag a picture, we should be able to add suggestions on top of it.

So, for the typical situation when someone starts to tag a new collection of pictures and we don't have enough information about the persons already tagged in this collection (maybe just one face per person), we should use

another method than the usual machine learning algorithms.

Our idea is to do the tagging suggestion using two algorithms. The first one uses only the data from the current collection, works from one single face per person and the person in the picture is not required to face the camera. The second uses the data collected from many tags and only takes frontal faces into account. This algorithm only learns persons that have been tagged massively.

We will first present our method to recognize a person inside the same collection (the same event/day) without requiring previous training.

Then, we will continue with recognition algorithms based on learning from many frontal faces of the same person.

We will then describe how we combined the two approaches to have a full recognition process working for frontal and non frontal faces.

4.2 Recognition inside a collection

This section describes an algorithm for recognizing someone that has been tagged maybe only once and not necessarily while facing the camera. We wanted here an algorithm that is fast and has a high recognition rate (but also potentially makes some mistakes). The goal is to ease the user's task and it is faster to discard a bad recognition than to have to type the whole name for a person that wasn't recognized. That's why we implemented a simple algorithm that is not extremely precise but works well in the case of looking inside the same collection of pictures.

We will start by describing what we expect to have in the context of a user uploading a collection of several images. We will then expose the algorithm we implemented based on that context. We will then present the results we obtained.

4.2.1 Assumptions

For this algorithm, we position ourselves in the case where a user uploads a collection of photos. These photos are, more than 99% of the time, related to a same event and have been taken on the same day. This means that people, with a high probability, are wearing the same clothes on every picture. That information is really useful and we must take advantage of it, especially if the persons are not facing the camera. In the case of someone not facing the camera, we still want to recognize him, that's why we don't rely on the face detectors at all for this algorithm.

4.2.2 Algorithm

This algorithm processes candidates as follows: it starts from a face that has just been tagged and from the image it comes from. It first finds the n best possibilities for a face to recognize and then compares the clothes to choose the best option among the possible faces. With this method, we can use a simple algorithm for finding the n possible options for the face and we can then confirm the best choice comparing the clothes.

Face search

The face search is done greedily with color histograms. We have an already tagged face represented as a rectangle in a bigger image. We first compute the color histogram of the head and search for it in the images of the collection that we have to analyze.

The first step is to find the n most probable regions of the inspected image that are close to the face we are looking for in terms of color histograms. The distance used is the χ^2 distance:

$$d(H_1, H_2) = \sum_I \frac{(H_1(I) - H_2(I))^2}{H_1(I)}$$

Where H_1 and H_2 are the two color histograms to compare.

This distance has proven to give the best results in our case. We compute the histograms in the HSV colorspace and compute the histogram on the Hue channel only, as this channel accounts for the color distribution while being independent of lighting and saturation.

So, this first step gives us n sub-regions in the image that may contain the face that we are searching for. They are the best possible choices in terms of color. The next step is to actually circle (or refine the location of) the face potentially contained in that sub-region.

To do that, we use an algorithm made for face tracking in video, called Camshift for Continuously Adaptive Mean Shift and introduced by Gary R. Bradski [14]. This algorithm uses color histograms for skin to compute the probability of having a face at each position in the image. It uses the Mean Shift algorithm to shift to the best position for the face and then adjusts the scale and orientation of the face.

More in details, the skin probability at each pixel location is computed using the back projection of a color histogram computed in the HSV color space. The Hue Saturation Value space separates the color information (hue) from the saturation and brightness. The histogram is computed on the hue channel and, in our case, the initial histogram used to build face probability is

computed using the previously tagged face (the one we are looking for in the new image). The computed histogram is used as a model, or as a lookup table, to compute the probability for each new pixel to be part of a face.

After computing the probability distribution for the whole sample using the lookup table, the next step is to use the Mean Shift algorithm [15] to shift towards the most probable position. The Mean Shift algorithm is a non parametric algorithm used for finding the modes of a density function. It is initialized with a window size and a location. Then, at each iteration, it computes the mean of the search window then shifts the center of the search window to the mean location. It repeats until the window location doesn't change anymore. The final position corresponds to a local maximum.

In our case, the mean is computed, first, by calculating the zeroth and the first moment:

$$M_{00} = \sum_x \sum_y I(x, y)$$

$$M_{10} = \sum_x \sum_y xI(x, y); M_{01} = \sum_x \sum_y yI(x, y)$$

Where $I(x, y)$ is the probability value at position (x, y) , x and y picked in the search window's range.

The new location for the search window's center is then:

$$x_c = \frac{M_{10}}{M_{00}}; y_c = \frac{M_{01}}{M_{00}}$$

The Mean Shift algorithm gives us the face's position with highest probability. To find its size and orientation, we use the Camshift procedure.

After running the Mean Shift algorithm, we use the second moments to compute the size and orientation of the face:

$$M_{20} = \sum_x \sum_y x^2 I(x, y); M_{02} = \sum_x \sum_y y^2 I(x, y)$$

The orientation is:

$$\theta = \frac{\arctan \left(\frac{2 \left(\frac{M_{11}}{M_{00}} - x_c y_c \right)}{\left(\frac{M_{20}}{M_{00}} - x_c^2 \right) - \left(\frac{M_{02}}{M_{00}} - y_c^2 \right)} \right)}{2}$$

Then, let

$$a = \frac{M_{20}}{M_{00}} - x_c^2,$$

$$b = 2 \left(\frac{M_{11}}{M_{00}} - x_c y_c \right),$$

$$c = \frac{M_{02}}{M_{00}} - y_c^2$$

The length l and width w are thus:

$$l = \sqrt{\frac{(a+c) + \sqrt{b^2 + (a-c)^2}}{2}}$$

$$w = \sqrt{\frac{(a+c) - \sqrt{b^2 + (a-c)^2}}{2}}$$

At the end of that part of the process, we have n oriented ellipses that circle the n most probable positions for the face we are looking for in the new image. The next step is to make a decision among these n possibilities using clothes comparison.

Clothes comparison

Now that we have a more precise location and size for the face, we can look at the clothes in order to choose the best possibility among the n choices. The clothes comparison works as follows: we first build a torso model, a model that says where the upper part of a person's body is regarding to his head. Then we use this model as a mask for each face's region and we compare, again, color histograms.

For building the torso model, the shape and position of the upper body regarding the head, we implemented a simple interface for circling heads and bodies and used it to annotate approximately 170 persons from various pictures. We averaged the result using the face area to scale each torso at the same size. For each sample, there is a 1 in the corresponding pixel position if the pixel is considered as being part of the upper body. If we sum and average the torso we selected, we obtain a matrix of values between 0 and 1. From this model, the first idea would have been to compute a weighted color histogram as described in [16] and shown in picture 4.1. But we need to compute a greedy search for the head, a refinement with Camshift then a torso comparison for n possibilities and we want the whole process to be real time. That's why we decided to threshold the averaged torso, set to 0 all the values below that threshold and to 1 all the values above. The color histogram is computed from the explored image by taking into account the pixel positions that correspond to a 1 in the torso mask.

The torso color histograms are also computed in the HSV space and compared using the χ^2 distance. The distances for face and torso are combined and the best among the n possibilities is chosen at the end.

Algorithm 1 sums up the whole recognition process.

Algorithm 1 Recognition process for a face inside a collection

Input: *modelImage*, *tagPosition*, *torsoMask*, *imagesCollection*, *scales*, *n*

Output: *tagPositions*

```

    # first step: compute the histograms for the tag we are looking for
    faceModel  $\leftarrow$  faceHistogram(modelImage, tagPosition)
    torsoModel  $\leftarrow$  torsoHistogram(modelImage, tagPosition, torsoMask)
    bestNPositions  $\leftarrow$  []
    for all image in imagesCollection do
        for all scale in scales do
            for all subWindow do
                faceHist  $\leftarrow$  faceHistogram(image, subWindow)
                distance  $\leftarrow$  chiDistance(faceHist, faceModel)
                if distance < max(distances(bestNPositions[image])) then
                    putIn(bestNPositions[image], subWindow)
                end if
            end for
        end for
        tagPosition[image]  $\leftarrow$  []
        bestDistance  $\leftarrow$  MAX_INT
        for all position in bestNPositions[image] do
            ellipse  $\leftarrow$  camshift(faceModel, image, subWindow)
            torsoHist  $\leftarrow$  torsoHistogram(image, ellipse, torsoMask)
            distance  $\leftarrow$  chiDistance(torsoHist, torsoModel)
            distance  $\leftarrow$  combine(distance, distanceFace(position))
            if distance < bestDistance then
                bestDistance  $\leftarrow$  distance
                tagPosition[image]  $\leftarrow$  ellipse
            end if
        end for
        if isNotAcceptable(bestDistance, threshold) then
            tagPosition[image]  $\leftarrow$  []
        end if
    end for
    return tagPosition

```

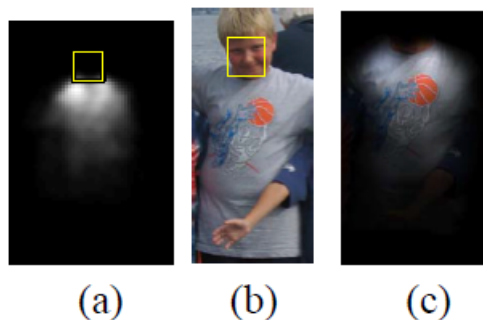


Figure 4.1: Picture from [16] (a) Spatially weighted mask shown relative to a face detection. (b) Cut out of a person with face detection superimposed. (c) Weight mask superimposed over the image.

4.2.3 Results

In this section, we will show an example of pictures taken from CDS. We tagged a first picture from a collection then we ran the algorithm on another one from the same collection. Building an entire evaluation set with annotated photos would have taken too much time. Here we just present a few examples of photos we tested during our implementation process.

We also show the intermediate results of the algorithm. For each tag in the picture, we show what were the n best possibilities for the face position and what the Camshift algorithm outputted. We show then which of the possibilities had the lowest distance.

Figure 4.2 shows the picture we used for tagging and the one we used for testing.

In figures 4.3, 4.4, 4.5, 4.6, 4.7 and 4.8 are presented the 10 best possibilities for each tagged face of figure 4.2(a) refined with an ellipse using the Camshift algorithm. The best possibility in terms of torso and face distance to the model is drawn in green.

We can see that, for the tag Mark (figure 4.3), the face of this person is not among the 10 best possibilities outputted by the function. Instead, we have the head of his colleague that is detected but, since the clothes don't match, the best choice (in green) results in something completely irrelevant. Fortunately, the resulting distance is pretty high compared to the true recognitions. So we can discard this recognition using a threshold learned from several examples. The tag Joe (figure 4.4) has the same problem.

For the tag Liviu (figure 4.5), the search for the 10 closest regions in terms of color histogram gives results that are not centered at all on faces. Thus the Camshift algorithm, which is supposed to refine ellipses on faces, tends to

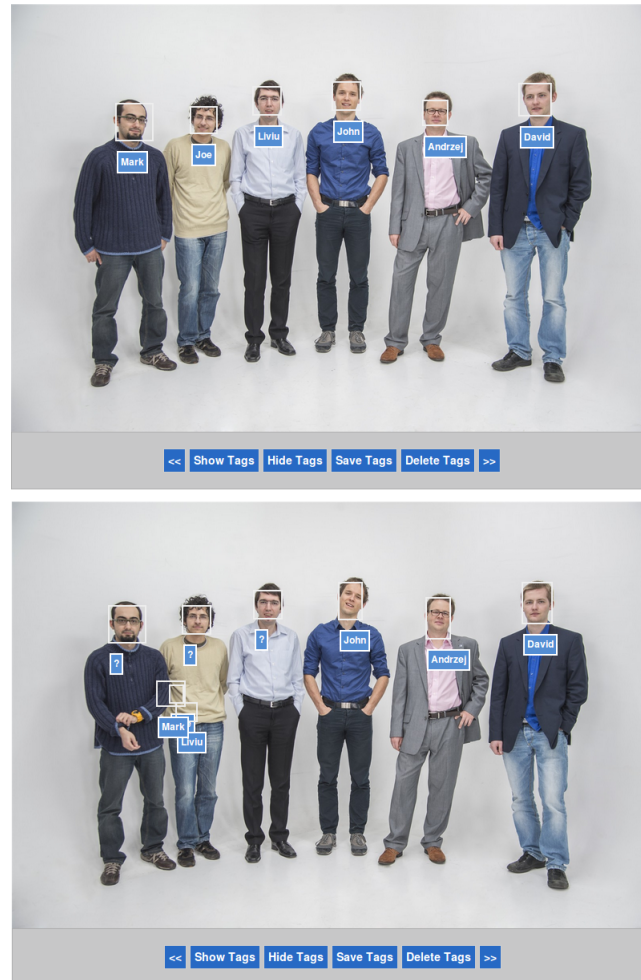


Figure 4.2: First recognition example using clothes. top: photo tagged by hand, bottom: resulting suggestions, retrieved without any thresholding.



Figure 4.3: Possibilities for the tag Mark. The model has been taken from figure 4.2 and the ellipses show the 10 best choices for the face color histograms, refined with Camshift. The green ellipse attests for the best choice after clothes comparison. Note that Mark's face wasn't in the 10 best possibilities so the 1st choice is completely irrelevant. It is discarded after thresholding.

shrink the ellipses because it cannot find the face position. In this case too, the resulting distance between faces and torsos histograms is high enough compared to the one found for true recognitions and we can easily discard it using a comparison with a threshold.

For the three last tags, we can see that several different faces have been outputted as possible candidates for the face we are looking for. For Andrzej (figure 4.7), the closest face, if we use only color histograms for the face and no clothes comparison, is not his face but his colleague's face. However, the right face is among the 10 closest histograms and, if we take the 10 best possibilities and if we compare the clothes between the tagged picture and the new one, the face that comes as the best choice is the one we want.



Figure 4.4: Possibilities for the tag Joe. The model has been taken from figure 4.2 and the ellipses show the 10 best choices for the face color histograms, refined with Camshift. The green ellipse attests for the best choice after clothes comparison. As for Mark, Joe's face is not among the 10 best choices, thus the best possibility is completely irrelevant and the resulting distance to the model is pretty high. It is discarded.

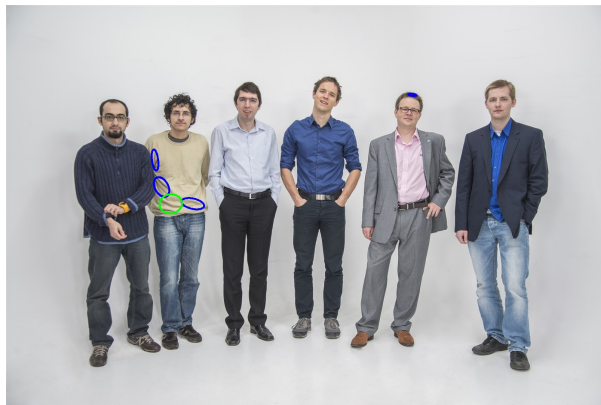


Figure 4.5: Possibilities for the tag Liviu. The model has been taken from figure 4.2 and the ellipses show the 10 best choices for the face color histograms, refined with Camshift. The green ellipse attests for the best choice after clothes comparison. Here, the 10 best possibilities for the color histograms are not faces. The Camshift algorithm outputs shrunk ellipses, as it cannot converge towards a face position. This suggestion is discarded before retrieving the suggestions to the tagging interface.



Figure 4.6: Possibilities for the tag John. The model has been taken from figure 4.2 and the ellipses show the 10 best choices for the face color histograms, refined with Camshift. The green ellipse attests for the best choice after clothes comparison. John has been recognized. There was two serious candidates: John and David but the clothes comparison allowed to choose the right face.



Figure 4.7: Possibilities for the tag Andrzej. The model has been taken from figure 4.2 and the ellipses show the 10 best choices for the face color histograms, refined with Camshift. The green ellipse attests for the best choice after clothes comparison. There was 4 faces among the 10 best positions and Andrzej's face was not the choice with smallest distance if we only consider the face histograms. Fortunately, the clothes comparison allows us to find the true result.



Figure 4.8: Possibilities for the tag David. The model has been taken from figure 4.2 and the ellipses show the 10 best choices for the face color histograms, refined with Camshift. The green ellipse attests for the best choice after clothes comparison. David has been detected as the most probable face after clothes comparison between the 10 choices and the clothes of the model.

So this method has the advantage to be fast enough to run in real time while still achieving good recognition rates. It gets along well with our goal of recognizing a lot of faces, even though sometimes we also make some mistakes.

4.3 Recognition from previous tagging

In this section we will describe several algorithms that can be used to recognize persons from several frontal faces. We tested them using the ORL database [17] containing faces with small light variations and facial expression variations. The algorithms are grouped by characteristics.

We will first describe all the algorithms we tested then we will present our results and our final implementation choice.

4.3.1 Context

As said before, we are working on faces taken "in the wild". That means that we are expecting sometimes big changes in lighting, facial expressions and facials details (glasses, beard, hair style, etc). This context is really challenging as we would need a lot more data for each person to be described properly. We also need the recognition process to be pretty fast. That requirement made us to drop some robust, efficient algorithms because they were too slow to run in real time.

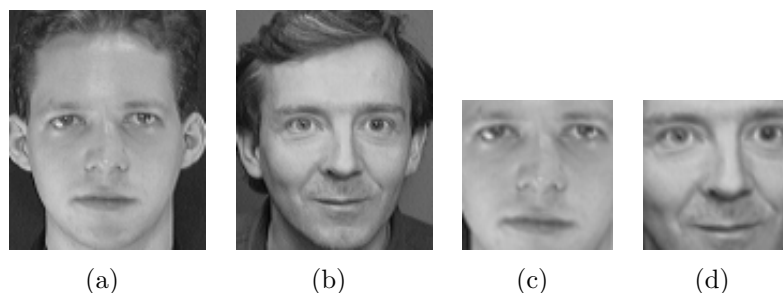


Figure 4.9: An example of face normalization on the AT&T faces. (a) and (b) are the faces before normalization and (c) and (d) the normalized faces. The eyes have been detected and the faces have been scaled, cropped and rotated so that the eyes are at the same position in each sample.

We will present algorithms that are fast enough for our case and compare them in terms of recognition rate.

4.3.2 Algorithms

PCA based

The algorithm called Eigenfaces [18], first developed by Sirovich and Kirby, is one of the first recognition algorithm that was elaborated and that actually achieved good results under constrained conditions.

It uses the whole image of the face as a feature and performs a dimension reducing algorithm to project the data into a lower dimension subspace. The recognition is then done by finding the nearest neighbor in this subspace.

The data used is the face, represented as a 2D matrix of grayscale values. The first step of the algorithm is to align, crop and scale all the faces. Aligning means putting the eyes and mouth at the same position and orientation for all the faces. To do that, an eye and mouth detector is used. These detectors work the same way as the face detector described in the previous section but they are trained with eyes and mouth samples. See an example of face normalization in figure 4.9.

After aligning all the data, each face matrix is transformed into a vector by concatenating all the rows into one row. If the image size was $n \times n$, the vector's dimension is then n^2 . The values are the grayscale pixel values.

The next step is to project these vectors into a lower dimension subspace. The Principal Component Analysis method is used. The idea of this method is that only a few dimensions are meaningful in terms of data variance. By using an orthogonal transformation, it gets a set of possibly correlated variables to

a set of linearly uncorrelated variables, the principal components.

The PCA method can be seen also as a method allowing to discard noise while preserving meaningful information.

The PCA algorithm is performed as follows.

First the average vector is subtracted from each vector.

Then, if we call X the matrix composed of the faces vectors (with mean subtracted), one per column, the covariance matrix is:

$$S = XX^T$$

From this matrix, we compute the eigenvalues λ_i and eigenvectors ν_i such that:

$$S\nu_i = \lambda_i\nu_i$$

Then, for a desired subspace of dimension k , the k eigenvectors corresponding to the k largest eigenvalues are taken and the data is projected onto this subspace. In the case of faces, the eigenvectors are called eigenfaces.

The principal components are then:

$$T = XW$$

with $W = (\nu_1, \nu_2, \dots, \nu_k)$.

So all the faces are represented now by a vector of dimension k . To recognize a new face, it is first projected onto the basis of eigenfaces then it is compared to the other faces. The Euclidean distance is used to compare faces and the new face to recognize is compared against the mean face of each person. The queried face is then labeled with the person with the closest mean.

If we sum up the algorithm, the steps are:

- Alignment and scaling of all the faces.
- Flattening of the matrices into a vector, mean subtraction, and construction of the X matrix, containing one vector per column.
- Computation of the eigenvalue decomposition.
- Build of the subspace composed of the k eigenvectors corresponding to the k largest eigenvalues.
- Projection of each of the faces vectors onto that subspace.
- Computation of the mean face for each person.

In terms of implementation, OpenCV has a version of the Eigenfaces algorithm which is available. However, the way it is implemented is not suitable for a high number of faces. The search for the nearest face is done linearly and the function for saving the data into an xml file doesn't save as much as we would like (for example, we wanted to add the tag id, the one from our database). We thus started from the OpenCV implementation and modified it to meet our goals. The nearest neighbor search has been modified to perform a tree search and the tree model is saved in the xml file. We also extended the search to retrieve the n nearest neighbors.

LDA based

The problem of the PCA method described below is that it finds the directions that maximize the variance for the whole data (it maximizes the global variance). This is a problem, for example, if a person has a set of faces that were taken under highly changing illumination. This means that these faces will be spread in the subspace. So, what we would like is a method that, instead of maximizing global variance, maximizes the variance between classes (persons).

The Linear Discriminant Analysis method, designed by Sir R. A. Fisher [19], tackles this problem. It maximizes the variance between classes while minimizing the variance inside classes. It was first used for face recognition by Belhumeur, Hespanha and Kriegman [20]. Figure 4.10 illustrates a case where LDA and PCA perform a completely different dimension reduction.

Let $\{X_1, X_2, \dots, X_c\}$ be the c classes, each one corresponding to one person, each one composed of N_i sample images $\{x_1, x_2, \dots, x_{N_i}\}$, $i = 1, 2, \dots, c$.

The between-class scatter matrix is defined as:

$$S_B = \sum_{i=1}^c N_i (\mu_i - \mu)(\mu_i - \mu)^T$$

and the within-class scatter matrix as:

$$S_W = \sum_{i=1}^c \sum_{x_k \in X_i} (x_k - \mu_i)(x_k - \mu_i)^T$$

With μ_i the mean image of class i and μ the total mean.

If S_W is nonsingular, the matrix, with orthogonal columns, that maximizes the ratio of the determinants of the between-class scatter matrix to the within-class scatter matrix of the projected samples is:

$$W_{opt} = \arg \max_W \frac{|W^T S_B W|}{|W^T S_W W|} = [w_1, w_2, \dots, w_m]$$

where the w_i are the generalized eigenvectors of S_B and S_W associated to the m largest eigenvalues λ_i :

$$S_B w_i = \lambda_i S_W w_i; i = 1, 2, \dots, m$$

There are at most $c - 1$ eigenvalues so $m < (c - 1)$.

However, in the case of face recognition, the S_W matrix is singular because of the fact that the number of dimensions is much greater than the number of training samples N . To overcome this problem, the data is projected onto a lower dimensional subspace using PCA. The number of dimensions is reduced to $N - c$.

W_{opt} is now:

$$W_{opt}^T = W_{fld}^T W_{pca}^T$$

with

$$W_{pca} = \arg \max_W |W^T S_T W|$$

$$W_{fld} = \arg \max_W \frac{|W^T W_{pca}^T S_B W_{pca} W|}{|W^T W_{pca}^T S_W W_{pca} W|}$$

and S_T the total scatter matrix:

$$S_T = \sum_{k=1}^N (x_k - \mu)(x_k - \mu)^T$$

So the Fisherfaces method learns a transformation matrix that is specific to the classes and, if the data used to train is good enough, it can overcome problems such as illumination changes and expression changes. By good enough, we mean that, if we want to recognize someone under all types of illuminations, we should have training samples for this person that have been taken under different lighting conditions. This implies that we need a potentially big amount of samples to fully describe a person.

The steps for training and recognizing faces with the Fisherfaces method are the same as for the Eigenfaces method. What changes is the subspace construction.

LBPH

The Local Binary Patterns histogram method [9] uses texture and region information to compare faces. The features used for texture information are LBP (see section 3.2.2 for more information) extended to use neighborhood of different sizes. They are characterized by a number P of sampling points, a Radius R and are called circular LBP (see figure 4.11). As for the basic

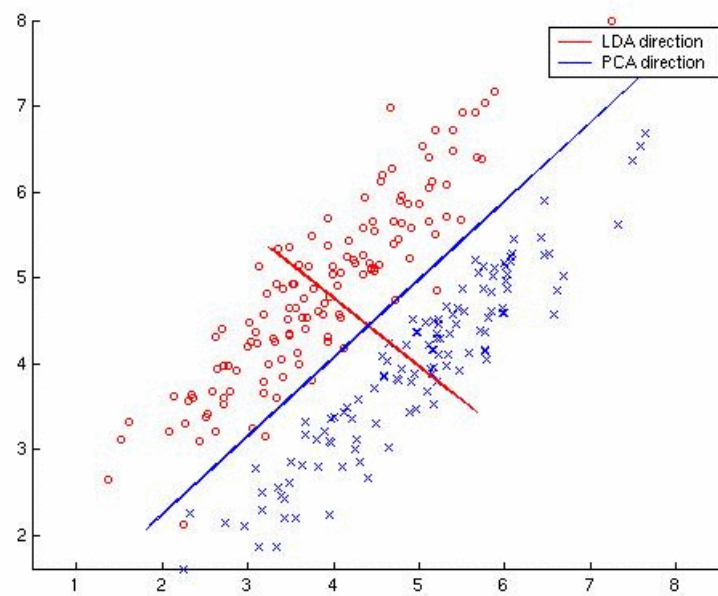


Figure 4.10: Example where LDA finds a better subspace than PCA. The blue and red dots represent two different classes. The PCA projects points onto the blue line, which results in a really bad class separation. LDA uses the between classes variance to find the red line and projects the points on it. The two classes are better separated.

LBP features, the values of the P points are thresholded by the center value. It results in a vector, a binary number of length P , for each pixel.

For this algorithm, only uniform LBP are used, the ones that contain at most two bitwise transitions from 0 to 1 or vice versa. Each vector of length P that is considered as being uniform receives a label, the remaining non uniform patterns are labeled with the same label.

Then the histogram of the labels is computed. For $f_l(x, y)$, the labeled image:

$$H_i = \sum_{x,y} I \{f_l(x, y) = i\}, i = 0, \dots, n - 1$$

where n is the number of labels and

$$I \{A\} = \begin{cases} 1 & \text{if } A \text{ is true} \\ 0 & \text{if } A \text{ is false} \end{cases}$$

This histogram describes the local information of the image such as edges, dots, uniform areas, etc. Then the spacial information is taken into account by dividing the image into m regions R_0, R_1, \dots, R_{m-1} . The final histogram is:

$$H_{i,j} = \sum_{x,y} I \{f_l(x, y) = i\} I \{(x, y) \in R_j\}, i = 0, \dots, n - 1, j = 0, \dots, m - 1$$

To classify a new image, a nearest neighbor approach is used, along with the χ^2 distance. A complex machine learning method is not used here, as this method is meant to be used even when there is only one face to characterize one person in the training set.

If we sum up the steps of this method, we have to:

- Align the faces the same way we did for the Eigenfaces and Fisherfaces methods.
- Compute the LBP features at different positions in the samples.
- Assign to each different feature which is uniform a label.
- Split the samples into regions and build a histogram of the labels inside each region.
- Use the computed histograms to characterize each face and recognize a new face by comparing its histogram to the model ones using the χ^2 distance.

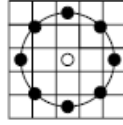


Figure 4.11: A circular neighborhood with 8 points and a radius of 2. The points are bilinearly interpolated if not centered on a pixel. The LBP response is computed as follows: for each point on the circle, we compare its value with the center one. If it is bigger, we output 1 and 0 otherwise. The result is then a binary vector of length 8.

Bayesian method

The Bayesian Intra-personal/Extra-personal Image Difference Classifier [21] is an algorithm using probabilistic measure of similarity in contrast to method based on standard Euclidean L2 norms.

PCA and LDA are pretty famous methods, they are based on projecting data onto a subspace such that points from the same class are relatively close compared to points from another class. But the problem of these methods is that they assume that there exists an optimal solution, that we can find a projection such that the classes don't overlap.

As this is not always the case, Moghaddam, Wahid and Pentland defined a new concept, intra-personal and extra-personal differences between two face images. The intra-personal variation stands for variations in appearance for a same person (i.e different expressions, lighting, etc) and the extra-personal variation stands for differences in identity.

The algorithm works on intensity difference between pair images and computes the probability that a difference image is characteristic of common variations of appearance for the same person and the probability that this difference comes from the fact that there are two different persons.

So the problem of classifying a face into a class for a PCA or LDA algorithm becomes a binary problem for face differences. A new face to recognize is compared to known faces. The signed pixel intensity difference is computed for each pair of images and this pair is then classified either into the intra-personal class, meaning that the two images both belong to the same person, or into the extra-personal class, meaning that they are not from the same person.

This algorithm relies on the assumption that the intra-personal and extra-personal difference images form distributions that are approximately Gaussian.

The parameters for the distributions are computed during a training phase.

For testing, the image to recognize is compared to each image of the training set.

If we name Ω_I the intra-personal class, Ω_E the extra-personal class and $\Delta = I_1 - I_2$ the difference image, the similarity between two images is defined as the probability of the difference image to belong to the intra-personal class:

$$S_{MAP}(I_1, I_2) = P(\Omega_I|\Delta)$$

with MAP standing for Maximum A Posteriori classification.

Using Bayes rule, we have:

$$P(\Omega_I|\Delta) = \frac{P(\Delta|\Omega_I)P(\Omega_I)}{P(\Delta|\Omega_I)P(\Omega_I) + P(\Delta|\Omega_E)P(\Omega_E)}$$

So the similarity for each difference image is computed and, for the best one, the class of the corresponding image of the training set is retrieved.

Another similarity measure defined later by Moghaddam, Wahid and Pentland is the Maximum Likelihood (ML) measure, which uses only the intra-personal class probability and ignores the extra-personal probability:

$$S_{ML} = P(\Delta|\Omega_I)$$

For the *MAP* similarity measure, the likelihoods $P(\Delta|\Omega_I)$ and $P(\Delta|\Omega_E)$ have to be estimated. However the problem is that the difference vector Δ has a high number of dimensions ($\Delta \in R^N$ and $N > 10^4$ usually) so it is difficult to compute reliable second order estimates of the likelihoods because of the insufficient number of observations (number of difference images) compared to the vector's dimension.

A first step is thus to use PCA to reduce the number of dimensions from N to a chosen number M , $M \ll N$.

PCA is run twice for the MAP similarity. Once for a set of intrapersonal image differences and once for extra-personal image differences. It is run only once for the ML similarity measure.

After the PCA, we obtain a projection matrix Φ_M and a vector of eigenvalues λ . The probability, approximated as a Gaussian, without PCA would be:

$$P(\Delta|\Omega) = \frac{\exp\left(-\frac{1}{2} \sum_{i=1}^M \frac{y_i^2}{\lambda_i}\right)}{(2\pi)^{M/2} \prod_{i=1}^M \lambda_i^{1/2}} \cdot \frac{\exp\left(-\frac{1}{2} \sum_{i=M+1}^N \frac{y_i^2}{\lambda_i}\right)}{(2\pi)^{M/2} \prod_{i=M+1}^N \lambda_i^{1/2}}$$

But, by projecting the data onto the PCA space, the right part of the product is unknown.

Still, this right part needs to be taken into account and a way to do that is to replace the previous equation by:

$$P(\Delta|\Omega) = \frac{\exp\left(-\frac{1}{2} \sum_{i=1}^M \frac{y_i^2}{\lambda_i^2}\right)}{(2\pi)^{M/2} \prod_{i=1}^M \lambda_i^2} \cdot \frac{\exp\left(-\frac{1}{2\rho} \epsilon^2(\Delta)\right)}{(2\pi\rho)^{(N-M)/2}}$$

With y_i the difference image vectors projected onto the PCA subspace ($y = \Phi_M^T \Delta$) and ρ :

$$\rho = \frac{1}{N-M} \sum_{i=M+1}^N \lambda_i$$

which can be computed from the PCA transformation, and $\epsilon^2(\Delta)$:

$$\epsilon^2(\Delta) = \|\tilde{\Delta}\|^2 - \sum_{i=1}^M y_i^2$$

the residual reconstruction error, with $\|\tilde{\Delta}\|^2$ the square of the vector length before it is projected onto the PCA space. $\sum_{i=1}^M y_i^2$ is the length after projection.

The last step is to take care of the recognition step. As said before, the comparison between the input image and the images of the database is done one by one, as the difference image is computed for each pair. However, this is not suitable for a large collection of images. In [22], it is suggested to use the Eigenfaces method to first select the n best possible images in the database and then use the algorithm described here to find the best image.

We thus implemented this combination of the two methods. The Bayesian method was partly implemented by Marcio Luis Teixeira for the CSU study [21]. However, we had to do a lot of modifications to make it usable in real time and inside our complete process of tagging. For the Eigenfaces method, we extended the OpenCV class in order to have a method that retrieves the n first choices, along with the tag id. We wrapped the resulting C classes to use them in the Python code.

The steps for this method are:

- Normalize the faces as for the previous methods.
- Compute the differences images using the training set. Two sets of difference images are computed: images for intra-personal difference and images for inter-personal difference.
- Compute a dimension reduction for each of the two sets of difference images.

- Estimate the likelihoods $P(\Delta|\Omega_I)$ and $P(\Delta|\Omega_E)$ using the formulas above.

For assigning the most probable class to a new face, the process is:

- Find the n best possibilities for the face's class using the Eigenfaces method.
- Compute the difference images between the face to recognize and the n faces retrieved by the previous step.
- For each difference image, compute its probability to be characteristic of an intra-personal difference and its probability to be characteristic of an inter-personal difference.
- Output the model image for which the intra-personal probability was the highest.

4.3.3 Results

In this section, we will present our results for the different methods we implemented, the database we used and the parameters we varied.

Evaluation framework

To evaluate the recognition performance, we used a database called AT&T. The AT&T database, also called the ORL Database of Faces is a database of faces taken at the Olivetti Research Laboratory in Cambridge [23]. It is composed of portraits from 40 persons, each person having 10 pictures of him/her. The persons are facing the camera but their expression, the lighting and the time the pictures have been taken change from one photo to another. We chose this database to test our algorithm as it is free and widely used for evaluating face recognition algorithms. For other studies on other databases, see, for example, the one done by the Colorado State University [24] on the FERET database [25].

Results

We first present the results for the Eigenfaces and Fisherfaces models, for which we varied the number of dimensions for the subspace (see figure 4.12). We can see that a small number of dimensions results in a higher error rate for the Fisherfaces method. This is due to the fact that the more we decrease the number of dimensions, the more we lose details. The number of

dimensions has less impact on the Eigenfaces method. This can be explained by the fact that this method builds the subspace regardless of the samples classes. A good number of dimension to choose is a number of the same order as the number of classes.

For the LBPH method, we varied the radius for the computation of the Circular Local Binary Patterns (figure 4.13) and the grid size (figure 4.14). The optimal radius for the LBPH method should vary depending on the sample size. For our study, the optimal one is the one at a distance of 6 from the center. This distance corresponds to the point at which the information extracted from the features is not too local, not subject too much to insignificant variations, but still close enough so that the feature response brings some useful discriminating information.

The grid size specifies the number of cells we want to split the sample in horizontally and vertically. This is used for the histogram computation which is done per region (per cell). A small grid size (i.e. large cells) give high error rates, as the resulting LBP description is not local enough. On the other hand, splitting too much the sample is also not good, as it will result in a histogram which is too dependent on local variations and doesn't attest for global arrangement of the features.

For the Bayesian method, see the CSU study for a full test of each parameter. The tests on the ORL database give an error rate of 0.33. However, this methods achieves pretty good results compared to the other ones in the case where the number of faces per person is low. We chose this method along with the Eigenfaces one for finding the n possibilities.

4.3.4 Propagation to non frontal faces

In this section, we will talk about the problem of recognizing someone who is not facing the camera.

The recognition techniques described above are relatively sophisticated and work well as long as there is sufficient data to describe a person.

However, these algorithms have an inconvenient: they only work on persons facing the camera and they rely heavily on the alignment step. If someone is not facing the camera, there is nothing we can do to recognize him, as we wont be able to align his face with the ones already recorded in the database. Moreover, we also need the eye detection to be successful if we want to proceed the alignment. This step is also not guaranteed to succeed, especially if the person is wearing glasses that somehow cover his eyes.

A solution to this problem could be to use other models to represent faces, such as 3D models, in order to be able to fully describe a person, from its

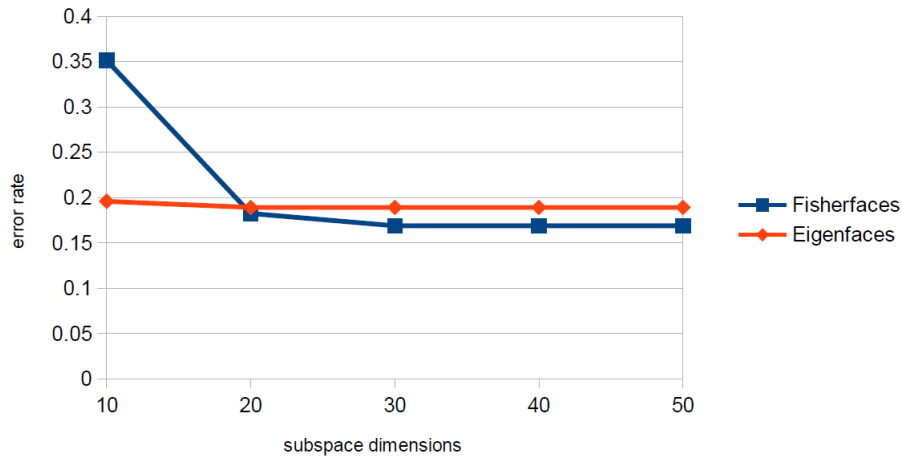


Figure 4.12: Recognition results for the Eigenfaces and the Fisherfaces methods varying the subspace dimensions. A low number of dimensions results in a higher error rate, as we are losing details. For the Fisherfaces method, a number lower than the number of classes results in an even higher error rate.

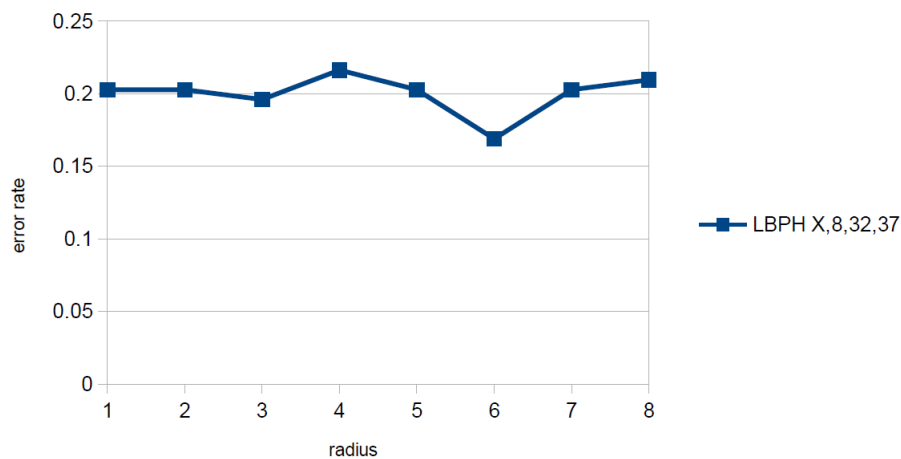


Figure 4.13: Recognition results for the LBPH method varying the radius for the Circular Local Binary Patterns. The optimal radius depends on the sample size. A small radius results in features that are too much describing local variations and a large radius result in features that don't account for local texture anymore.

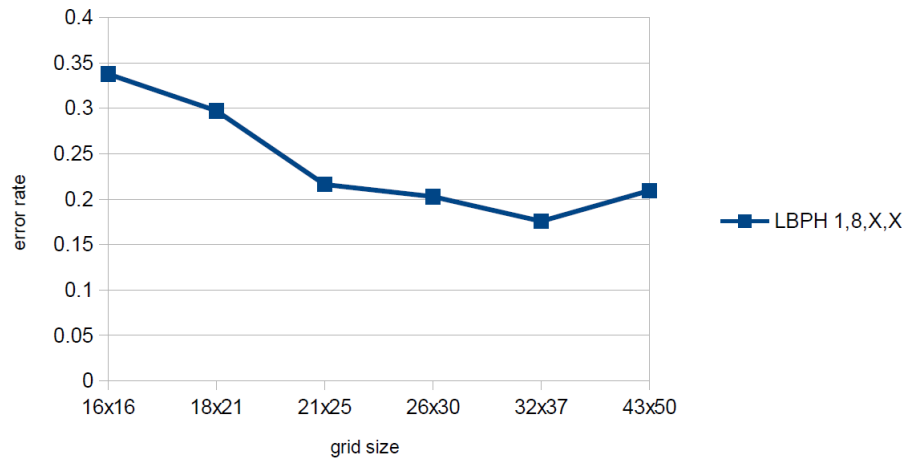


Figure 4.14: Recognition results for the LBP method varying the grid dimensions. Small grid dimensions mean large regions for computing the LBP features histogram and a loss of details. A bigger grid mean more attention being paid to local variation.

frontal face to its profile. However, these methods are more suited for videos and require way more data than we will have from users tagging static images.

If we think about this problem, even for a person, it is difficult to relate two faces of being from the same person if we have only one showing the face from front and one showing the face from profile. Another problem is that we need a process that runs in real time, a requirement difficult to achieve with such complicated algorithms.

What we propose, instead of using another machine learning algorithm for recognizing profile faces, is to use the one that works with clothes comparison described in the previous section. If we manage to recognize a specific person in one photo, because at that time this person is facing the camera, we can propagate this recognition to other pictures where the person is not facing the camera using clothes comparison.

So, as the collection of photos is loaded for tagging, we first compare the frontal faces to the ones in the database. When the recognition using the database data is done, we expand these results to profile faces using clothes. This allows to have more suggestions while not requiring too much computational resources.

4.4 Summary of the full process

In this section, we will give an overview of the full tagging process, from the time a user starts to tag, to the time it is saved and added for training. We will explain what happens when an image is part of a collection and when it is uploaded alone. We will also describe how the recognition models are trained periodically using the new tags.

The complete tool that will enable the user to tag pictures in an easy way is the following. The first and most basic assistance for tagging will be face detection, so that, at least, the user just has to click on the square and enter the face's name.

Then, using the models learned from the database of former tagged frontal faces, frontal faces in the new collection will be added their most probable identity.

Still independently from user interaction, suggestions will be propagated from frontal to profile faces.

Then, as soon as the user starts to tag new faces, modifies or accepts suggestions, the suggestions will be modified using clothes comparison for another time using user information.

In terms of implementation, this requires to keep a model of the photo collection, along with lists of tags entered by the user and tags suggested with the different methods. It also requires to solve conflicts between suggestions coming from the user information and coming from the database models. We need a way to store the suggestion probability and a way to compare a probability coming from the machine learning algorithms and a distance coming from the clothes comparison. If there are two different suggestions proposed for one face, we should keep the most probable one.

In figure 4.15, we show a workflow for the case of tagging a collection containing only one image. In figure 4.16 we detail the process for a collection of several images.

The algorithms 2, 3, 4, 5 and 6 give the pseudo code corresponding to the different actions.

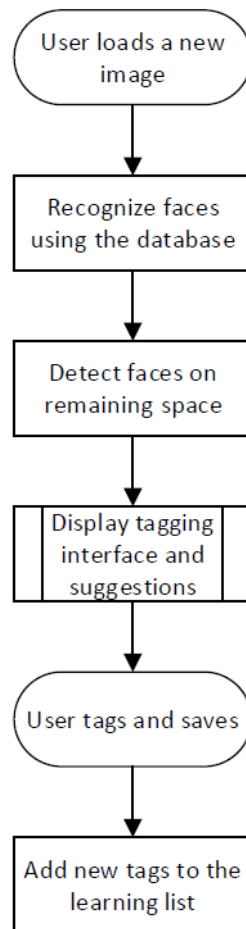


Figure 4.15: Tagging and suggestion process in the case only one image is uploaded

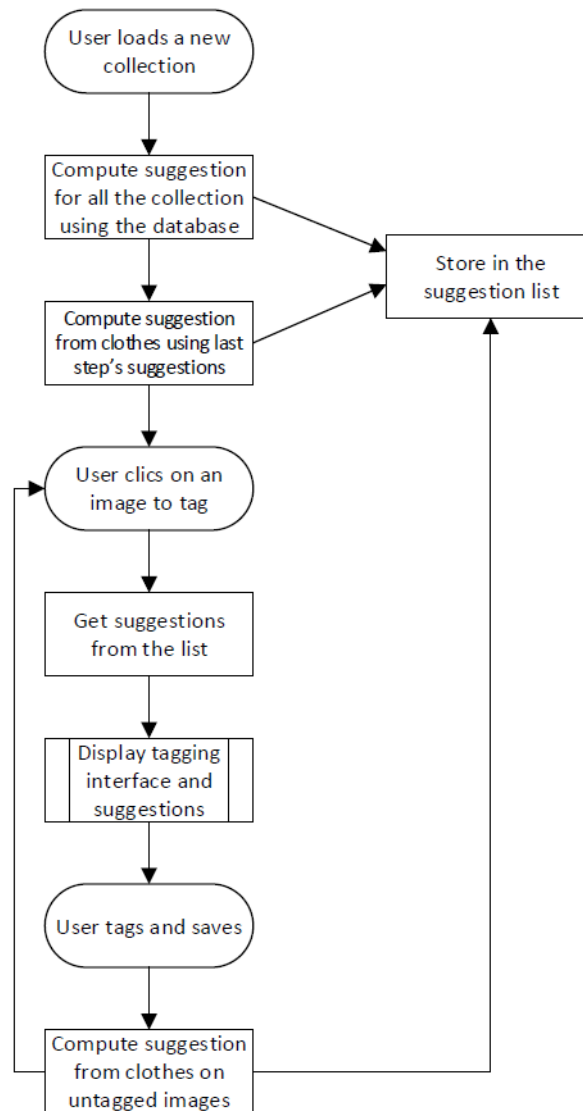


Figure 4.16: Tagging and suggestion process for a collection

Algorithm 2 OnInitialization

Input: *pathEigenModel*, *pathsBayesianModel*, *pathTrainedCascades*
 # this function is called when the server starts
 loadEigenRecognizerFromFile(*pathEigenModel*)
 loadBayesianModelFromFile(*pathsBayesianModel*)
 loadCascadeClassifiers(*pathTrainedCascades*)

Algorithm 3 OnNewImageToTag

Input: *idRecord*, *idImage*, *EigenBayesianRecognizer*, *n*,
cascadeClassifiers, *collectionHandler*
this function is called when a user starts to tag a photo
if *isSingleImage(idRecord)* **then** # the user uploaded only one photo
 imagePath \leftarrow *path(idRecord)*
 tags \leftarrow *fetchPreviousTagsFromDB(idRecord)*
 suggestions \leftarrow *EigenBayesianRecognizer.predict(imagePath, n)*
else # the user uploaded a collection of photos
 if *isNotLoaded(collectionHandler)* **then**
 imageList \leftarrow *imagesInCollection(idRecord)*
 collectionHandler.init(imageList)
 end if
 imagePath \leftarrow *path(idRecord, idImage)*
 tags, suggestions \leftarrow *collectionHandler.getTagsAndSuggestions(idImage)*
end if
detects the faces on the remaining space
faces \leftarrow *detectFaces(imagePath, tags, suggestions, cascadeClassifiers)*
renderTaggingInterface(imagePath, tags, suggestions, faces)

Algorithm 4 OnTagSave

Input: *idRecord*, *idImage*, *newTags*, *collectionHandler*
this function is called after a user validated the tags he entered on
an image
if *not isSingleImage(idRecord)* **then**
 collectionHandler.addTags(idImage, newTags)
end if
saves the new tags under 2 formats: tags objects and a json file
normalize the faces and save them in another table
saveTagsIntoDB(idRecord, idImage, newTags)

Algorithm 5 recognizerTraining

Input: *eigenRecognizer*, *bayesianRecognizer*
this function is called from time to time in background for
training including the new tags
faces, idImages \leftarrow *fetchNormalizedFacesFromDB().sortByTitle()*
PCADimensions \leftarrow *getEigenParameters()*
eigenRecognizer.train(PCADimensions, faces, idImages)
bayesianRecognizer.train(faces, idImages)

Algorithm 6 CollectionHandler

```

function INIT(imageList)
    # gets the tags for the already tagged images
    tagList  $\leftarrow$  fetchTagsFromDB(imageList)
    untagged  $\leftarrow$  untaggedImages
    recognizers  $\leftarrow$  launchRecognizers(files)
    # gets the suggestions from the trained recognizers
    suggestions  $\leftarrow$  suggestNewTags(tagList, untagged, recognizers)
    # extends the previous suggestions using clothes
    suggestion  $\leftarrow$  propagateWithClothes(suggestions, params)
end function
function ADDTAGS(idImage, newTags)
    tagList.append(newTags)
    suggestion.append(propageteWithClothes(newTags, params))
end function
  
```

Chapter 5

Conclusion

Photos take now an important place in digital libraries. In CDS, the CERN instance of Invenio, there are more than 150,000 pictures uploaded so far and this number is about to increase more and more within a few years. However, despite the fact that photos are gaining more importance in Invenio, there wasn't any way of tagging who is inside these photos.

In this project, we implemented a tagging interface that allows the user to square any part of the photo and to give it a name. But more than that, we implemented more features in order to make the user's task easier.

For the face detection, we saw that the existing detector, the cascade classifier, trained and available in OpenCV, gives bad results if the persons are not facing the camera. To solve this problem, we trained our own cascade classifiers for different angles for the head position regarding the camera. We built a training dataset from faces in the wild and we also built a robust way to evaluate the future performance of our detectors. In the result part, we discussed the role of each of the parameters and we stated what was the best configuration for our case.

In the next chapter, we addressed the problem of face recognition. We described our context and its specific properties we can take advantage of.

On one hand, we are challenged with hard conditions, as photos are taken in the wild and people appearance might change a lot from one photo to another. The illumination can be completely different and people might not be facing the camera. On the other hand, users usually upload pictures as a collection from a same event. The photos inside the collection have been taken on the same day and we can take advantage of this information.

We presented two ways of recognizing faces. The first one uses color histogram comparisons between faces and clothes of the upper body. It is meant to be used to compare photos from the same collection. The second method learns from frontal faces and several faces per person. This method has to

be trained again periodically with the new tags added by the users.

Our face recognition method is a fully working framework that uses a combination of famous algorithms for face detection and extends them for cases where they don't work using clothes comparison. It is both fast and efficient enough to be a useful tool for a user who wants to tag a photo in Invenio.

Bibliography

- [1] CERN. Centre europeen pour la recherche nucleaire. <http://www.cern.ch>.
- [2] The first website, created by tim berners-lee. <http://www.w3.org/History/19921103-hypertext/hypertext/WWW/TheProject.html>.
- [3] Alberto Pepe, Martin Vesely, Nicholas Robinson, Jean-Yves Le Meur, Maja Gracco, Thomas Baron, and Tibor Simko. Cern document server software: the integrated digital library. Technical report, 2005.
- [4] CDS. The cds invenio installation at cern. <http://cds.cern.ch>.
- [5] EPFL. Infoscience. <http://infoscience.epfl.ch>.
- [6] Opencv computer vision library. <http://docs.opencv.org/index.html>.
- [7] Paul Viola and Michael Jones. Rapid object detection using a boosted cascade of simple features. In *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, volume 1, pages I–511. IEEE, 2001.
- [8] Rainer Lienhart and Jochen Maydt. An extended set of haar-like features for rapid object detection. In *Image Processing. 2002. Proceedings. 2002 International Conference on*, volume 1, pages I–900. IEEE, 2002.
- [9] Shengcai Liao, Xiangxin Zhu, Zhen Lei, Lun Zhang, and Stan Z Li. Learning multi-scale block local binary patterns for face recognition. In *Advances in Biometrics*, pages 828–837. Springer, 2007.
- [10] Yoav Freund and Robert E Schapire. A desicion-theoretic generalization of on-line learning and an application to boosting. In *Computational learning theory*, pages 23–37. Springer, 1995.

-
- [11] Antonio Torralba, Kevin P Murphy, and William T Freeman. Sharing features: efficient boosting procedures for multiclass object detection. In *Computer Vision and Pattern Recognition, 2004. CVPR 2004. Proceedings of the 2004 IEEE Computer Society Conference on*, volume 2, pages II–762. IEEE, 2004.
 - [12] Martin Kostinger, Paul Wohlhart, Peter M Roth, and Horst Bischof. Annotated facial landmarks in the wild: A large-scale, real-world database for facial landmark localization. In *Computer Vision Workshops (ICCV Workshops), 2011 IEEE International Conference on*, pages 2144–2151. IEEE, 2011.
 - [13] Xiangxin Zhu and Deva Ramanan. Face detection, pose estimation, and landmark localization in the wild. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 2879–2886. IEEE, 2012.
 - [14] Gary R Bradski. Computer vision face tracking for use in a perceptual user interface. 1998.
 - [15] Yizong Cheng. Mean shift, mode seeking, and clustering. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 17(8):790–799, 1995.
 - [16] Josef Sivic, C Lawrence Zitnick, and Richard Szeliski. Finding people in repeated shots of the same scene. 2006.
 - [17] Ferdinando S Samaria and Andy C Harter. Parameterisation of a stochastic model for human face identification. In *Applications of Computer Vision, 1994., Proceedings of the Second IEEE Workshop on*, pages 138–142. IEEE, 1994.
 - [18] Lawrence Sirovich and Michael Kirby. Low-dimensional procedure for the characterization of human faces. *JOSA A*, 4(3):519–524, 1987.
 - [19] Ronald A Fisher. The use of multiple measurements in taxonomic problems. *Annals of eugenics*, 7(2):179–188, 1936.
 - [20] Peter N. Belhumeur, João P Hespanha, and David J. Kriegman. Eigenfaces vs. fisherfaces: Recognition using class specific linear projection. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 19(7):711–720, 1997.

- [21] Baback Moghaddam, Wasiuddin Wahid, and Alex Pentland. Beyond eigenfaces: Probabilistic matching for face recognition. In *Automatic Face and Gesture Recognition, 1998. Proceedings. Third IEEE International Conference on*, pages 30–35. IEEE, 1998.
- [22] Marcio Luis Teixeira. *The bayesian intrapersonal/extrapersonal classifier*. PhD thesis, Colorado State University, 2003.
- [23] Ferdinando S Samaria and Andy C Harter. Parameterisation of a stochastic model for human face identification. In *Applications of Computer Vision, 1994., Proceedings of the Second IEEE Workshop on*, pages 138–142. IEEE, 1994.
- [24] The csu face identification evaluation system. <http://www.cs.colostate.edu/evalfacerec/index10.php>.
- [25] P Jonathon Phillips, Harry Wechsler, Jeffery Huang, and Patrick J Rauss. The feret database and evaluation procedure for face-recognition algorithms. *Image and vision computing*, 16(5):295–306, 1998.