FACIAL IMAGE-BASED GENDER AND AGE ESTIMATION

August 2013

Authors:
Bernat Bas <bernatbaspujols@gmail.com>
Francesc Riera <xescka@gmail.com>

Supervisors:
Jörgen Ahlberg <jorgen.ahlberg@isy.liu.se>
Nenad Markuš <nenad.markus@fer.hr>
Philippe Salembier (UPC) <philippe.salemier@upc.edu>

Project Examiner:
Robert Forchheimer <robert.forchheimer@liu.se>
Introducció i repartició de tasques (UPC)

Aquest Treball Final de Grau (TFG) ha sigut desenvolupat conjuntament per Francesc Riera i Bernat Bas a la Universitat de Linköping (LiU, Suècia), sota la supervisió de Jörgen Ahlberg (LiU), Robert Forchheimer (LiU) i Philippe Salembier (UPC), durant el segon semestre del 2013. El treball ha estat realitzat per a ser supervisat, presentat i avaluat segons els criteris establerts per la Universitat de Linköping; és per això que la memòria del projecte no compleix amb alguns criteris establerts per la ETSETB de la UPC (per exemple l’extensió o elaboració de memòries individuals). Tot i així, creiem que per tal de fer entendre el treball realitzat és important respectar l’estructura i extensió de la memòria.

El títol oficial del treball és “Facial image-based gender and age estimation”. Sense afany d’entrar en matèria, es pot deduir ràpidament a partir del títol en què consisteix el nostre treball: fer una estimació sobre l’edat i el sexe de individus tot utilitzant imatges facials.

A priori pot semblar que l’estimació de l’edat i del sexe són processos distints i diferenciables, tot i així ambdós mètodes requereixen d’unes eines i estructures força similars. Aquest és el motiu pel qual vam fer el treball en parella ja que d’aquesta manera hem pogut aprofundir més en tasques d’investigació que en feines de desenvolupament d’eines (per exemple funcions). És impossible diferenciar clarament quina part ha desenvolupat cadascú. Els dos ens hem implicat en tasques tals com investigació, documentació, programació, avaluació de mètodes, decisions o redacció. Cada pas que hem fet en aquest projecte l’hem discutit conjuntament, així mateix, els dos hem desenvolupat el mateix tipus de tasques per tal de tenir una vista i intervenció global dins del projecte. Òbviament no hem realitzat exactament les mateixes feines, sinó que dia a dia ho hem repartit i coordinat de tal manera que els dos toquéssim tots els àmbits. Aquest model de “els dos fem de tot” creiem que ha sigut molt productiu a nivell de coneixement del projecte, ja que ens ha lliurat als dos una visió molt àmplia del treball.
Abstract

The principal objective of this project is to develop methods for the estimation of the gender and the age of a person based on a facial image, using classification for the gender estimation and regression for the age. The extracted information can be useful in, for example, security or commercial applications. This is a difficult estimation problem, since the only information we have is the image, that is, the looks of the person. Using image features such as gradients, pixel differences, and Histograms of Oriented Gradients (HOGs), and high-level features like hair, moustache or beard, a classifier/regressor is trained. The training process needs to be optimized in terms of pre-processing, feature selection, choice of classifier/regressor, and classification/regression parameters. Our experiments show that HOG is the most useful feature in order to estimate both age and gender. The Support Vector Machine is the best classifier and Random Forest is the best regressor. For gender estimation the misclassification rate is about 2%, and the performance of the age estimation is close to what humans achieve.
Contents

1 Introduction

2 Background theory
  2.1 Image classification
  2.2 Preprocessing
  2.3 Commonly used image features
    2.3.1 Pixel difference
    2.3.2 Gradient
    2.3.3 Hessian
    2.3.4 Histograms of Oriented Gradients
    2.3.5 Scalar-invariant descriptors
    2.3.6 Facial features
  2.4 Estimators
    2.4.1 Classification and regression
    2.4.2 Tree
    2.4.3 Random forest
    2.4.4 Random Ferns
    2.4.5 Support Vector Machines
  2.5 Evaluation methods
  2.5.1 K-fold cross validation

3 Dataset
  3.1 Introduction
  3.2 Feret
  3.3 FDP
  3.4 G200
  3.5 Gallagher
    3.5.1 Gallagher subset
  3.6 Morph
    3.6.1 Morph subset
  3.7 Gender database
  3.8 Age database
  3.9 Experiments
Chapter 1

Introduction

Some of the most basic classifications regarding humans are gender and age. They are among the very first things a person decides on sight of someone. These decisions are based on many different features strictly coming from the person, but also from the environment.

The marketing or sales departments of companies are usually interested in their products’ targeted customers. Thus, it is important in many fields to have statistics of the target audience of their products. In the same way, some services, permissions or products are only allowed to an audience of certain gender or age, and it has to be somehow controlled.

Bringing together the two ideas mentioned above, the need of estimating the gender and age in some automatic way appears. A human can easily make these estimates from faces. Yet, it is still a challenging task for a computer. This project is focused on gender and age estimation based on face images using computer vision techniques. The aim of the project is not to create an estimation system, but to develop the estimation methods and finally to create a system prototype using those methods.

Although the system prototype is expected to perform almost as well as humans would do, it has some limitations. As we said, there is a lot of information that humans use to estimate the age or the gender, consciously or unconsciously, such as the voice, the clothing, the behaviour...that the system does not use. Thus is difficult to achieve a performance as good as human. Even just with facial images, humans would use some high-level information, like the skin tone, the hairstyle, the facial hair...that can be hard for the prototype system to extract. On the other hand the system will use computer vision techniques that humans are not able to use. Since the project has multiple applications, and every application has its own specific characteristics, we cannot focus on either the final error or the estimation time that the system takes. Thus, we will try to look for a trade-off between the error and the computational complexity, taking the features that give a better performance, but being aware of the complexity, and skipping those that need too much information for a very small improvement.

For training the prototype system, a database of facial images labelled with the age and gender is needed. This database has to be representative for the real input images of the system. Since we do not have to make any application but just develop the system, we do not need to look for a very specific database.

The steps of any estimation process are: Preprocessing, feature extraction, estimation, and evaluation. The preprocessing step is defined in order to prepare the data for the feature extraction step. These methods work in pixel level (gradients), in object level (keypoint
tracking) or image level (crop and rotation). The images are also converted to grayscale and resized to a fixed size so all of them have the same amount of features.

For developing the prototype system, two types of features are needed: high-level and low-level features. High-level features are those that have a direct connection with the age/gender, such as presence of hair, moustache, beard or wrinkles. On the other hand, the low-level features do not have a clearly visual correlation with the prediction and they are extracted using the most common computer vision techniques, such as the gradient, hessian, pixel difference or Histogram of Oriented Gradients (HOGs). For extracting them we need some knowledge about where some facial keypoints are located, hence some sort of tracking algorithm is needed.

There is an important difference between age and gender: for gender, there are two clearly distinguished classes (male and female), while for age estimation, the values to predict are continuous, thus the estimation method needs to be different. For this reason, the gender is estimated with a classification method and the age with a regression method. The investigated estimation methods are Trees, Random Forests, Ferns and Support Vector Machines. This estimation difference has two principal consequences. The first one is the estimator, which has to be adapted from classification to regression. Support Vector Machines cannot be adapted to regression, thus they will only be used in classification. The second consequence is the estimator evaluation. In gender classification, the percentage of misclassification is used in order to compare estimators. In age regression, the error relative to the true age is used. Then, in order to compare between different regressors, the area under the first few samples of the relative error distribution curve is used.

The system was developed in Matlab, because it allows implementing all the needed functions easily, and some already implemented functions can easily be found in both, on the Internet or on the Matlab toolboxes.

The methods developed in this project can be applied in many fields for different purposes. It can be used in security issues controlling that only people over a specific age buy certain products or use some services, or only people of a specific gender go in some places. Road or airport controls, where a concrete person is wanted, can also be done with this system, as far as his age and gender are known. It has also marketing applications, finding out the statistics of which kind or people buy some product and defining its target. Thus, companies can make specific advertising campaigns focused on this target. The closest the system gets to the real age, the better, but all these applications allow a certain error margin, so if instead of a specific value, a range of ages is returned, the system is still useful.
Chapter 2

Background theory

2.1 Image classification

The first thing to do while facing a big problem is to analyse it and divide it into different specific parts. A generic problem can be split into many steps with specific functions, so a tedious work is divided into small and well-defined parts.

In this case, the division has been made in the most convenient way to analyse each step and to separate depending on the specific task. The main steps are shown in Figure 2.1.

![Figure 2.1: Scheme for system development.](image)

These steps need to be well defined, and their parameters depend on the application. Some sort of preprocessing is usually needed to ensure that the system will be able to extract all the needed features. Once all the features have been extracted, the estimation can be implemented. Finally, some kind of test needs to be done in order to evaluate the system’s performance.

2.2 Preprocessing

It is frequent the case that, in order to improve the performance of the system, a preprocessing step for the input samples is needed. The new data captured (input to the system) may change depending on the environment, and these changes can be interpreted as noise for the system. It may also happen that the input sample has more information than the useful, and all this extra information needs to be removed. Those are the reasons why the preprocessing step is needed.

The preprocessing step takes an input sample, processes it and returns it as an output, which will be used as an input to another function. The amount and kind of preprocessing done depends on that function.

In the particular case of image processing, the preprocessing step may include detecting a specific object in the whole image, adapting the image by rotating, cropping, equalizing... so
the feature extraction becomes easier.

2.3 Commonly used image features

A fundamental step in facial expression analysis is extracting effective features from original face images. Based on feature point (e.g., mouth corners and eye pupils) detection, geometrical features can be exploited. However, accurate and reliable detection and tracking of feature points are difficult problems in real-world scenarios. Another kind of features describes the appearance (skin texture) of the face. These features are less sensitive to errors in feature point detection and can encode changes in skin texture that are important for facial expression modelling.

Feature extraction can be done by having either a large amount of easily computable features or just a small group of very specific ones. In many practical applications, speed or computational efficiency is a key concern, and thus it is highly desired that the features used can be computed easily and efficiently. Therefore, a combination of low-level and high-level features is extracted.

We consider low level features all those features that work at pixel level or surroundings, and do not use object recognition or detection, e.g., Gradients or HOGs. A large number of features can be extracted in a short time.

In contrast, high level features are those that are based on feature point or area detection, which try to quantify the existence or magnitude of some specific object, shape or colour, e.g., wrinkles or beard detection. They usually take some time to be extracted, and do not give a large amount of features, but very specific.

2.3.1 Pixel difference

The pixel difference feature is a very simple and very common feature. It consists of using the difference between two pixels’ intensities as a feature. The intensity values of two pixels selected from the whole image have to be extracted, and then compute the difference between them. If it was done this way, the result would be a single number for each comparison. However a threshold is usually set up so a binary result can be obtained. In case the threshold is 0, the result is just the comparison of two pixels, telling which of them has the largest value:

\[ \text{pixel}_i - \text{pixel}_j > \text{threshold}. \] (2.1)

The particular case when \( \text{threshold} = 0 \):

\[ \text{pixel}_i > \text{pixel}_j. \] (2.2)

The binary result of each comparison, which is represented numerically as 1 or 0, is used as the feature. Even for a small image, there is a large amount of pixel-comparison features. The amount is usually too large to be handled easily. In order to reduce this large amount of features, the images are usually converted into the grey colour space and down-sampled, but anyway, the number is still too big. For example, for a 24 x 24 pixels image, 165,600 features will be obtained. We assume that the comparison of pixel \( i \) and \( j \) is the same as \( j \) and \( i \), so the half of the comparisons can be skipped.

Given this large amount of features, the goal is to minimize the number of outputs of the function whereas still achieving high discrimination accuracy.
One of the first solutions for reducing this amount is not to obtain all the possible differences but only compute, for each pixel, the difference between the pixels from the same column and row. With this technique the number of the features is remarkably reduced and there is still a good representation of all the image features. For the same image from the example above, in this case we will obtain 13,248 features. Although the amount has been reduced by one magnitude, it is still large.

It has also been proved that results with high accuracy on face orientation discrimination and gender classification can be obtained by comparing the intensities of just a few pixels [1]. Thus we can just choose some random pixels and compare them, and still have a good performance. The amount of pixels depends on the application and on the image size, and needs to be optimized by observing the given results. Typically a few thousands may be good enough.

2.3.2 Gradient

The gradient is another low-level feature that may be useful. Like pixel difference feature, it is hard to understand how the classification is done from the gradient, because it is not something that has a strong association in the real world. Even though figures close to the original image appear in the gradient images, could be understood by humans, for the computer is just an amount of pixel values. Thus, it is hard to understand which is the exact way to distinguish between genders or ages using Gradients. However, this feature is very fast to compute, and even though with this, it may be helpful to classify [2], [3].

The gradient of a scalar field (i.e., a grey scale image) is a vector field that points in the direction of the greatest rate of increase of the scalar field, and whose magnitude is that rate of increase. In the image case, the gradient depends on the variation of the pixel intensities and it is obtained by convolving the image with a filter. This technique is typically used for edge detection. Several filters can be used to compute it: 
\[ [1, -1], [1, 0, -1], [1, -8, 0, 8, -1], Sobel \ldots \]

The gradient can be computed for any given direction, but taking two perpendicular directions is enough to get the full range of directions (typically horizontal and vertical directions are the chosen ones). For each direction, the variation of the whole image in that direction is calculated, so a gradient image as large as the image is, representing the variation in this way is obtained. Therefore, two gradient images (in x and y directions) are obtained, one with the horizontal variation and the other with the vertical variation. An example of the result can be seen in Figure 2.2.

![Original image, horizontal gradient (dx) and vertical gradient (dy) respectively.](image-url)

Figure 2.2: Original image, horizontal gradient (\( dx \)) and vertical gradient (\( dy \)) respectively.
2.3.3 Hessian

The hessian is a square matrix with the second-order derivatives. It works using the same technique as the gradient feature, but it computes the gradient from the gradient image instead of from the original image. Three matrices for each input image are obtained in this case ($dxy$ is the same as $dyx$).

2.3.4 Histograms of Oriented Gradients

Histograms of Oriented Gradient descriptors (HOG) are feature descriptors used in computer vision and image processing for the purpose of object detection. Dalal and Triggs introduced this feature for human detection [4], [5], [6], [7], [8], [9]. The essential thought behind the HOG descriptors is that object appearance and shape within an image can often be characterized rather well by the distribution of intensity gradients or edge directions. The implementation of these descriptors is achieved by dividing the image into small regions (windows), and, for each window, create a histogram in order to describe the distribution of the directions. Those histograms are contrast-normalized in order to obtain the probability distribution.

Algorithm implementation

1. Gradient computation
   The first step is to compute the gradient, which is usually done by filtering the image as explained in the 2.3.2 Gradient section.

2. Angle and magnitude computation
   The orientation (angle) is needed to make the histogram and the magnitude to weight the histogram. The gradient magnitude $m(x, y)$ and orientation $\theta(x, y)$ are calculated using the $x$ and $y$ directional gradients $\partial x(x, y)$ and $\partial y(x, y)$. The result can be seen in Figure 2.3.

   \[ m(x, y) = \sqrt{\partial x(x, y)^2 + \partial y(x, y)^2} \quad (2.3) \]

   \[ \theta(x, y) = \arctan \left( \frac{\partial y(x, y)}{\partial x(x, y)} \right) \quad (2.4) \]

Figure 2.3: Orientation (Eq. (2.4)) and gradient magnitude (Eq. (2.3)) of an image.
3. Divide into windows
The next step is divide the angle and magnitude images in windows, as can be seen in Figure 2.4. The number of windows for row and column as well as the overlap between the neighbouring ones are parameters to select. That overlapping ensures consistency across the whole image without the loss of local variations. Two main window geometries exist: rectangular R-HOG and circular C-HOG.

![Figure 2.4: Orientations image divided in windows. Then contents of one single window for both orientation and magnitude images.](image)

4. Orientation binning and normalization
For each block a histogram of the orientations is computed, as can be seen in Figure 2.5. The angles go from $-\pi$ to $+\pi$. The number of bins determines the precision of the histogram, thus using $B$ bins the limits of intervals of the histogram go from $[-\pi + \frac{2\pi}{B} (i-1)]$ to $[-\pi + \frac{2\pi}{B} i]$ with $i = 1, \ldots, B$. The gradient magnitude is also important and it is used to weight the histogram. Consequently, the value of the magnitude is added to the corresponding bin in the histogram instead of add only one every time that an angle is between the limits of a bin. In pseudo code:

```pseudo
def add_to_histogram(image)
    for each block in image
        histogram = np.zeros(B, dtype=np.float32)
        for each pixel in block
            angle = angle_of_pixel
            if angle is within the limits of i
                histogram[i] += magnitude
        return normalized_histogram
```

Finally, normalization is applied in the histogram computed in each window using the formula

$$Hist_{norm} = \frac{Hist}{\|Hist\| + \epsilon}$$

(2.5)

where $Hist$ is the histogram without normalization, $\|Hist\|$ the Euclidean norm of the vector and $\epsilon$ a small constant (whose value do not influence the results, is just to avoid have a zero in the denominator). The normalization ensures that low contrast regions are stretched.
5. Concatenation

The last step consists in concatenating the normalized histograms into a 1-D vector, which will be as large as the number of features.

2.3.5 Scalar-invariant descriptors

Some of the most used features in image processing are the ones that can give a description of a local neighbourhood for the interesting points on an object in an image. Either of SIFT, SURF or DAISY are algorithms used to detect and describe local features in images. These are particularly useful to match the same object appearing in different images.

The SIFT (Scale-Invariant Feature Transform) algorithm was developed in 1999, and it is based on the appearance of the object at particular interest points. The first step of the algorithm is to detect which are the interest points (keypoints). To ensure the scale-invariability the image is convolved with Gaussian filters at different scales, and then the difference of successive Gaussian-blurred images are taken. The key step in achieving invariance to rotation is using an algorithm similar to HOG. [9],[10]

Using the same idea as SIFT, the SURF (Speed Up Robust Features) and DAISY (uses a structure similar to a daisy flower to analyse the points to describe) algorithms appeared few years later. Both of them use different techniques for computing a description for the keypoints much faster than SIFT. Both, SURF and DAISY, have similar performance to SIFT.

Besides these three, there are MSER (Maximally Stable External Regions) and MSCER (Maximally Stable Colour External Regions) algorithms, which have the same purpose but they are not as used as the SIFT, SURF or DAISY.

All of the above techniques are used for finding keypoints, provide a description, and afterwards, corresponding the same points in different images by finding the ones with the closest description.
2.4. ESTIMATORS

2.3.6 Facial features

When working with image processing, the list of features to extract is pretty well known. The features explained in the former sections are the most basic features for any image processing project, but there are many other possible features that are very specific for the application of the project. Thus, when working with face images, some particularities of this kind of images can be exploited. In the case of facial features the facial area, position of the eyes, skin colour, mouth size, presence of glasses or earrings could be some examples.

The detection of some object presence in the image can be difficult to do, but at the same time it can provide very useful information for classifying. Analysing one by one the training samples, some particularities may be found. The goal is to, besides the common features in image processing, extract very specific ones, that can only be useful in a specific project and that can change depending on the application.

In most of the cases extracting these features is a hard job, it takes time and is computationally complex (sometimes they require a very heavy preprocessing step) and is not always possible to extract it properly. There is a trade-off between how well do you want to extract each feature and how complex it is. Therefore, in some cases they could help a lot in the estimation, but in some others they would be useless.

2.4 Estimators

When working with image processing, the list of features to extract is pretty well known. The features explained in the former sections are the most basic features for any image processing project, but there are many other possible features that are very specific for the application of the project. Thus, when working with face images, some particularities of this kind of images can be exploited. In the case of facial features the facial area, position of the eyes, skin colour, mouth size, presence of glasses or earrings could be some examples.

The detection of some object presence in the image can be difficult to do, but at the same time it can provide very useful information for classifying. Analysing one by one the training samples, some particularities may be found. The goal is to, besides the common features in image processing, extract very specific ones, that can only be useful in a specific project and that can change depending on the application. In most of the cases extracting these features is a hard job, it takes time and is computationally complex (sometimes they require a very heavy preprocessing step) and is not always possible to extract it properly. There is a trade-off between how well do you want to extract each feature and how complex it is. Therefore, in some cases they could help a lot in the estimation, but in some others they would be useless.

The unknown nature of the observation is called class. It is denoted by \( Y \) and, in the simplest case, takes values in the binary set \(-1, 1\) or \(0, 1\). However the unknown nature can take any real number as a value. In some applications those values are integers (e.g. in size estimation) and in others are strings (e.g. race estimation). The goal of the system is, using training data, trying to estimate the probabilistic function, and predict the corresponding output \( Y \) of a new input \( X \). The estimation problem has some typical examples:

- In computer vision
  - \( X \) is an image.
  - \( Y \), \(1, 0\) depends on if the image contains or not an object.
The practice of pattern estimation has been improved a lot in the last few years. The introduction of new and effective techniques of handling high-dimensional problems—such as boosting and support vector machines—have revolutionized the practice of estimation. Some of the most common methods are explained in the following sections, as well as the difference between classification and regression.

2.4.1 Classification and regression

The estimation problem can be a classification or regression problem, and it is important to know the difference, and when we have to use each of them. Classification estimators, as the name implies, are used to separate the dataset into classes belonging to the response variable. Usually the response variable has two classes, but it can have more. Thus, classification estimators are used when the unknown nature (Y) is discrete. [11]

On the other hand, regression estimators are needed when the unknown variable is numeric and continuous. For example, the predicted price of a consumer good. Hence, regression estimators are applicable for prediction type of problems as opposed to classification [12].

One big difference between classification and regression is the way to compute the misclassification value. In classification the number of errors is usually counted, and, since the output is binary, it is easy to know which percentage of misclassification is obtained. In regression this method cannot be used, because knowing how wrong is the output is needed, and not only a binary decision whether it is wrong or not. Therefore, the absolute value or the square of the difference between the obtained output and the ground-truth value are some ways to compute the regression error.

2.4.2 Tree

The classification and regression trees (CART) are estimation methods that emulate the tree structure as it can be seen in Figure 2.6, generating nodes and branching until the leaves [13], [14], [15], [16]. Each internal node represents test on an attribute, each branch represents the outcome of the test and each leaf (pink nodes in the tree below) represents class label (decision taken after computing all features). All the features can be used one or more times with different thresholds.

In each node a feature is evaluated setting up a threshold. The outcome of the comparison is usually binary (if it has more that 2 splits, it can be divided in many binary decisions). Then one of the two branches is chosen until the next node. Once the tree is created, the classification is an easy step, because just binary comparisons have to be computed until reaching to a leaf node. The leaf node will give the value of the sample.
2.4. ESTIMATORS

CHAPTER 2. BACKGROUND THEORY

Figure 2.6: Example of Tree classifier structure.

A tree can be generated by splitting the source set into subsets based on an attribute value test. This process is repeated on each derived subset in a recursive manner. We seek a property to be tested at each node that makes the immediate descendent node as pure as possible. The impurity can be measured in three different criteria:

- Entropy impurity (the most acceptable measure in most cases):
  \[ i_E(N) = -\sum_j P(\omega_j) \log_2 P(\omega_j) \]  

- Gini impurity:
  \[ i_G(N) = \sum_j P(\omega_j)(1 - P(\omega_j)) = 1 - \sum_j P(\omega_j)^2 \]  

- Misclassification impurity:
  \[ i_M(N) = 1 - \max_j P(\omega_j) \]  

\(P(\omega_j) = \frac{n_j}{n_{TOTAL}}\) is the fraction of patterns at a given node belonging to category \(\omega_j\), and \(N\) is the node where the impurity is calculated.

The recursion is completed when the subset at a node has all the same value of the target variable, or when splitting no longer adds value to the predictions. Since the goal of this technique is to create a simple estimator, the size of the tree is a very important factor and some stopping criteria are defined. The most used one is to minimize the function

\[ \alpha \cdot \text{size} + \sum_{N_{leaf}} i(N) \]  

where \(i(N)\) is the impurity calculated previously and \(\text{size}\) is the number of nodes. Large values of \(\alpha\) result in smaller tree size. Even though the application of these criterions, if the tree is still too large then a pruning can be applied.
Pruning is a method to reduce the tree size by merging different leaves into one. When doing so, the misclassification error increases but the tree size decreases, so a faster but less trustable tree is obtained. It preferred create a tree and then prune it over using a heavy stopping criteria, because the obtained results are better, even though there is a waste of time and computational capacity, because an extra effort has to be done.

In a standard classification tree, the idea is to split the dataset based on homogeneity of data. Imagine that we have two variables: age and weight that predict if a person is going to sign up for a gym membership or not. In our training data if it showed that 90% of the people who are older than 40 signed up, we split the data here and age becomes a top node in the tree. We can almost say that this split has made the data ‘90% pure’.

In a regression tree this is the idea: Since the target variable does not have classes, we fit a regression model to the target variable using each of the independent variables. Then for each independent variable, the data is split at several nodes. Since using the impurity equations would not be useful because the there is not a finite number of classes, some sort of measurement needs to be implemented. At each node, the error between the predicted value and the actual value is squared to get a sum of Squared Errors (SSE). The node errors across the variables are compared and the variable yielding the lowest SSE is chosen as the root node. This process is recursively continued.

2.4.3 Random forest

Using one single tree with high training set size it is likely to produce overfitted trees. Thus, one solution is a random forest. A random forest operates by joining a multitude of decision trees at training time and outputting the class that is the mode (the value that appears most often) of the classes’ output by individual trees, with each tree grown using some form of randomization. [17],[18],[19],[20].

To ensure the randomness of the trees, and that they will be representative enough, just a randomly picked subset of the samples is used to create each tree so it changes for everyone. Thus, every tree will be different, and all of them would represent properly the full set. This technique is called Bagging.

For doing the estimation once the forest is created, the input sample has to pass through all the trees and get an output from all of them. All the trees vote and the most given output value is the value for that sample.

It is also possible to have regression random forest. For estimating the final decision in regression the mean of the outcomes of all the trees is computed.

2.4.4 Random Ferns

Ferns are a still more simplified group of trees, in which some kind of uniformity is desired. Ferns are non-hierarchical structures where each one consists of a small set of randomly picked binary tests and returns the probability that a patch belongs to any one of the classes that have been learned during training. These responses are then combined in a Naive Bayesian way. As before, we train our classifier with a statistically good representation of the possible input samples. The Ferns much faster and simpler to implement than the randomized trees even though the misclassification rate may be higher.

As in random forests, leaves store the posterior probabilities. During testing the probability of a sample belonging to any one of the classes that have been learned during training
is returned. The result of each test and the ordering on the set defines a binary code for accessing the leaf node. In the same way as random forests, the test sample is passed down all the randomized ferns. Each node in the fern provides a result for the binary test, which is used to access the leaf that contains the posterior probability. The posteriors are combined over the ferns in the same way as for random forests over trees [17],[4],[21],[20].

A tree can be transformed into a Fern by performing the following steps (see the Figure 2.7). First, we constrain the tree to systematically perform the same test across any given hierarchy level, which results in the same feature being evaluated independently of the path taken to get to a particular node. Second, we do away with the hierarchical structure and simply store the feature values at each level. This means applying a sequence of tests to the patch, which is what Ferns do.

![Figure 2.7: Scheme for transforming a Tree into a Fern.](image)

### 2.4.5 Support Vector Machines

Support Vector Machines (SVM) are a set of related supervised learning methods used for classification and regression. They belong to a family of generalized linear classifiers. SVM can be mathematically described as a function that receives data (observations, features) as input and generates a function that can be used to predict some features of future data. Basically the SVM search the optimal hyperplane for linearly separable patterns [14],[17]. However, this idea can be extended to non-linearly separable patterns using transformation of original data to a new space [22],[23]. This transformation is called kernel function.

![Figure 2.8: Different ways to separate the data.](image)

As said before, SVM search the best hyperplane to separate the data (there are several ways to separate the data as can be seen in Figure 2.8). Using other classification methods (for example neural networks) it is also possible to find the hyperplane to do so. There are many hyperplanes that can separate the data. However, only one of these achieves maximum separation.

To show the theory of support vector machine we start with a simple problem: 2-D linearly separable data, where a simple line is the separator. In higher dimensions, hyperplanes are needed, but the problem formulation is almost the same. In this 2D example the separator can be expressed as:

\[ g(x, y) = ax + by = c \]
Thus the separation criteria is:

\[ ax + by > c \quad \text{CLASS 1} \]
\[ ax + by < c \quad \text{CLASS 2} \]

As it is said before, the optimal hyperplane is found, i.e., the one with the maximum distance from the nearest training patterns. The support vectors are those patterns which are nearest to the separator, and which determines the distance to the separator. Continuing to the 2D example, the distance \( C \) from a point \((x_0, y_0)\) to a classifier line is

\[ \frac{|ax_0 + by_0 + c|}{\sqrt{a^2 + b^2}} \]

In the support vectors, the distance \( C \) can also be expressed using vector notation, with the expression

\[ C = \frac{w^T \cdot x + w_0}{\|w\|} = \frac{1}{\|w\|} \]

where \( w \) is the weight vector \((a \text{ and } b \text{ in our example}), w_0 \) is the offset \((c \text{ in the example})\) and \( x \) is the feature vector \((x_0, y_0)\).

In order to maximize the \( C \), we need to minimize \( \|w\| \), but subject to that the distance of the vectors has to be bigger than \( C \). Thus, the equation

\[
\max_{w,w_0} C \quad \text{subject to} \quad \frac{1}{\|w\|} y_i(w^T x_i + w_0) \geq C \quad i = 1, \cdots, N.
\]

is obtained, where \( y_i \) is the output, \( x_i \) is the input and \( N \) is the number of data, and it can be written as this expression

\[
\min_{w,w_0} \frac{1}{2} \|w\|^2 \quad \text{subject to} \quad y_i(w^T x_i + w_0) \geq 1 \quad i = 1, \cdots, N.
\]

This is a constrained optimization problem that is solved by Lagrangian method. The function to optimize is

\[ L = \frac{1}{2} \|w\|^2 - \sum_{i=1}^{N} \alpha_i \cdot (y_i(w^T x_i + w_0) - 1). \]

It is a convex problem, where the solution has to accomplish the conditions of Karush-Kuhn-Tucker (sufficient and necessity):

\[ \nabla_w L = 0 \quad \rightarrow \quad w = \sum_{i=1}^{N} \alpha_i y_i x_i \]
\[ \nabla_{w_0} L = 0 \quad \rightarrow \quad 0 = \sum_{i=1}^{N} \alpha_i y_i \]

\[ \alpha_i \geq 0 \quad \forall i \]
\[ \alpha_i (y_i(w^T x_i + w_0) - 1) \quad \forall i \]
Hence, the Lagrangian function is

\[ L = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{i=1}^{N} \alpha_i \alpha_k y_i y_k x_i^T x_k \text{ subject to } \alpha_i \geq 0 \text{ and } \sum_{i=1}^{N} \alpha_i y_i = 0. \]

Solving the equation, the \( w \) and \( w_0 \) values are obtained by the expressions

\[ w = \sum_{i=1}^{N} \alpha_i y_i x_i \]
\[ w_0 = -w^T x_i + y_i \]

Then, the classification is done with the expression that we have seen before:

\[ c = \text{sign}(w^T x_i + y_i) \]

**Soft-margin classifiers**

In real world problem it is not likely to get an exactly separate line dividing the data (see Figure 2.9). We have a hyperplane, which might exactly separate the data, but this may not be desirable if the data has noise. Sometimes, is better to ignore a few data points, and assuming a probability of some vector misclassified. Now we have:

\[ y_i (w^T x_i + w_0) \geq 1 - \xi_i \quad i = 1, \ldots, N. \]

Then the Lagrangian is:

\[ L = \frac{1}{2} \|w\|^2 + P \sum_{i=1}^{N} \xi_i - \sum_{i=1}^{N} \alpha_i \cdot (y_i (w^T x_i + w_0) - 1 + \xi_i) - \sum_{i=1}^{N} \beta_i \xi_i \]

Figure 2.9: Case in which the data cannot be linearly separate.
2.5 Evaluation methods

It is obvious that having a good estimator important, but it is also important to have a good method to analyse how good each estimator is so the comparison between them is possible. It has no sense to try to find the very best estimator if it cannot be evaluated properly. In order to find a good characterization of the system and taking into account the amount of available samples for testing the system, some methods have been proposed.

Learning the parameters of a prediction function and testing it on the same data is a methodological mistake. A model that would just repeat the labels of the samples that it has just seen would have a perfect score but would fail to predict anything else. This is called overfitting. Therefore, usually the available samples are randomly split in two subsets: train
2.5. EVALUATION METHODS

and test samples. The train subset is used to learn the parameters for the estimator, and then those parameters are used in the test subset. Thus, the training and testing samples are not the same. There is, however, a potential flaw in this method. When the dataset is randomly split into two subsets, it may cause a non-representative enough dataset, and we run some risk of creating a test set that is missing some characteristics that are present in the full data set. The intuitive way to reduce this risk is to repeat the train/test split several times, each time calculating the error on the test set. However, even with several splits, random partitioning does not guarantee that every point will appear in at least one test set. [24],[25]

Once you can ensure that the results are statistically significant, another parameter to discuss is what we want to analyse for each sample. Thus, a method in which the evaluation will be done has to be defined. One of the more useful methods is the number of misclassifications in classification, or the error percentage in regression, but other parameters, like the time to compute, the computational complexity or so on, are also useful in some cases.

Finally, once all the results are obtained, it is also important to find a way in which they can be analysed properly. In some cases is important to have a statistical representation of the error for every kind of samples, so it can be known in which group of samples the error is higher, on which percentage of misclassification is obtained for each group.

2.5.1 K-fold cross validation

There are some different methods for applying the cross validation in a set of samples, but one of the most used is k-fold. K-fold cross validation provides a framework for creating several train/test splits and guaranteeing that each data point is in the test set at least once [26]. The procedure is simple:

1. Split the data into k equal-sized groups
2. For i=1 to k,
   (a) select group i to be the test set and all other (k-1) groups to be the training set
   (b) train the model on the training set and evaluate on the test set

Each iteration is called a fold. In general practice, setting n=10 ("10-fold cross validation") is accepted as giving a highly accurate estimate of the generalization error of a model. However, since 10 folds means retraining the models 10 times, 10-fold cross validation may be computationally expensive for large and/or complicated data sets. In such cases, 5-fold cross validation or 10-fold cross validation run on a subset may be appropriate.
Chapter 3

Dataset

3.1 Introduction

The dataset is a critical point when developing systems based on machine learning. In order to well estimate the system parameters, many samples have to be used to train it. In this section we look for public available datasets that fit with our purpose: facial images labelled with age and gender.

Because of its non-rigidity and complex three-dimensional structure, the appearance of a face is affected by a large number of factors including face pose, illumination, facial expression, age, occlusion, and facial hair. The development of algorithms robust to these variations requires databases of sufficient size that include carefully controlled variations of these factors. Furthermore, common databases are necessary to comparatively evaluate algorithms. Collecting a high quality database is a resource-intensive task.

There are several databases with facial images, but not all of them are useful at the same level, so they are described in the following sections. They are summarized in the Table 3.1.

<table>
<thead>
<tr>
<th>Dataset name</th>
<th>Images</th>
<th>Gender</th>
<th>Age</th>
<th>Eyes</th>
<th>Mouth</th>
<th>Nose</th>
<th>Race</th>
</tr>
</thead>
<tbody>
<tr>
<td>Feret</td>
<td>2,409</td>
<td>✓</td>
<td>-</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>-</td>
</tr>
<tr>
<td>FDP</td>
<td>568</td>
<td>✓</td>
<td>-</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>-</td>
</tr>
<tr>
<td>G200</td>
<td>200</td>
<td>✓</td>
<td>-</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>-</td>
</tr>
<tr>
<td>Gallagher</td>
<td>931</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Morph</td>
<td>55,134</td>
<td>✓</td>
<td>✓</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>✓</td>
</tr>
</tbody>
</table>

Table 3.1: The information contained in each database

Once proved the importance of the dataset, in the next section all the datasets used are explained, with its performance, statistics, origin, and some problems that we solved.

3.2 Feret

Feret\(^1\) is a freely available database from “The Facial Recognition Technology”. There are 993 subjects and 2,409 frontal face images, which 1,495 are males and 914 are females. Moreover

\(^1\)http://www.itl.nist.gov/iai/humanid/feret/feret_master.html
it has the facial keypoints labelled with the position of the eyes, nose and mouth, which may be very helpful for the image preprocessing.

In addition, this database has the age labelled. However we are sure that some ages are not correctly labelled, thus we will not use this database for the age estimation.

### 3.3 FDP

The FDP database is obtained from the Zagreb University. Although there are a lot of images, only some of them are labelled with the gender or age and facial keypoints. We choose the images that are labelled with the facial keypoints and we create a new subset. This subset has 568 frontal images, of which 364 males and 204 females. Like in Feret, this database has the age labelled, but there is incoherence with the age in some subjects. Thus the labels are not correct and we cannot use them to train the age estimation system.

### 3.4 G200

G200 is a dataset made by ourselves and we named it G200 because there are 200 images with the gender labelled. We found the images on the net and we labelled the gender and the facial keypoints one by one. There are 200 images from 200 subjects: 98 males and 102 females. The name comes from 'Gender 200 images'.

### 3.5 Gallagher

This Gallagher\(^2\) dataset is a collection of typical digital image snapshots, captured in real life at real events of real people with real expressions. The 931 faces with 32 identities in each of the 589 images are labelled with the age, the gender and the eye position. The ages vary between 1 and 32 years. However, we obtained incoherencies with the age of people older than 6 (some of them are labelled incorrectly), hence we discard that subset of subjects and we used only the children and young people.

Despite the eye position is labelled, we found some inaccuracies (because the eye position is placed at the same y for both eyes, and sometimes the face of the subject is rotated) thus the eyes are not localized correctly.

#### 3.5.1 Gallagher subset

As mentioned before, we create a new sub-set using the images of the young people. This dataset is created in order to have a representative group of these ages. In the Figure 3.1, the histogram of the ages in the sub-set can be seen.

Observe that there are just six ages, since 1 to 6. We do not have more representative data, but we expected that would be enough to describe the young people.

\(^2\)http://chenlab.ece.cornell.edu/people/Andy/GallagherDataset.html
3.6 Morph

Morph\(^3\) is the largest database used in this project. It is the Non-Commercial part, however it costs about 100$. It contains 55,134 unique images of more than 13,000 individuals, spanning from 2003 to late 2007. It has the age, gender and race labelled. The age range goes from 16 to 77. The Table 3.2 and Table 3.3 show the distribution of images by gender, ancestry and age.

\begin{table}[h]
\centering
\begin{tabular}{|l|c|c|c|c|c|c|}
\hline
 & African & European & Asian & Hispanic & "Other" & TOTAL \\
\hline
Male & 36,832 & 7,961 & 141 & 1667 & 44 & 46,645 \\
Female & 5,757 & 2,598 & 13 & 102 & 19 & 8,489 \\
\hline
TOTAL & 42,589 & 10,559 & 154 & 1,769 & 63 & 55,134 \\
\hline
\end{tabular}
\caption{Distribution of Morph images by race and gender.}
\end{table}

\begin{table}[h]
\centering
\begin{tabular}{|l|c|c|c|c|c|c|}
\hline
 & <20 & 20-29 & 30-39 & 40-49 & 50+ & TOTAL \\
\hline
Male & 6,638 & 14,016 & 12,447 & 10,062 & 3,482 & 46,645 \\
Female & 831 & 2,309 & 2,910 & 1,988 & 451 & 8,489 \\
\hline
TOTAL & 7,469 & 16,325 & 15,357 & 12,050 & 3,933 & 55,134 \\
\hline
\end{tabular}
\caption{Distribution of Morph images by age and gender.}
\end{table}

3.6.1 Morph subset

This is a sizeable large dataset. We want to reduce the size of the dataset in order to make the age study possible. Hence, a subset of a representative data is generated. We try to have, for each race, the same number of males and females for age range, if possible.

Firstly we split the database by races. Then for each race, we fragmented the age in ranges form two years (from 16 to 17, from 18 to 19 and so on) and for each range we select

\footnote{\url{https://ebill.uncw.edu/C20231_ustores/web/product_detail.jsp?PRODUCTID=8}}
randomly as maximum 30 males and 30 females. The final dataset histogram can be seen in Figure 3.2 and Figure 3.3.

![Histogram of ages and races from the Morph subset.](image1)

![Histogram of ages and gender from the Morph subset.](image2)

Figure 3.2: Histogram of ages and races from the Morph subset.

Figure 3.3: Histogram of ages and gender from the Morph subset.

The total amount of images in that subset is 3,260, which 1,944 are males and the others 1,316 are females. With this selection, all the ages are approximately equal represented (the histogram is almost plane), so the system would not fit to the age range with more subjects.

### 3.7 Gender database

In this section we describe the database that we used to train and test the gender estimation system. We create our database mixing three databases: FERET, FDP and G200.

All these images are labelled with the position of the eyes, mouth and nose. We did not use the Morph dataset because we had not obtained the database when we developed the gender estimation.
3.8 Age database

The age database is created by mixing the subsets with the age labelled described above (Gallagher and Morph). The resulting database has 3,500 images and its age distribution can be seen in the Figure 4.

![Age database histogram by ages.](image)

Observe that there are no subjects between 6 and 16 years with the ages labelled, so it may be a problem. As is said before the gender of the Gallagher subset is not labelled, so the 55.5% are males, the 37.6% are females and the left 6.9% are non-gender labelled coming from the Gallagher dataset. The major part of the race of the database is African or European (82.2%) and the left are Asian, Hispanics and Others.

3.9 Experiments

In order to realize how important it is having a wide range of samples and to have a big enough dataset, some tests have been run.

If we implemented the system with any dataset (Feret for example) (in this test neither the features nor the estimators used are important) we would be able to obtain a quite small test error, but when we applied new inputs from any other dataset to the system it would work so badly, so we can quickly realized that the dataset is not representative enough. This happens because the estimator is overfitted to the dataset.

We tried to prove in a numerical way that the overfitting problem exists. For doing so, we use the Feret to train the system and obtain a train and test error, and the FDP for analysing the behaviour of the system. The errors obtained are showed in Table 3.5.
3.9. EXPERIMENTS

CHAPTER 3. DATASET

<table>
<thead>
<tr>
<th>Classifier trained with (FERET)</th>
<th>FDP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Train error</td>
<td>Train error</td>
</tr>
<tr>
<td>1,15%</td>
<td>6,67%</td>
</tr>
</tbody>
</table>

Table 3.5: Error obtained training with one dataset and testing with another.

We put the two databases together (Feret and FDP) and we use both for train and test, and the results obtained are presented in Table 3.6.

<table>
<thead>
<tr>
<th>Classifier trained with (FERET+FDP)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Train error</td>
</tr>
<tr>
<td>1,18%</td>
</tr>
</tbody>
</table>

Table 3.6: Error obtained merging the two databases and using them in train and test.

Both errors are higher than before, but these values are much trustable, and when we use new inputs, the performance is as expected.

In order to solve the problem we added other databases, so not all the images have the same illumination, size or are cropped at the same points and the variety increases. It is not only important to have a large database, but to have a lot of images very different ones from the others, coming from different sources to ensure that all the possibilities are represented. Doing so, the test error obtained is representative of the system performance.
Chapter 4

Our implementation

The goal of this chapter is to explain how we used the existing theory (explained in Chapter 2) to develop the system. Following the same scheme shown in Chapter 2, we give a detailed explanation of how the parameters have been chosen and which values are used. While the information given in Chapter 2 is generic and useful for all cases, the information given in the current chapter is adapted to this project.

4.1 Preprocessing

In order to make the feature extraction step easier, a set of functions to preprocess the input images has been developed. The preprocessing step follows the scheme seen in Figure 4.1.

![Figure 4.1: Preprocessing scheme](image)

4.1.1 RBG to GRAY

The first step is to convert the input images from RGB to grey space (or capture it directly in grey space if possible). The transformation from RGB to grey space is done using the equation

$$Y = 0,3R + 0,59G + 0,11B$$

but we do not need to make our implementation because we can use the Matlab function `Y_image=rgb2gray(RBG_image)`. Even though the colour information is useful for classifying, we decided not to use it, so we can have a faster system (the carried information is one third of the original amount) and almost as good as if we had the colour information.

4.1.2 Eye tracker

We use the eyes as keypoints in the face, and many further steps assume that the eye position is known. As said in the Chapter 3, in some databases the eye positions are annotated, while for some we need to detect them properly. Since the goal of this project is not eye detection,
we use an external code given by VISAGE TECHNOLOGIES for doing so. This code needs as an input the facial image and returns the position of both eyes. In order to know which the left eye is and which the right, we will assume that the face will never be more rotated respect to the horizontal axis than 90 grades, from any of both sides.

4.1.3 Rotation

The next step is an image rotation. Since we are interested in some concrete face objects that can give particular information, we need all of them located in the same area for every image. Thus, we are able to make the feature extraction obtaining the information from the same points, and the comparison between features from different images makes sense. The criterion for the rotation is to place both eyes in the same vertical coordinate (Y), as we assume that everybody has the eyes at the same height. Therefore, the eye position must be known. Once we know the eye position, we need to compute the rotation angle. The angle formed by the imaginary line joining the two eyes and the horizontal axis is computed (see Figure 4.2) with the formula

$$\phi = \arctan \left( \frac{le(y) - re(y)}{le(x) - re(x)} \right)$$

where le refers to the left eye, re to the right one, and x and y the position in horizontal and vertical axis respectively. Notice that using this technique, the angle changes from a negative value to a positive one when it is around 90 grades because of the inverse of the tangent function. Thus, when the $\phi$ is bigger than 90 grades the rotation is done in the wrong side, getting an upside down image. This is not a problem since, as mentioned before, we assume that the angle will never be bigger than 90 grades, which is not a really hard restriction because in the most of applications that this system can have, all the pictures will be taken in a horizontal way (with maybe just a very little rotation).

![Figure 4.2: Rotation angle calculation.](image)

When the angle is known, the image rotation is done with a Matlab function that has the image to rotate and the angle as input parameters, and as output the image rotated $I=\text{imrotate}(I,\text{angle})$. Once the image is rotated, the keypoints position also needs to be rotated. For doing so, the imrotate function is not useful, thus some transformation matrices
have to be used. First of all we need replace the axis in the centre of the image (Matlab places
the axis in the upper-left corner), so the (0, 0) point will be in the middle instead of in the
upper-left corner. This transformation is done with the matrix

\[ M_1 = \begin{pmatrix}
1 & 0 & 0 \\
0 & 1 & 0 \\
-X/2 & -Y/2 & 1
\end{pmatrix} \]

where \(X\) and \(Y\) are the size of the original image in both horizontal and vertical directions. Then the rotation matrix

\[ M_{rot} = \begin{pmatrix}
\cos(-\phi) & \sin(-\phi) & 0 \\
-\sin(-\phi) & \cos(-\phi) & 0 \\
0 & 0 & 1
\end{pmatrix} \]

can be applied in order to obtain the position of the points in the new space, where \(\phi\) is the
same angle as found before. Finally, we need to replace the axis in the upper-left corner again
with a matrix similar to \(M_1\) to avoid having negative values. For calculate the values of this
matrix, first of all we need to know where the corners of the original image have been placed
after the transformation, hence we create a matrix with the position of the four corners and
we apply the rotation \(M_1 \ast M_{rot}\) to this matrix. Then, we just need to move the 0 value of
the axis to the lowest value of the rotated corners rotated. Therefore, we find the lowest
value for the X-axis and for the Y-axis, and these values are the ones that will be used in the
transformation matrix

\[ M_2 = \begin{pmatrix}
1 & 0 & 0 \\
0 & 1 & 0 \\
\min X & \min Y & 1
\end{pmatrix} \]

To sum up, the rotation is done using the formula

\[ rot = (e_{rx}, e_{ry}, 1) = (e_x, e_y, 1) \ast M_1 \ast M_{rot} \ast M_2 \]

where \(e_x, e_y\) are the known values of the position of one eye, and \(e_{rx}, e_{ry}\) are the position
rotated in horizontal and vertical direction respectively. See in Figure 4.3 a scheme of all the
steps done in order to obtain the rotation.

![Figure 4.3: Keypoints rotation scheme](image)
4.1.4 Crop

It is frequently the case that the input image has more information than just the face, so the background and part of the body also appear in the image. Since we will only use facial information we need the image to be fitted to the face due to avoid this extra information than will only affect the system performance. For doing so, we studied which is the averaged face size and which is the relationship between the face size and some facial distances. These distances are computed with the equations:

\[
\text{dist}_{\text{eyes}} = \text{right}_\text{eye}(x) - \text{left}_\text{eye}(x) \quad (4.1)
\]

\[
\text{dist}_{\text{eye-mouth}} = \text{eye}(y) - \text{mouth}(y) \quad (4.2)
\]

\[
\text{midpoint}_{\text{eyes}} = \frac{\text{left}_\text{eye}(x) - \text{right}_\text{eye}(x)}{2} + \text{right}_\text{eye}(x) \quad (4.3)
\]

Then the crop boundaries are calculated with

\[
\text{lower}_\text{boundary} = \text{eyes}(y) - \text{dist}_{\text{eye-mouth}} \times 1.5 \quad (4.4)
\]

\[
\text{upper}_\text{boundary} = \text{mouth}(y) + \text{dist}_{\text{eye-mouth}} \times 0.7 \quad (4.5)
\]

\[
\text{right}_\text{boundary} = \text{midpoint}_{\text{eyes}} - \text{dist}_{\text{eyes}} \times 1.3 \quad (4.6)
\]

\[
\text{left}_\text{boundary} = \text{midpoint}_{\text{eyes}} + \text{dist}_{\text{eyes}} \times 1.3 \quad (4.7)
\]

We look for a trade-off between ensuring that the whole face appears on the image, and guarantee that only the minimum background appears on it. Therefore, we established initial values that approximate the crop and are not very restrictive for any of both cases, but those will be studied in further sections and may be modified afterwards.

4.1.5 Resize

The amount of features that returns some feature extraction functions depends on the number of pixels in the image (for example in Gradient or Hessian). Due to the fact that we need to extract the same amount of features from each input sample, all the images have to be resized to a fixed size.

Like in crop, the image size is a concern point in the project development and thereby initial values are set up, but they will be studied and adjusted to the ones that fit the best. The initial values are 256 pixels in height and 192 pixels in width. The resize is done with the Matlab function \texttt{im\_r=imresize(im,[256,192])} which allows as an input the original image and the size of the new one, and returns as an output the resized image. If wanted, instead of the new size, the input can be the relation between the size wanted and the original size; hence the aspect ratio will not be modified.

Whether making the images bigger or smaller, altering the number of pixels means to have pixels in new positions and therefore their value needs to be estimated. Matlab function allows us different ways to interpolate the values of the new pixels and we used the bicubic interpolation (the default one), in which the output pixel value is a weighted average of pixels in the nearest 4-by-4 neighbourhood.
This is the description of the resize Matlab function, which we used to modify the size of the images:

\[
B = \text{imresize}(A, [\text{numrows} \text{ numcols}]) \text{ returns image } B \text{ that has the number of rows and columns specified by } [\text{numrows} \text{ numcols}].
\]

\[
B = \text{imresize}(A, \text{scale}) \text{ returns image } B \text{ that is scale times the size of } A. \text{ The input image } A \text{ can be a grayscale, RGB, or binary image. If scale is between 0 and 1.0, } B \text{ is smaller than } A. \text{ If scale is greater than 1.0, } B \text{ is larger than } A.
\]

In some type of features, we have an internal parameter called Downsampling factor \((df)\). This factor is which controls the size reduction, hence using \(df = 2\) the size of the image is reduced to the half, and so on. Note that all the input images have a different aspect ratio, and we force them to have the same size (and aspect ratio). Consequently the appearance of the face is altered but, because we do the crop in function of the eyes and mouth position, all the images have the eyes \((x \text{ and } y)\) and mouth \((y)\) in the same position.

### 4.1.6 Histogram equalization

Depending on the environment conditions at the time to take the picture, there may be many differences between the illumination and contrast of different images on the same dataset, which can make the estimation more difficult. For analysing so, the first is step is to compute the histogram.

In statistics, a histogram is a graphical representation of the distribution of data. It is an estimate of the probability distribution of a variable and was first introduced by KARL PEARSON [1]. A histogram is a representation of tabulated frequencies, shown as adjacent rectangles, built over discrete intervals (bins), with an area equal to the frequency of the observations in the interval (usually are chosen to be of the same size). The height of a rectangle is also equal to the frequency density of the interval, i.e., the frequency divided by the width of the interval. The total area of the histogram is equal to the number of data. The histogram does not represent spatial information.

An image histogram is a type of histogram that acts as a graphical representation of the tonal distribution in a digital image. It plots the number of pixels for each tonal value. By looking at the histogram for a specific image a viewer will be able to judge the entire tonal distribution at a glance.

The next step is the histogram equalization, this is, trying to have the image histogram as flat as possible, so there is almost the same amount of pixels at each bin. Thus, all the images have almost the same histogram and consequently similar illumination and contrast. In Figure 4.4 can be seen the equalization of an image, and the difference between the equalized and the non-equalized image histogram.

The probability of an occurrence of a pixel of level \(i\) in the image is

\[
p_x(i) = p(x = i) = \frac{n_i}{n}, \quad 0 \leq i \leq L
\]

\(L\) being the total number of gray levels in the image, \(n\) being the total number of pixels in the image, and \(p_x(i)\) being in fact the image’s histogram for pixel value \(i\), normalized to \([0, 1]\). We can also define the cumulative distribution function as

\[
cdf_x(i) = \sum_{j=0}^{i} p_x(j)
\]
which is also the images accumulated normalized histogram. A transformation to the output 
Y has to be done in order to flatten the histogram. The optimal transformation for the 
equalization is the cumulative distribution function, which will map the original values to 
the new ones, expanding the values with more samples and contract the values with less 
representation.

![Image](213x580) **Figure 4.4: Example of histogram equalization**

The histogram on a colour image has to be computed for each component, but we work 
with images in grey space, so just one histogram is needed. For doing so we use the Matlab 
function `im_eq=histeq(I)`.

```
histeq enhances the contrast of images by transforming the values in an intensity image,
or the values in the colormap of an indexed image, so that the histogram of the output
image approximately matches a specified histogram.
```

histeq enhances the contrast of images by transforming the values in an intensity image, 
or the values in the colormap of an indexed image, so that the histogram of the output image 
approximately matches a specified histogram.

### 4.2 Used image features

#### 4.2.1 Pixel difference

Our implementation of the pixel difference function in Matlab allows the user to select to 
extract all the possible differences, only the ones using the pixels from the same row and 
column, or a specific amount. Neither all the differences nor the row and column differences 
functions have been used in the system development, due to the fact that they return a very 
large amount of features. Therefore, we only use the function that allows you to select how 
many features are wanted, and this parameter will be studied in further sections.

We have not fixed a threshold to compute the difference, so we just calculate the difference 
and save the value. If the estimator needs to use the pixel difference as an input for the system, 
it will set up a threshold, which will be much better that the one that we could establish, so
4.2. USED IMAGE FEATURES

Chapter 4. Our Implementation

this is the reason why we just compute the difference with any threshold and let the estimator do it.

4.2.2 Gradient

The gradient is one of the most typical “low level features” in image processing. As it is explained in Chapter 2.3.2 there are many methods to compute it. In our case, we implement the gradient using the Matlab function $[F_x, F_y] = \text{gradient}(F)$.

$$[F_X, F_Y] = \text{gradient}(F)$$

where $F$ is a matrix returns the $x$ and $y$ components of the two-dimensional numerical gradient. $F_X$ corresponds to $\frac{\partial F}{\partial x}$, the differences in the $x$ (horizontal) direction. $F_Y$ corresponds to $\frac{\partial F}{\partial y}$, the differences in the $y$ (vertical) direction. The spacing between points in each direction is assumed to be one.

In this project $F$ is an NxM sized image. Consequently, $F_X$ and $F_Y$ are also images of NxM (with the information of horizontal and vertical variation respectively). The gradient in each position is computed using the equation and in the edges are approximated by the equation.

$$F_X(i,j) = \frac{f(i+1,j) - f(i-1,j)}{2} \quad F_Y(i,j) = \frac{f(i,j+1) - f(i,j-1)}{2}$$ (4.8)

$$F_X(1,j) = \frac{f(1,j) - f(2,j)}{2} \quad F_X(N,j) = \frac{f(N-1,j) - f(N,j)}{2}$$

$$F_Y(i,1) = \frac{f(i,1) - f(i,2)}{2} \quad F_Y(i,M) = \frac{f(i,M-1) - f(i,M)}{2}$$ (4.9)

In order to have a 1D vector with the values, the two gradient images are reshaped in a single NxM vector, and then they are concatenated forming an output vector of NxMx2 values. The first half of this vector contains the information of $F_X$ and the second half $F_Y$.

4.2.3 Hessian

Together with the gradient, it is a very common feature. As explained before, it can be considered as the gradient of the gradient, thus we implement the previous gradient function twice. When the gradient of the input image is computed we obtain $F_X$ and $F_Y$, and then we compute the gradient over the previous outputs. Consequently we obtain four matrixes coming from the first gradients: $F_{XX}$, $F_{XY}$, $F_{YY}$ and $F_{YX}$. However, it is known that $F_{XY}$ and $F_{YX}$ are exactly the same, so we can discard one.

Each of the resultant matrixes has the same size than the input image (NxM). The concatenation of the three hessian images will have a 3xNxM dimension vector.

4.2.4 Histogram of Oriented Gradients

The Histogram of Oriented Gradients (HOG) implementation has been done with a Matlab function found in the net\(^1\). This function allows choosing the number of windows in both horizontal (Wx) and vertical (Wy) directions and the number of bins (B). The default gradient filter is the simple $[1, 0, -1]$ in both directions, hence two images are obtained: one with the

\(^1\)http://www.mathworks.com/matlabcentral/fileexchange/28689-hogdescriptor-for-matlab
4.2.5 Ratios

This type of feature consists of computing some facial ratios using the eyes, nose and mouth positions (keypoints), hence it can only be done if we have all this information annotated. The calculation of these ratios is done after the rotation step, thus the eyes are forced to be aligned. Note that the extraction of the facial ratios is done before the crop and resize in the preprocessing step because they force all the images to have the eyes and mouth in almost the same position, thus the ratios of all the images would be almost the same. In order to compute the ratios, the first step is to calculate the distance between keypoints. With the eyes aligned, eight distances can be computed, three on y-axis and five on x-axis:

1. dist\text{eyes} = \text{right\_eye}(x) - \text{left\_eye}(x)
2. dist\text{eye\_mouth} = \text{eye}(y) - \text{mouth}(y)
3. dist\text{eye\_nose} = \text{eye}(y) - \text{nose}(y)
4. dist\text{nose\_mouth} = \text{nose}(y) - \text{mouth}(y)
5. dist\text{eye\_Right\_nose} = \text{right\_eye}(x) - \text{nose}(x)
6. dist\text{eye\_Left\_nose} = \text{nose}(x) - \text{left\_eye}(x)
7. dist\text{eye\_Right\_mouth} = \text{right\_eye}(x) - \text{mouth}(x)
8. dist\text{eye\_Left\_mouth} = \text{dist\_mouth}(x) - \text{left\_eye}(x)

Then, using the previous distances, all the possible ratios are computed. Note that dist1/dist2 and dist2/dist1 give the same information. Consequently with the 8 abovementioned distances, 28 ratios can be calculated:

1. ratio1 = \frac{\text{dist\text{eye\_mouth}}}{\text{dist\_eyes}}
2. ratio2 = \frac{\text{dist\text{eye\_nose}}}{\text{dist\_eyes}}
3. ratio3 = \frac{\text{dist\text{nose\_mouth}}}{\text{dist\_eyes}}
4. ratio4 = \frac{\text{dist\text{eye\_Right\_nose}}}{\text{dist\_eyes}}
5. ratio5 = \frac{\text{dist\text{eye\_Left\_nose}}}{\text{dist\_eyes}}
6. ratio6 = \frac{\text{dist\text{eye\_Right\_mouth}}}{\text{dist\_eyes}}
7. ratio7 = \frac{\text{dist\text{eye\_Left\_mouth}}}{\text{dist\_eyes}}
8. ratio8 = \frac{\text{dist\text{eye\_nose}}}{\text{dist\text{eye\_mouth}}}
9. ratio9 = \frac{\text{dist\text{nose\_mouth}}}{\text{dist\text{eye\_mouth}}}
10. ratio10 = \frac{\text{dist\text{eye\_Right\_nose}}}{\text{dist\text{eye\_mouth}}}
11. ratio11 = \frac{\text{dist\text{eye\_Left\_nose}}}{\text{dist\text{eye\_mouth}}}
12. ratio12 = \frac{\text{dist\text{eye\_Right\_mouth}}}{\text{dist\text{eye\_mouth}}}
13. ratio13 = \frac{\text{dist\text{eye\_Left\_mouth}}}{\text{dist\text{eye\_mouth}}}
14. ratio14 = \frac{\text{dist\text{nose\_mouth}}}{\text{dist\text{eye\_mouth}}}
15. ratio15 = \frac{\text{dist\text{eye\_nose}}}{\text{dist\text{nose\_mouth}}}
16. ratio16 = \frac{\text{dist\text{eye\_Left\_nose}}}{\text{dist\text{nose\_mouth}}}
17. ratio17 = \frac{\text{dist\text{eye\_Right\_mouth}}}{\text{dist\text{nose\_mouth}}}
18. ratio18 = \frac{\text{dist\text{eye\_Left\_mouth}}}{\text{dist\text{nose\_mouth}}}

4.2. USED IMAGE FEATURES

CHAPTER 4. OUR IMPLEMENTATION

variations in x (FX) and the other in y (FY). These two images are combined to get the magnitude and orientation matrices using the equations XX and XX (chapter 2 Section 2.3.4). These matrices are divided in windows with an overlap of the 50% between the windows. Instead of computing the histogram of the gradient, the implemented function calculates the cumulative distribution function (cdf) for every window. Note that the histogram and the cdf are different ways to represent the same information. The outcome of the function is a vector that concatenates the values of each window. The size of this vector is $B \ast W_x \ast W_y$. 

4.2.5 Ratios

This type of feature consists of computing some facial ratios using the eyes, nose and mouth positions (keypoints), hence it can only be done if we have all this information annotated. The calculation of these ratios is done after the rotation step, thus the eyes are forced to be aligned. Note that the extraction of the facial ratios is done before the crop and resize in the preprocessing step because they force all the images to have the eyes and mouth in almost the same position, thus the ratios of all the images would be almost the same. In order to compute the ratios, the first step is to calculate the distance between keypoints. With the eyes aligned, eight distances can be computed, three on y-axis and five on x-axis:

1. $\text{dist}_{\text{eyes}} = \text{right\_eye}(x) - \text{left\_eye}(x)$
2. $\text{dist}_{\text{eye\_mouth}} = \text{eye}(y) - \text{mouth}(y)$
3. $\text{dist}_{\text{eye\_nose}} = \text{eye}(y) - \text{nose}(y)$
4. $\text{dist}_{\text{nose\_mouth}} = \text{nose}(y) - \text{mouth}(y)$
5. $\text{dist}_{\text{eye\_Right\_nose}} = \text{right\_eye}(x) - \text{nose}(x)$
6. $\text{dist}_{\text{eye\_Left\_nose}} = \text{nose}(x) - \text{left\_eye}(x)$
7. $\text{dist}_{\text{eye\_Right\_mouth}} = \text{right\_eye}(x) - \text{mouth}(x)$
8. $\text{dist}_{\text{eye\_Left\_mouth}} = \text{dist\_mouth}(x) - \text{left\_eye}(x)$

Then, using the previous distances, all the possible ratios are computed. Note that $\text{dist1}/\text{dist2}$ and $\text{dist2}/\text{dist1}$ give the same information. Consequently with the 8 abovementioned distances, 28 ratios can be calculated:

1. $\text{ratio1} = \frac{\text{dist}_{\text{eye\_mouth}}}{\text{dist\_eyes}}$
2. $\text{ratio2} = \frac{\text{dist}_{\text{eye\_nose}}}{\text{dist\_eyes}}$
3. $\text{ratio3} = \frac{\text{dist}_{\text{nose\_mouth}}}{\text{dist\_eyes}}$
4. $\text{ratio4} = \frac{\text{dist}_{\text{eye\_Right\_nose}}}{\text{dist\_eyes}}$
5. $\text{ratio5} = \frac{\text{dist}_{\text{eye\_Left\_nose}}}{\text{dist\_eyes}}$
6. $\text{ratio6} = \frac{\text{dist}_{\text{eye\_Right\_mouth}}}{\text{dist\_eyes}}$
7. $\text{ratio7} = \frac{\text{dist}_{\text{eye\_Left\_mouth}}}{\text{dist\_eyes}}$
8. $\text{ratio8} = \frac{\text{dist}_{\text{eye\_nose}}}{\text{dist}_{\text{eye\_mouth}}}$
9. $\text{ratio9} = \frac{\text{dist}_{\text{nose\_mouth}}}{\text{dist}_{\text{eye\_mouth}}}$
10. $\text{ratio10} = \frac{\text{dist}_{\text{eye\_Right\_nose}}}{\text{dist}_{\text{eye\_mouth}}}$
11. $\text{ratio11} = \frac{\text{dist}_{\text{eye\_Left\_nose}}}{\text{dist}_{\text{eye\_mouth}}}$
12. $\text{ratio12} = \frac{\text{dist}_{\text{eye\_Right\_mouth}}}{\text{dist}_{\text{eye\_mouth}}}$
13. $\text{ratio13} = \frac{\text{dist}_{\text{eye\_Left\_mouth}}}{\text{dist}_{\text{eye\_mouth}}}$
14. $\text{ratio14} = \frac{\text{dist}_{\text{nose\_mouth}}}{\text{dist}_{\text{eye\_mouth}}}$
15. $\text{ratio15} = \frac{\text{dist}_{\text{eye\_nose}}}{\text{dist}_{\text{nose\_mouth}}}$
16. $\text{ratio16} = \frac{\text{dist}_{\text{eye\_Left\_nose}}}{\text{dist}_{\text{nose\_mouth}}}$
17. $\text{ratio17} = \frac{\text{dist}_{\text{eye\_Right\_mouth}}}{\text{dist}_{\text{nose\_mouth}}}$
18. $\text{ratio18} = \frac{\text{dist}_{\text{eye\_Left\_mouth}}}{\text{dist}_{\text{nose\_mouth}}}$
Many of these ratios may be correlated, thus they may have the same information. For example, if the nose is placed exactly between the eyes, the distance between eyes will be twice the distance from any eye to the nose, and consequently the difference of either of both eyes with the nose will be the same. We do not discard these distances at the beginning because we cannot be sure, for example, if the nose is exactly located between eyes. However in order to reduce the feature number and the useless information, the correlation between ratios is studied.

Firstly, the preceding 28 ratios are computed for all the dataset images (only in gender database, 3,177 images) and concatenated in a 28x3.177 matrix. Then, the correlation matrix is calculated with the Matlab function \texttt{corrcoef(X)} with the previous matrix as an input. This function returns matrix \textit{R} of correlation coefficients. This matrix is calculated with the formula

\[
R(i, j) = \frac{\text{cov}(X_i, X_j)}{\sqrt{\text{cov}(X_i, X_i) \cdot \text{cov}(X_j, X_j)}}
\]

where

\[
\text{cov}(X_i, X_j) = E[(X_i - \mu_i)(X_j - \mu_j)]
\]

The correlation coefficient is a measure that determines how similar two variables are. When the absolute value is 1 means perfect correlation and when it is 0 means no correlation, thus absolutely independence between variables. The absolute value is used in our problem, because is not important if the dependence is positive or negative, we only want to know whether they are dependent or not. Then the squared matrix \textit{R} has a size of 28. The diagonal coefficients indicate the correlation of a variable with itself, hence the correlation is 1. The non-diagonals coefficients represent the correlation between the variables associated to the row and column of the position of the coefficient. Therefore, the absolute value of \textit{R}(i, j) is the same of \textit{R}(j, i) so only the upper/lower triangular part of the matrix is needed. Observe the absolute value of upper triangular part of \textit{R} in Figure 4.5.
Figure 4.5: Correlation between facial ratios.
As it is said before, when the correlation coefficient is 1 or close to, it indicates that the two variables are correlated. Three thresholds are established in order to binarise the coefficients correlation, and then determine which the correlated variables are. The used thresholds are 0.85, 0.9 and 0.95, being 0.85 the most restrictive and 0.95 the least.

The selection method done consists in binarise the coefficients correlation with the threshold and obtain a binary matrix. Then start selecting the first ratio and discard the variables that have the binarised correlation coefficient with the first variable equal to 1. Continue with the next non-discarded variable and repeat the same process. The Table 4.1 shows the result of the selection method for each threshold.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0.95</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>0.9</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>0.85</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>√</td>
<td>√</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>√</td>
<td>√</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
</tbody>
</table>

Table 4.1: Chosen ratios for each threshold.

Observe that the features are reduced from 28 to 18, 14 and 8 respectively for each threshold. Since having 18 features is not that much, we select 0.95 as the threshold. Thus, the used ratios are:

- $\text{ratio1} = \frac{\text{dist}_\text{eye}_\text{mouth}}{\text{dist}_\text{eyes}}$
- $\text{ratio2} = \frac{\text{dist}_\text{eye}_\text{nose}}{\text{dist}_\text{eyes}}$
- $\text{ratio3} = \frac{\text{dist}_\text{nose}_\text{mouth}}{\text{dist}_\text{eyes}}$
- $\text{ratio4} = \frac{\text{dist}_\text{eyeRight}_\text{nose}}{\text{dist}_\text{eyes}}$
- $\text{ratio6} = \frac{\text{dist}_\text{eyeRight}_\text{mouth}}{\text{dist}_\text{eyes}}$
- $\text{ratio10} = \frac{\text{dist}_\text{eyeRight}_\text{mouth}}{\text{dist}_\text{eye}_\text{nose}}$
- $\text{ratio11} = \frac{\text{dist}_\text{eyeLeft}_\text{nose}}{\text{dist}_\text{eye}_\text{mouth}}$
- $\text{ratio12} = \frac{\text{dist}_\text{eyeRight}_\text{mouth}}{\text{dist}_\text{eye}_\text{mouth}}$
- $\text{ratio13} = \frac{\text{dist}_\text{eyeLeft}_\text{mouth}}{\text{dist}_\text{eye}_\text{mouth}}$
- $\text{ratio14} = \frac{\text{dist}_\text{nose}_\text{mouth}}{\text{dist}_\text{eye}_\text{nose}}$
- $\text{ratio15} = \frac{\text{dist}_\text{eyeRight}_\text{nose}}{\text{dist}_\text{eye}_\text{nose}}$
- $\text{ratio16} = \frac{\text{dist}_\text{eyeLeft}_\text{nose}}{\text{dist}_\text{eye}_\text{nose}}$
- $\text{ratio19} = \frac{\text{dist}_\text{eyeRight}_\text{nose}}{\text{dist}_\text{nose}_\text{mouth}}$
- $\text{ratio20} = \frac{\text{dist}_\text{eyeLeft}_\text{nose}}{\text{dist}_\text{nose}_\text{mouth}}$
- $\text{ratio21} = \frac{\text{dist}_\text{eyeRight}_\text{mouth}}{\text{dist}_\text{nose}_\text{mouth}}$
- $\text{ratio22} = \frac{\text{dist}_\text{eyeLeft}_\text{mouth}}{\text{dist}_\text{nose}_\text{mouth}}$
- $\text{ratio24} = \frac{\text{dist}_\text{eyeRight}_\text{mouth}}{\text{dist}_\text{eyeRight}_\text{nose}}$
- $\text{ratio25} = \frac{\text{dist}_\text{eyeLeft}_\text{mouth}}{\text{dist}_\text{eyeRight}_\text{nose}}$

4.2.6 Wrinkles

It is studied that a correlation exists between the age and the wrinkles[27, 28], as can be seen in Figure 4.6. In the cosmetics market there are many skin care products its aim is reduce the number of wrinkles of your skin in order to make your looking younger. Is a fact that humans, using the number of wrinkles, are able to do quite good age estimation.
4.2. USED IMAGE FEATURES

The difficulty of using this type of feature is to detect and quantify the amount of wrinkles. In a facial image can appear spots, warts, shadows, glasses, hair, etc... that can make the wrinkle detection hard. In order to avoid as much as possible the false wrinkles detection, the first step purposed is to select the areas where the detection algorithm will be applied [29, 30, 31, 32]. These areas are the forehead and the region below each eye, as shown in Figure 4.7.

Two different methods are used in order to track these regions. If we have the information about the eyes, nose and mouth position we implement one method, which will be modified when only the eyes position is known. The first step is to compute the distance between the eyes using the equation (4.2). Then the forehead is tracked using the equations

\[
\text{lower\_boundary\_forehead} = \text{eyes}(y) - \text{dist}_{\text{eyes}} \times 1.5 \tag{4.10}
\]

\[
\text{upper\_boundary\_forehead} = \text{mouth}(y) + \text{dist}_{\text{eyes}} \times 0.45 \tag{4.11}
\]

\[
\text{right\_boundary\_forehead} = \text{right\_eye}(x) \tag{4.12}
\]

\[
\text{left\_boundary\_forehead} = \text{left\_eye}(x) \tag{4.13}
\]
To track the regions under the eyes, the upper and lower boundaries are given by

\[
\text{lower\_boundary\_under\_eyes} = \text{eyes}(y) - \text{dist}_{\text{eye\_mouth}} \times 0.33
\] (4.14)

\[
\text{upper\_boundary\_under\_eyes} = \text{eyes}(y)
\] (4.15)

And for the region under the right eye, the lateral limits are

\[
\text{right\_boundary\_forehead} = \text{right\_eye}(x) - \text{dist}_{\text{eyes}} \times 0.33
\] (4.16)

\[
\text{left\_boundary\_forehead} = \text{right\_eye}(x) - \text{dist}_{\text{eyes}} \times 0.25
\] (4.17)

Analogously, the lateral limits under the left regions are

\[
\text{right\_boundary\_forehead} = \text{left\_eye}(x) - \text{dist}_{\text{eyes}} \times 0.25
\] (4.18)

\[
\text{left\_boundary\_forehead} = \text{left\_eye}(x) - \text{dist}_{\text{eyes}} \times 0.33
\] (4.19)

Once the regions are tracked, we use an algorithm to quantify the number of wrinkles that appear on each region (Figure 4.8a). The wrinkles are posed as small objects, like short horizontal lines, so the first step is to calculate the gradient in vertical (Figure 4.8b) (note that when we want to detect horizontal objects the vertical gradient has to be computed). In addition, to ensure that only the important vertical frequencies are used, the second gradient is computed too (Figure 4.8c). Then a binarization is done over the second gradient image using the Matlab function \texttt{B=im2bw(I)} , and the threshold used is 0.03 (Figure 4.8d).

\begin{figure}[h]
\centering
\subfloat[Input region]{\includegraphics[width=0.3\textwidth]{input_region.png}} \hspace{0.5cm}
\subfloat[First gradient]{\includegraphics[width=0.3\textwidth]{first_gradient.png}}
\subfloat[Second gradient]{\includegraphics[width=0.3\textwidth]{second_gradient.png}} \hspace{0.5cm}
\subfloat[Binarized image with threshold 0.03]{\includegraphics[width=0.3\textwidth]{binarized_image.png}}
\caption{Wrinkles detection process}
\end{figure}

In the Figure 4.9 the behaviour of the proposed method for wrinkle detection applied over various regions of facial images is shown.
4.2. USED IMAGE FEATURES

CHAPTER 4. OUR IMPLEMENTATION

Observe in Figure 4.9 that the \( \text{Num} \) (referring to number of wrinkles) and the \( \text{Sum} \) (approximation to the wrinkles density) are almost correlated with the age. The first image (4.9a), which corresponds to a one-year-old child, has a lot of noise, however there are no wrinkles detected. Analysing the second image (4.9b), there are no wrinkles apparently, but our method detects some. However the amount is not that big as the third case, a 60-year-old person (4.9c), where the density detected is quite big. The last image (4.9d) belongs to a thirty-one-year-old person, where the tracked region includes the forehead and also part of the hair, and the purposed method do not distinguish between hair, wrinkles, big shadows... Consequently, there could be some images that give outliers, thus we have to take care and know that this type of feature can be very useful but it could have some errors.

4.2.7 Hair

Together with the wrinkles, the information of the hairstyle, hair density, moustache and beard are features easily associable to specific gender or/and age. We can make by ourselves a good approximation between these features and the truth. Anyone can associate a bald subject to an old male; otherwise a face with a long hair it may be a girl, or if it has a moustache or beard it would usually be a male and also the age range can be more a less estimated. That is why we say that these features are “high-level” features, because a clearly connection between the feature and the class that we want to estimate (age and gender) exists.

Our purpose is to know the density of the hair in different parts of the face, depending on the region where we apply the algorithm we talk about hair (forehead), moustache (between nose and mouth) or beard (chin). The purposed method is quite similar to the one used in wrinkle detection. The first step consists of select the areas where the algorithm will be applied later. The areas are selected using the position of the facial keypoints (eyes, mouth and nose) and the distances expressed in equation (4.2). The forehead is tracked with the equations

\[
\text{lower\_boundary\_hair} = \text{eyes}(y) - \text{dist}_{\text{eyes}} \ast 0.9
\]  

(4.20)
4.2. USED IMAGE FEATURES

CHAPTER 4. OUR IMPLEMENTATION

\[ upper_{\text{boundary}}_{\text{hair}} = \text{mouth}(y) - \text{dist}_{\text{eyes}} \times 1.7 \] (4.21)

\[ right_{\text{boundary}}_{\text{hair}} = \text{right}_{\text{eye}}(x) - \text{dist}_{\text{eyes}} \times 0.7 \] (4.22)

\[ left_{\text{boundary}}_{\text{hair}} = \text{left}_{\text{eye}}(x) + \text{dist}_{\text{eyes}} \times 0.7 \] (4.23)

The region where the moustache is analysed is cropped using the equations

\[ dist_{\text{nose-mouth}} = \text{mouth}(y) - \text{nose}(y) \] (4.24)

\[ lower_{\text{boundary}}_{\text{moustache}} = \text{mouth}(y) - \text{dist}_{\text{nose-mouth}} \times 0.33 \] (4.25)

\[ upper_{\text{boundary}}_{\text{moustache}} = \text{nose}(y) + \text{dist}_{\text{eyes}} \times 0.5 \] (4.26)

\[ right_{\text{boundary}}_{\text{moustache}} = \text{right}_{\text{eye}}(x) \] (4.27)

\[ left_{\text{boundary}}_{\text{moustache}} = \text{left}_{\text{eye}}(x) \] (4.28)

and the chin is cropped using

\[ lower_{\text{boundary}}_{\text{chin}} = \text{mouth}(y) - \text{dist}_{\text{nose-mouth}} \times 1.11 \] (4.29)

\[ upper_{\text{boundary}}_{\text{chin}} = \text{nose}(y) + \text{dist}_{\text{eyes}} \times 0.4 \] (4.30)

\[ right_{\text{boundary}}_{\text{chin}} = \text{right}_{\text{eye}}(x) \] (4.31)

\[ left_{\text{boundary}}_{\text{chin}} = \text{left}_{\text{eye}}(x) \] (4.32)

In Figure 4.10 we can see the regions tracked on three different faces. Once the regions are tracked, the hair detection algorithm is applied using as an input the tracked regions. As it has been done with the wrinkle estimation, we want to guess the quantity and the density of hair. In order to calculate the quantity of hair, the region is binarized using a threshold.

That threshold is different for each region:

\[ th_{\text{hair}} = 0.3 \quad th_{\text{beard}} = 0.2 \quad th_{\text{moustache}} = 0.25 \]

We assume that the hair is darker than the skin, so with the binarization all the values higher than the threshold (brighter) are mapped to 0, and the values lower than the threshold (darker) are mapped to 1. Therefore, computing the sum of all the binarized image pixels, the quantity of hair, moustache or beard can be estimated.

To guess the density of the hair we also use a method similar to the wrinkles density one. The structure of the algorithm is the same: Firstly the gradient is computed. Then, the image is binarized and finally the number of objects is counted.
4.2. USED IMAGE FEATURES

CHAPTER 4. OUR IMPLEMENTATION

(a) Hair zone
(b) Moustache zone
(c) Chin (beard) zone

Figure 4.10: Track of hair regions.

However, in the wrinkles method we purposed to compute the vertical gradient in order to detect the horizontal transitions, but in this case we assume that the hair located in the forehead can be either horizontal or vertical, thus both gradients are computed. Otherwise, to detect the moustache and the beard, only the horizontal gradient is calculated, due to the hair located in these parts is usually vertical.

Once the gradient is obtained, the images are binarized and then the objects are counted. Note that in the forehead region two gradients are computed, thus to obtain the hair quantity approximation the sum of the objects counted in each binarized gradient has to be done.

To calculate the gradient, the image binarization and the objects (hair) detection are used with the same Matlab function that we used in wrinkle detection (gradient, im2bw and bwboundaries). The description of these functions can be found in the Section 4.2.6 Wrinkles.

Figure 4.11, Figure 4.12 and Figure 4.13, show the behaviour of hair, moustache and beard detector comparing with two different subjects. The title of the binarized image is the quantity estimation (sum of pixel values of this image), and the title of the binarized gradient image is the hair density estimation (number of found objects).

Figure 4.11: Hair (forehead) detection comparing two subjects. In the left, the original image and the tracked area. Next to it three images: at the top, the selected area, in the middle the binarized image, and at the bottom the binarized gradient. The same structure repeated for the other subject.
4.2. USED IMAGE FEATURES

CHAPTER 4. OUR IMPLEMENTATION

Figure 4.12: Moustache detection example comparing two subjects. In the left, the original image and the tracked area. Next to it three images: at the top, the selected area, in the middle the binarized image, and at the bottom the binarized gradient. The same structure repeated for the other subject.

Figure 4.13: Beard detection example comparing two subjects. In the left, the original image and the tracked area. Next to it three images: at the top, the selected area, in the middle the binarized image, and at the bottom the binarized gradient. The same structure repeated for the other subject.

4.2.8 Adaptation of the distances when the nose and mouth position is missing

Not all the databases have all the keypoints position annotated, thus we have to use external software to do it. As we said in Section 4.1.2 we can find the eyes positions with the software provided for VISTAGE TECHNOLOGIES. However this software do not return the nose and mouth position, so we have to adapt the functions which uses these positions:

\[
\tilde{d}_{\text{nose-mouth}} = \text{dist}_{\text{eyes}} \times 0.6251
\]

\[
\tilde{d}_{\text{eyes-mouth}} = \text{dist}_{\text{eyes}} \times 1.084
\]

\[
\tilde{d}_{\text{eyes-nose}} = \text{dist}_{\text{eyes}} \times 0.4588
\]
4.3 Estimators

4.3.1 Tree

The implementation of the Tree estimator in Matlab has been done using the functions already existing in the MATLAB STATISTICS TOOLBOX. In particular, we use \texttt{class\_tree\_op}, which needs the features and the labels as an input, and returns a binary classification Tree. In the regression case the function is \texttt{RegressionTree.fit}, which works in the same way but returns a regression Tree.

Some parameters can be added to these functions, but we used the default configuration. These parameters control the tree size by fixing a minimum number of observations at each branch node, being the default number 10. This is, a leaf node will not be split in two leaves becoming it a branch node if there are less than 10 training samples in the node. The pruning is also set up, and the pruning criterion established is the one that tries to minimize the error. The criterion for choosing a split is the MSE (Mean Squared Error).

For evaluating the Tree performance we use the function \texttt{resubLoss} in classification. This function needs as an input the tree created and returns the resubstitution loss, meaning the loss computed for the data that has been used to create the tree. This function can only be used for computing the train error, and is only useful for classification. For calculating the test error for either classification or regression, or the train error in regression we use the function \texttt{predict}, which needs as inputs the tree and the samples to evaluate, and returns, for each input sample, its predicted value.

4.3.2 Random Forest

The implementation of the random forest function has also been done using the existing functions in the Matlab toolboxes. For the random forest creation, the used function is \texttt{TreeBagger}, which needs as inputs the features and the labels as well as some parameters to customize and adapt the function to the requirements. The number of trees forming the forest, or either it has to be a classification or regression estimator are parameters to establish. Besides this, we also activate the \texttt{OOBPred} to store information on what observations are out of bag for each tree. This info can be used by \texttt{oobPredict} to compute the predicted class probabilities for each tree in the ensemble. The evaluation is done with the same function than Trees, \texttt{predict}.

4.3.3 Fern

Fern implementation function is not developed in any toolbox included in Matlab. We could find an implementation in the PIOTR’S IMAGE AND VIDEO MATLAB TOOLBOX\textsuperscript{2} in which is possible to find both ferns for classification and regression. The fern creation is done with the functions \texttt{fernsRegTrain} or \texttt{fernsClfTrain} (the first one is for the regression and the second for the classification case), which needs as inputs the features, the labels and some additional parameters, and returns the estimator and also the predicted value for the training samples. This value will be useful for calculating the train error. The parameters we use for the fern creation are to fix the fern depth to 4, the number of ferns to 1000, the number of repetitions per fern to 3 and fix the error function to the MSE in the regression case or combine the probabilities using Bayes assumption.

\textsuperscript{2}http://vision.ucsd.edu/~pdollar/toolbox/doc/
4.3. ESTIMATORS

The evaluation of the ferns is done with the functions \texttt{fernsRegApply} or \texttt{fernsClfApply}, which are functions to predict the output applying the fern to any input.

4.3.4 Support Vector Machines

The Support Vector Machine (SVM) implementation has been done using the Matlab functions \texttt{svmtrain} and \texttt{svmclassify} for training and testing respectively. The train function needs as input the features and labels of the input samples, as well as some options to choose the svm parameters, and returns the estimator and the train error. The parameters to choose are the method to find the separation hyperplane (we use Sequential Minimal Optimization because is the fastest), the fraction of samples that are allowed to violate the KKT conditions (we use between 0.2 to 0.4) and the kernel (we can choose between linear, quadratic, polynomial, Gaussian, Multilayer Perceptron or create our own kernel function).

The classification function needs as inputs the classifier generated by the training function and the features for the samples to classify. It returns the estimated class for each input sample. This svm function can only be used in classification so we just use svm in gender estimation.
Chapter 5

Gender estimation

This chapter contains the explanation and description of all the process done in order to find the best image pre-processing parameters, features, and classifier to estimate the gender of the subjects[33],[34],[35]. Obviously, the choice may change depending on the selection criteria. E.g., it can be done in terms of misclassification, estimation time or computational complexity. As said before, we do not look for the lowest error, but a trade-off between the error and the computational complexity.

The scheme followed in order to develop and to optimize the gender estimation is shown in Figure 5.1. Image pre-processing, feature extraction and estimation are explained in the previous sections. Each of these parts has their own variables and parameters that need to be optimized, but each of them is connected to the others. An exhaustive search is infeasible, and the following approach is therefore adopted:

1. **Local Feature Optimization**
   Analyse and study the behaviour of each feature, with the pre-processing and estimators fixed. Only local optimization, hence changing only the parameter values of one type of feature.

2. **Feature selection**
   In order to know how well each feature type performs, the pre-processing and estimation parameters are fixed and we try to find out which are the best features.

3. **Pre-processing optimization**
   Using the selected features and with the estimator parameters fixed, we optimize the pre-processing step.

4. **Estimator optimization**
   Use the pre-processing and features found in the previous steps and improve the behaviour of each estimator.

All the development is done using the Feret, FDP and G200 (see Chapter 3 Dataset).
5.1. LOCAL FEATURE OPTIMIZATION

CHAPTER 5. GENDER ESTIMATION

Figure 5.1: Scheme of the gender estimation development.

5.1 Local feature optimization

The goal of this part is to know the best parameters for each kind of feature. These are the parameters to optimize can be seen in Table 5.1.

<table>
<thead>
<tr>
<th>Feature name</th>
<th>Parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pixel Difference</td>
<td>Number of differences</td>
</tr>
<tr>
<td>Gradient</td>
<td>Downsampling factor</td>
</tr>
<tr>
<td>Hessian</td>
<td>Downsampling factor</td>
</tr>
<tr>
<td>Histogram of Oriented Gradients (HOG)</td>
<td>Number of bins</td>
</tr>
<tr>
<td></td>
<td>Number of windows</td>
</tr>
<tr>
<td></td>
<td>Downsampling factor</td>
</tr>
</tbody>
</table>

Table 5.1: Parameters to study in gender estimation development.

5.1.1 Pixel difference

Number of pixel differences

The number of pixel differences is important because it determines the number of features, and is thus important for the time and complexity of feature extraction and estimation. Obviously, using less features (number of pixel differences) the system becomes simpler and faster.

To compute the behaviour of the system depending on the number of pixel differences, we use a single tree as estimator because it is the simplest of the considered classifiers. It is important to remark that the exact error value it is not important, we compute that just for the behaviour of the error, and we assume that it is approximately the same for all the classifiers.
5.1. LOCAL FEATURE OPTIMIZATION

We define a variable \( N \), which indicates the number of pixel differences studied. The numbers of pixel differences studied are \( N = 1, 10, 100, 500, 1000, 5000, 10000, 20000, 50000 \) and \( 100000 \). A random sequence is created in order to randomly extract the \( N \) pixel differences. For each \( N \), 50 different sequences are generated, and for each sequence a k-fold (with \( k = 20 \) and tree as a classifier) is calculated. Then, the average of the error for each \( N \) used is obtained in order to get a good approximation of the behaviour, and the results can be seen in Figure 5.2.

![Figure 5.2: Misclassification error depending on the number of pixel differences for Tree.](image)

It can be observed in Figure 5.2 that the misclassification error decreases while the number of pixel differences increases, but there is a knee when 100 differences are used. For larger values of \( N \), the error is decreasing more slowly. Using 100 differences the error is 17.65\%, and with 100,000 it is 14.11\%. That means just a difference of -3\% in misclassification using 1000 times more features. For this reason, 100 pixel differences are selected because is a good trade-off between classification error and number of features.

**Downsampling factor**

The effect of the downsampling factor when using pixel differences is studied in this subsection. This downsampling factor is applied on the image that the pre-processing step returns. We work with image sizes 256x192 (49,152 pixels), thus when a 4 downsampling factor is applied, that means that the image used to compute the differences have a size of 64x48 (3,072 pixels, 42 times less). In this type of feature the downsampling factor is not a very important parameter: despite the number of pixels may change, the number of output features is always the same. Because only the downsampling factor is studied, the other parameters have to be fixed to a single value. We use 100 random pixel differences because it is the number that we selected in the previous subsection. In this section all the classifiers are used with a k-fold cross validation (with \( k = 50 \)). The downsampling factor values used are: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 20, and 32.
5.1. LOCAL FEATURE OPTIMIZATION

5.1.2 Gradient

Downsampling factor

In terms of Gradient, the downsampling factor is an important parameter because the size of the images determines the number of features in the output, and it is related with the complexity of the classifier. As it is shown in the Gradient section, the number of output features using an image of size NxM pixels is \((2 \times NxM)\). The downsampling factor values used are: 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 20 and 32.
Figure 5.4: Misclassification error in Gradient depending on the downsampling factor for each classifier.

We can observe in Figure 5.4 the misclassification error in function of the downsampling factor. The error decreases until downsampling factor 6, where there is a knee and the error starts to increase slowly. However, with downsampling 6 [43, 32] (1376 pixels, 2752 features) the error is 3.7% and with downsampling 16 [16, 12] (192 pixels, 384 features) the error is 5.2%; that means +1.5% error using -86% of features. Consequently we choose 16 as a downsampling factor for the gradient.

5.1.3 Hessian

Downsampling factor

Similarly to the gradient, the downsampling factor, i.e. the size of the input images, is the parameter that determines the number of output features. If the image size is NxM pixels, then the features obtained by the Hessian are 3xNxM. If the image is big, the number of features obtained is a significantly big value (the default image size is 256x192, thus 256x192x3=147,456 features are obtained); consequently we want a downsampling factor that gives a good trade-off between misclassification error and number of features. The values studied are the same of gradient: 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 20 and 32.

Observe in Figure 5.5 that the behaviour on almost all the estimators is that the error decreases as the downsampling factor increases. Otherwise, in support vector machine (in that case the best estimator) the error increase gradually since downsampling 7 (where the minimum is). The difference between the minimum error and the error with downsampling 16 is just 1.19% and the number of features is reduced from 3009 to 576 (-81%), hence we consider that using a downsampling factor of 16 is a good trade-off between misclassification error and number of features.
5.1. LOCAL FEATURE OPTIMIZATION

CHAPTER 5. GENDER ESTIMATION

Figure 5.5: Misclassification error in Hessian depending on the downsampling factor for each classifier.

5.1.4 Histogram of Oriented Gradients

Number of bins
The number of output features that HOG obtain is the product between the number of bins (B) and the number of windows (Wx · Wy). The behaviour of the number of bins is studied but both, the downsampling factor (df) and number of windows (W), are fixed to 1 and 10 respectively.
Figure 5.6: Misclassification error in HOG depending on the number of bins for each classifier.

Analysing the Figure 5.6 we can observe that using one bin (i.e. “without” histogram) the error is high but not as high as expected. Using one single bin means that all the features should be 1, due to all the angles are between the same range ($-\pi$ to $\pi$), but the normalization explained in the Chapter 4 Section 4.2.4 makes that some values are slightly lower than 1, and the estimator uses that variation to make a pretty good classification. Nevertheless, whereas the number of bins increases, the misclassification error decreases. We choose 9 bins because that value is the most used when HOG is implemented, and it can be observed in the plot that the error does not decrease a lot (e.g. the difference between using 9 or 20 bins is just -0.2%). In conclusion, we use 9 bins and thus the angle range (from $-\pi$ to $\pi$) is divided in 9 parts, that is, each bin represents $2\pi/9$ radians (40 degrees).

Number of windows

The number of windows means how many cells the image is divided into. There are two values, the number of cells for each row and the number for each column. To make the problem easier to handle, we consider the same number of windows for row and column. Thereby the cells do not have square shape (the size of the input image is 256x192, and that means that they have an aspect ratio of $\frac{3}{4}$). The downsampling factor is fixed to 1 (i.e. no downsampling) and the number of bins is 9 as seen in the last subsection.
Figure 5.7: Misclassification error in HOG depending on the number of windows for each classifier.

Figure 5.7 shows the misclassification error depending on the number of windows. Using just 1 window (i.e. all the image, without split it) the error is around 20%, but while the number of windows increases, the error decreases approximately until using 10 windows, where the error is almost constant and does not improve more. This is the reason why the number of windows is set to 10.

**Downsampling factor**

The downsampling factor is not an important parameter in terms of output features, because the number of features that HOG generates depend exclusively on the number of windows and bins. However the downsampling factor can be important to know which quality has to have the input image. We studied the behaviour of the downsampling factor using the selected number of bins and number of windows, 9 and 10 respectively, and the results can be seen in Figure 5.8.
The conclusion obtained analysing the results is that the downsampling factor is not a parameter that influences a lot in HOG. Clearly, using downsampling factor 1 (no downsampling) the error is a little bit lower than with 10 downsampling factor, but Figure 5.8 shows that if wanted, we could downsample the input image and the misclassification error would not increase a lot. However, we choose the downsampling 1 because is the best one in terms of misclassification error and the input image is not very big. If it exists a dominant direction in the window where the HOG is applied, even though it is low-pass filtered, the dominant direction is still there and is still dominant, so that is why downsampling does not affect that much.

5.2 Feature optimization

Once each feature is optimized, the next step is a global optimization. There are nine different possible features with these parameters presented in the Table 5.2.
### Chapter 5. Gender Estimation

#### 5.2. Feature Optimization

<table>
<thead>
<tr>
<th>Type</th>
<th>Feature name (Abbr.)</th>
<th>Parameters</th>
<th>Number of features</th>
</tr>
</thead>
<tbody>
<tr>
<td>Low level</td>
<td>Pixel difference (PD)</td>
<td>Num diff. = 100, DF=16</td>
<td>100</td>
</tr>
<tr>
<td></td>
<td>Gradient (Gr)</td>
<td>DF = 16</td>
<td>384</td>
</tr>
<tr>
<td></td>
<td>Hessian (He)</td>
<td>DF = 16</td>
<td>576</td>
</tr>
<tr>
<td></td>
<td>HOG (Ho)</td>
<td>Num Bins = 9, Windows = 10, DF=1</td>
<td>900</td>
</tr>
<tr>
<td>High level</td>
<td>Wrinkles (Wr)</td>
<td>-</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>Moustache (Mo)</td>
<td>-</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>Beard (Be)</td>
<td>-</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>Hair (Ha)</td>
<td>-</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>Facial Ratios (FR)</td>
<td>-</td>
<td>18</td>
</tr>
</tbody>
</table>

Table 5.2: Summary of the available features and their parameters.

The global optimization consists in both, analysing the behaviour of each feature and trying to obtain the best combination. For all the features the estimation is done using only one single feature, so we can know which the best one is. Once it is known which the best one is, another one is added, so we obtain the best combination of two features, and so on until the misclassification error does not decrease. To compute all the steps a k-fold cross validation is used, with k=20 and for each estimator (tree, random forest, Fern and support vector machine).

The first step is to estimate the gender using only one of the nine types of features. The Figure 5.9 shows the results.

![Figure 5.9: Obtained error when classifying with one single feature for each classifier.](image)

For each estimator the type of feature that gives the lowest misclassification error is chosen. As can be seen in Figure 5.9, HOG is the best one for all the estimators. Using Tree the error is 14.%, with Random Forest 7.06%, with Fern 21.29% and using Support Vector Machine 2.28%.

Those results obtained are better than what we expected because the error using SVM is very low, and the next step is to study which type of feature can improve the classification, thus two types of features are used (HOG and another). The results are presented in Figure 5.10.
5.2. FEATURE OPTIMIZATION

CHAPTER 5. GENDER ESTIMATION

The lowest value obtained using one type of feature for each classifier is plotted with a dashed line in order to compare how well the added feature performs. If there is not any feature that, for any estimator, makes the classification better, is not necessary to continue analysing the behaviour in adding more features. This happens when using support vector machine, where the error using only HOG was 2.28% and when other features are added to HOG, the error does not decrease.

Using other classifiers the error decreases, not that much, but sufficiently to continue searching. Classifying with Tree, the error decreases from 14.82% (with HOG) to 13.85% (by HOG and Gradient). When doing so with Random Forest the error also decreases from 7.06% to 5.97% when the Hessian is added to HOG. Finally, using Fern, the error decreases from 21.29% (HOG) to 20.33% (HOG and Pixel Difference).

The last step is repeated but in that case adding a new feature to the selected features for each classifier previously, thus now the classification is done with 3 features. Notice that this step is computed just for the classifiers in which the error decreased when the second feature was added, hence not Support Vector Machine.

The same process as before is done and the results are shown in Figure 5.11. In that case Beard is the third feature that gives less error in tree when is added to HOG and Gradient because the error decreases from 13.85% to 13.36%. For Random Forest and Fern, is Gradient the one that makes a highest reduction of the error when is added to Hog and Hessian, and Hog and Pixel difference respectively because the error goes from 5.97% to 5.78% in Random Forest and form 20.33% to 20.18% classifying with Fern.

The selection method is repeated using now 4 features, the three selected before and a new one.

Figure 5.10: Obtained error for each classifier when classifying with two features.

Figure 5.11: Obtained error for each classifier when classifying with three features.
As can be seen in Figure 5.12, neither using Random Forest nor Fern the error is reduced adding a fourth feature to the tree selected in the last step, thus we can stop trying more features for that classifiers. Using Tree the error decreases a little when Moustache is added to HOG, Gradient and Beard. The error diminish from 13.36% to 13.23%, so the fifth feature which added to these four makes a better classification is searched. The Figure 5.13 shows the results.

Observe in the plot that there is not any feature that, with the previously selected four, makes a better classification because the error is bigger for any feature added. Consequently the process is finished because the error has not improved. We introduce the obtained results in the Table 5.3.

![Figure 5.12: Obtained error for each classifier when classifying with four features.](image)

![Figure 5.13: Obtained error for each classifier when classifying with five features.](image)

<table>
<thead>
<tr>
<th>Number of used features</th>
<th>Tree</th>
<th>Random Forest</th>
<th>Fern</th>
<th>SVM</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 feature</td>
<td>0.1482 (Ho)</td>
<td>0.0706 (Ho)</td>
<td>0.2129 (Ho)</td>
<td>0.0228 (Ho)</td>
</tr>
<tr>
<td>2 features</td>
<td>0.1385 (Gr)</td>
<td>0.0597 (He)</td>
<td>0.2033 (PD)</td>
<td>STOP</td>
</tr>
<tr>
<td>3 features</td>
<td>0.1336 (Be)</td>
<td>0.0578 (Gr)</td>
<td>0.2018 (Gr)</td>
<td>STOP</td>
</tr>
<tr>
<td>4 features</td>
<td>0.1323 (Mo)</td>
<td>STOP</td>
<td>STOP</td>
<td>STOP</td>
</tr>
<tr>
<td>5 features</td>
<td>STOP</td>
<td>STOP</td>
<td>STOP</td>
<td>STOP</td>
</tr>
</tbody>
</table>

Table 5.3: Selection of the best features for each classifier.

In Figure 5.14 the behaviour of adding more features is plotted. In the other cases the
error decreases but very slowly. The highest improvement happens when Tree is used, where the error decreases 1.59%. However, we consider that the behaviour of adding more features is practically constant because on the one hand, the error does not reduce a lot, and, on the other hand, the number of features needed goes from 900 to 1,288 in Tree, to 1,860 using Random Forest and to 1,384 with Fern.

Concluding, we consider Support Vector Machine as the best classifier in terms of misclassification error when it uses Histogram of Oriented Gradients with 9 bins and 10x10 windows and the error in test is 2.28%.

5.3 Preprocessing optimization

In this section the behaviour of the preprocessing is studied. We analyse the importance of each step in the pre-processing step. Remember the preprocessing diagram shown in Figure 5.15.

![Figure 5.14: Behaviour of each classifier when adding the selected features.](image)

Figure 5.15: Pre-processing scheme.

But not every single part is studied. For example, we assume that the rotation is necessary to make a good estimation because it aligns all the faces; hence the rotation is not studied. Otherwise the crop limits, the size of the image and the usefulness of the histogram equalization are studied.
We make the classification with the features obtained in the previous section: HOG with 9 bins and 10 windows for row and column (HOG 9.10.10). In that section we assumed fixed parameters in the crop limits, size of images and we used the histogram equalization. It could be possible that the best features obtained (HOG 9.10.10) may be fitted to the pre-processing parameters chosen, but we assume that the found features are also the best whereas the pre-processing parameters are changed.

The classifiers and its parameters used in this section are the same as in the previous section: Single Tree, Random Forest (with 30 trees), Fern and SVM (with quadratic basis).

5.3.1 Crop limits

The crop limits determines which part of the face is used to extract the features, and the estimate, and how big is this. In this section we study the importance of using an image of the whole face and maybe a part of background, or otherwise, using a crop fitted to the face (maybe in some cases without the ears, part of hair or chin) and avoiding as much as possible he background. The limits used until now are the ones obtained with the equations (4.4) (lower boundary), (4.5) (upper boundary), (4.6) (right boundary) and (4.7) (left boundary). In order to compute the behaviour of the image crop, a cropped_factor (cf) is introduced in the expressions:

\[
\text{lower\_boundary} = \text{eyes}(y) - \text{dist}_{\text{eye\_mouth}} \ast 1.5 \ast \text{cf} \tag{5.1}
\]

\[
\text{upper\_boundary} = \text{mouth}(y) + \text{dist}_{\text{eye\_mouth}} \ast 0.7 \ast \text{cf} \tag{5.2}
\]

\[
\text{right\_boundary} = \text{midpoint}_{\text{eyes}} - \text{dist}_{\text{eyes}} \ast 1.3 \ast \text{cf} \tag{5.3}
\]

\[
\text{left\_boundary} = \text{midpoint}_{\text{eyes}} + \text{dist}_{\text{eyes}} \ast 1.3 \ast \text{cf} \tag{5.4}
\]

That factor controls the crop size, thus when \( cf \) decreases, the crop area is also tighter and the facial image resultant is more restricted. Otherwise, whereas \( cf \) increases, the image uses a larger area of the face, is more “opened” and tends to appear more background. Note that we can optimize each limit by itself, using a different cropped_factor for every limit \((cf_{up}, cf_{down}, cf_{right}, cf_{left})\) but to simplify the problem we assume the same factor for all the limits. The values of the \( cf \) used go from 0.5 to 2, with steps of 0.1. Once the crop is done, the images are resized to a fixed size 256x192 and histogram equalization is also computed.

We can observe in Figure 5.16 that the difference between using a fitted crop or a crop that includes part of the background is insignificant. Curiously we obtain a local minimum in the limits that we used in the former section i.e. with crop factor 1 (using support vector machine). However, using the other estimators the behaviour is practically the same: the error decreases while the \( cf \) increases until the \( cf = 1 \). We guess that the reason of that behaviour is because the features are extracted from a more representative part of the face. Then, when the crop is more and more opened (higher \( cf \) value), the error increases again. We assume that this happens because the background appears and it does not have information useful for the gender classification.

Thus, the parameters that we used to make the first crop are the best ones, so the cropped image limits are the same as the first expression \((cf = 1)\).
5.3. PREPROCESSING OPTIMIZATION

CHAPTER 5. GENDER ESTIMATION

5.3.2 Resize

The resize step is the one that changes the image size to a fixed value. In the cropping step the image size is modified in order to be adjusted to the face, but we need all the images to have the same size, hence a resize to a fixed value is needed. It is important to distinguish between the downsampling factor studied in previous sections and the size fixed in this step: the previous downsampling factor is applied to the output image of the pre-processing step, so it is an internal value of each type of feature function (it had to be studied for every single feature). In this case, the studied parameter is the one that determines the size of the output image in the pre-processing step, and all the features will be affected by the same value.

Note that we have chosen only one type of feature, HOG 9.10.10, and we have fixed the downsampling factor to 1, i.e. without downsampling. Hence, in this case, the study of the behaviour of the downsampling factor as an internal parameter from the HOG function is the same as the size of the pre-processed images.

However, when we studied the downsampling factor we did not consider the possibility of having images larger than 256x192 because we did not use interpolation or downsampling factors smaller than one. We studied since downsampling factor 1 to 10 with steps of 1. That means that we used images from 256x192 to 27x19 pixels. In addition note that the crop used was \( cf = 1 \) but in the previous section we found an optimal one.

Now we want to study the exact performance of the system depending on the image size while using the crop limits selected previously and images larger than 256x192 pixels.
We can observe in Figure 5.17 that the error is bigger when the size of the images is smaller. While the images are larger, the error also decreases. That is what we expected because with small images the high spatial frequencies are lost, and consequently the gradient of those images is not the same. Thus, the histogram of the oriented gradients is different and we lose information.

Note that, using Support Vector Machine, since the size factor goes from 0.3 until 8, the error is established in $2.8 \pm 0.4\%$. That means that we can have images of 77x59 pixels and the error is close to have images of 512x384. However, the number of features does not change, because is determined by the HOG (in this case, using HOG 9.10.10, is 900), so we do not need to have smaller images. Anyway it is good to know, because that means that our system allows quite small images 80x60.

### 5.3.3 Histogram equalization

The last step of the proposed pre-processing method is the histogram equalization. We implement it because we expect that equalizing the images increases the global contrast of the facial images and normalizes the scene illumination of the picture. In order to know the importance of this procedure, the features (HOG 9.10.10) are extracted using the crop limits and size obtained in the previous sections, and the estimation is computed with and without equalization. These are the results:

<table>
<thead>
<tr>
<th></th>
<th>Tree</th>
<th>Random Forest</th>
<th>Fern</th>
<th>SVM</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>TRAIN</strong></td>
<td>HIST EQ</td>
<td>5.10%</td>
<td>0.80%</td>
<td>20.38%</td>
</tr>
<tr>
<td></td>
<td>NO HIST EQ</td>
<td>5.75%</td>
<td>0.79%</td>
<td>21.67%</td>
</tr>
<tr>
<td><strong>TEST</strong></td>
<td>HIST EQ</td>
<td>14.55%</td>
<td>7.23%</td>
<td>21.29%</td>
</tr>
<tr>
<td></td>
<td>NO HIST EQ</td>
<td>14.60%</td>
<td>6.65%</td>
<td>21.30%</td>
</tr>
<tr>
<td></td>
<td>Difference</td>
<td>0.05%</td>
<td>-0.58%</td>
<td>0.01%</td>
</tr>
</tbody>
</table>

Table 5.4: Misclassification rate for each classifier comparing the equalized images against the non equalized.
Observe in Table 5.4 that the there are no error difference between using and not using the histogram equalization. Hence, we can skip that pre-processing step and make the system faster and easier.

We think that the histogram equalization is not needed because the HOG of an equalized histogram image and the HOG of the same image without equalization is almost the same, because the contours or frequencies do not change and HOG is invariant to illumination changes. Hence, the angles of the respective gradient are very close, and the histograms are also similar.

### 5.4 Classifier optimization

Until now the classification was made fixing the estimator and its parameters. However, in order to improve the classifier, the internal parameters of each estimator are studied. We work with four different classifiers: tree, random forest, Fern and support vector machine. As we have seen in the previous sections, support vector machine is the one that gives the lowest misclassification error. Thus we studied the internal parameters of this classifier.

The implementation of Support Vector Machine is done using Matlab. There are many parameters to determine but we focused in the kernel, the parameter that we consider the most important for the classifier. These are the results obtained are presented in Table 5.

<table>
<thead>
<tr>
<th>Kernel function</th>
<th>Train error</th>
<th>Test error</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear kernel</td>
<td>0%</td>
<td>2.61%</td>
</tr>
<tr>
<td>Quadratic kernel</td>
<td>0%</td>
<td>2.69%</td>
</tr>
<tr>
<td>Gaussian Radial Basis Function kernel (with sigma 1)</td>
<td>0%</td>
<td>3.87%</td>
</tr>
<tr>
<td>Multilayer Perceptron kernel</td>
<td>26.89%</td>
<td>26.84%</td>
</tr>
</tbody>
</table>

Table 5.5: Misclassification rate comparing four possible kernels in SVM.

Observe that the best kernels are the simplest ones: linear and quadratic. Using the Gaussian kernel, the classifier overfits to much the database. Therefore, the test error increases. We chose the linear kernel because is the one that gives a lowest misclassification error and it is the easiest, fastest and the one that requires less computational complexity to implement.

### 5.5 Results

We started the gender section with a diagram where we explained the process that the system follows, and with a lot of opened doors and parameters to determine. During that section we have analysed the behaviour of these parameters, we made decisions and discussed the results in order to obtain the parameters that give a good trade-off between the misclassification error and the complexity of the system. In Figure 5.18 we can observe the final system.

It is a very simple system, easier than what we expected because it only uses one type of feature: Histogram of Oriented Gradients. That function gives 900 features, which is quite a huge amount of features. We also have seen that, if needed, we can reduce the number of features using, for example, smaller number of windows and the error does not increase a lot. In addition the classifier is very simple.
5.5. RESULTS

The computational time (related with the computational complexity or number of operations) is an important parameter to study. There are many time gaps to analyse, but we focused in how long takes each of the steps. Note that the time depends on the computer and its power, so it is not important the absolute value, but the relationship with the others.

<table>
<thead>
<tr>
<th>Step</th>
<th>Absolut time (s)</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pre-processing</td>
<td>0.407340 seconds</td>
<td>88.32 %</td>
</tr>
<tr>
<td>HOG 9.10.10 extraction</td>
<td>0.045202 seconds</td>
<td>9.8 %</td>
</tr>
<tr>
<td>SVM Classification</td>
<td>0.008643 seconds</td>
<td>1.88 %</td>
</tr>
<tr>
<td><strong>TOTAL</strong></td>
<td><strong>0.461185 seconds</strong></td>
<td><strong>100 %</strong></td>
</tr>
</tbody>
</table>

Table 5.6: Absolute and relative time that takes each step.

As can be seen in Table 5.6, the classification step is the one that requires the least time. As mentioned, the absolute value is not important, as it depends on the computer. In case of wanting a faster system, the step that requires more work is the pre-processing, hence maybe some parts may be skipped or modified.

Finally, once all the parameters have been selected, in order to know how good our system is, we try to test it with another external database that we obtained when we had finished the gender optimization (that is the reason why we do not study the gender with this database). The database is MORPH, and has 33,421 subjects with the gender labelled. Two experiments are done, with 5-fold because we have a large amount of images. The first experiment consists in simulating a real system, where the subjects of the new database represent the people who test the system; therefore, we test with the new database a trained classifier. These are the results:

<table>
<thead>
<tr>
<th>Database</th>
<th>Tree</th>
<th>Random Forest</th>
<th>Fern</th>
<th>SVM</th>
</tr>
</thead>
<tbody>
<tr>
<td>TEST (Morph)</td>
<td>25.52%</td>
<td>26.99%</td>
<td>25.25%</td>
<td>15.12%</td>
</tr>
</tbody>
</table>

Table 5.7: Missclassification obtained in Morph dataset with the system trained with the another dataset

The error in all the cases is bigger than what we expected. Maybe the reason of that is because the new dataset has more racial variability, and in the dataset that we used to make the classifier this racial variability was not representative.
The second experiment that we did has been done in order to simulate a new system, with the new database. We expected to solve the problem that we have supposed before: the low racial variability. We assume that the parameters of pre-processing, type of features and classifiers that we have chosen are also the best ones with that new database, because those parameters only depend on the gender, and do not depend on the database. However, we suppose that we have a new real-problem only with MORPH database, and we can use that database to train our system. Concluding, in that second experiment we train and test with MORPH dataset. This is the performance of the system:

<table>
<thead>
<tr>
<th>Database</th>
<th>Tree</th>
<th>Random Forest</th>
<th>Fern</th>
<th>SVM</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRAIN (Morph)</td>
<td>8.93%</td>
<td>1.05%</td>
<td>22.41%</td>
<td>0.00%</td>
</tr>
<tr>
<td>TEST (Morph)</td>
<td>18.92%</td>
<td>12.55%</td>
<td>22.57%</td>
<td>6.94%</td>
</tr>
</tbody>
</table>

Table 5.8: Missclassification obtained in Morph dataset with the system trained with the Morph dataset

As we expected, the results are better than in the previous case. We can conclude that with our work we found the best parameters for the pre-processing, features, features selection and classifier, but we do not found the best classifier, thus the training has to be done with a representative data.
Chapter 6

Age estimation

This chapter contains the explanation of the followed method to make good age estimation. It is quite close to the previous Chapter 5 Gender, thus the process is similar. However, there is an important difference between age and gender: the estimation method. As explained before, classification is used to estimate the gender because the system provides a binary output (male or female). In contrast, the age is estimated with regression, thus the prediction output is the estimated age.

This change of estimation paradigm affects basically two parts of the chain. The first one is the estimator: we use a regressor instead of a classifier. Decision Tree, Random Forest and Fern classifiers are quite easy to convert to regressors, it is just change the criterion to assign a class in a leaf (see Chapter 2 Background Theory). However, Support Vector Machine (SVM) is a classifier and, although there are some complex algorithms than can adapt it to a regressor, we decided not to use it.

The second modification is the evaluation method. The average of the absolute error between the predicted age and the truth could be used as error measure, but we consider that a five year error is more grave for a child than for an elderly, so the error should be weighted by the age. This is the reason why the relative error is proposed and implemented with the equation

\[ \hat{e} = \frac{|\text{predicted\_age} - \text{age}|}{\text{age}} \]

Then, we compute, for every system, the amount of samples that have a lower relative error \( e \) than a given value, for several relative error values, and the result is normalized for the total amount of samples. This is called the Relative Error Distribution Function, REDF:

\[ \text{REDF}(x) = \frac{\text{samples with error} \leq x}{\text{total of samples}} = P(\hat{E} \leq x) \]

where \( \hat{E} \) is the function with all relatives errors. Ideally the REDF should pass through the point (0, 1), so that all samples have zero relative error. Examples are given in Figure 6.1. The closer the curve gets to the optimal one (dashed black line) the better the system is. Thus, the curve 1 (red) is better than the curve 2 (blue).

Finally, in order to compare systems, the area under its curve is calculated (AUC). The higher the AUC of REDF is, the better the system estimates. We compute the AUC between 0 and 10 in the x-axis (relative error) and between 0 and 1 in the y-axis (density of samples);
therefore the highest possible value is 10, which will only happen when the REDF pass through the point (0, 1).

The updated gender scheme followed to develop and to optimize the age regression is shown in Figure 6.2.

![Figure 6.1: Examples of REDF (Relative Error Distribution Function).](image1)

Observe in Figure 6.2 that almost the entire scheme is the same as for gender estimation; however, some steps are modified and added (in green) or removed (strikethrough). Firstly,
we assume that the best crop and resize parameters obtained in gender classification are also
the bests for estimate the age. Moreover, the gender classification is added as a feature and
the Facial Ratios are removed because we do not have the mouth and nose keypoints, and
without them the facial ratios cannot be computed. Finally, as mentioned before, we change
the classification to a regression and we do not use SVM. The optimization steps are the same
as for gender estimation:

1. LOCAL FEATURE OPTIMIZATION: Best parameters for each type of function.
2. FEATURE SELECTION: Select the used features.
3. PRE-PROCESSING OPTIMIZATION: Adjust pre-processing factors.
4. REGRESSOR OPTIMIZATION: Study the regressor parameters.

The time required to compute a regressor is quite high. For example, to train a single
regressor Tree with 3,500 samples and 1,000 features it takes about 20 seconds. Therefore, to
compute a Random Forest with 50 trees it takes approximately 10 minutes. With a 20-fold
validation the time required is too high. For this reason we start using a Random Forest with
10 trees, and a k-fold with k=5. We know that using a higher k, the approximation to the
estimation behaviour would be better, however k=5 is enough to have an approximation of
how good each estimation is. Moreover, we assume that the more trees we use in Random
Forest, the better the estimation should be done, but to avoid spending too much time
computing, we started with 10 trees in Random Forest.

All the development is done using the Age database, i.e. subset of Morph and Gallagher
datasets (see Chapter 3 Dataset for more information).

6.1 Local feature optimization

This part is exactly the same than in gender but obviously with different decisions and
different behaviour of each regressor. The types of features optimized are the same as in
gender (its features that have some parameters to optimize). In this section we only explain a
little introduction to each optimized parameter, its behaviour in the regressor and finally the
reached decision. A big explanation of each parameter can be found in Chapter 5 Gender.

Table 1 shows the parameters to optimize in each type of feature.

<table>
<thead>
<tr>
<th>Feature name</th>
<th>Parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pixel Difference</td>
<td>Number of differences</td>
</tr>
<tr>
<td></td>
<td>Downsampling factor</td>
</tr>
<tr>
<td>Gradient</td>
<td>Downsampling factor</td>
</tr>
<tr>
<td>Hessian</td>
<td>Downsampling factor</td>
</tr>
<tr>
<td>Histogram of Oriented Gradients (HOG)</td>
<td>Number of bins</td>
</tr>
<tr>
<td></td>
<td>Number of windows</td>
</tr>
<tr>
<td></td>
<td>Downsampling factor</td>
</tr>
</tbody>
</table>

Table 6.1: Parameters to study in age estimation development (equal to Table 5.1).
6.1. LOCAL FEATURE OPTIMIZATION

6.1.1 Pixel difference

Number of pixel differences

It is important to optimize the number of pixel differences (N) because it controls the number of features that will return the Pixel difference function. This experiment is done using N random differences, and, in order to have a good approximation to the behaviour of the regression, we used values from 1 to 10,000. All the pixel differences are extracted from images sized 256x192 without downsampling. Figure 6.3 shows the conduct of the estimation.

![AUC of REDF in Pixel difference in function of the number of differences (N)](image)

Figure 6.3: AUC in REDF in Pixel difference in function of the number of differences (N), in test.

Note that we looked for the higher AUC, the nearest to 10 (optimal classification). The N that gives a larger area is 1,000, thus is the value that we selected. The AUC increases when more differences are used. This is the expected performance because using more differences the regressor, in principle, can perform at least as good as others with fewer differences. A thousand features is quite a big amount of features, but still manageable.

Downsampling factor

The downsampling factor (DF) is not important in terms of number of features, because these are determined for the number of pixel differences (N). However, it is important to know the images size that the system allows. We fixed the N to 1,000, as selected in the previous step. Then we compute the AUC for the three systems changing de DF from 1 to 16, and the results can be seen in Figure 6.4.

The AUC is lower for the bigger downsampling factor values, because some parts of the images are removed. However the behaviour is quite flat and constant. This happens because we used exactly the same N and vary DF. If we had not fixed N, the result would be better using larger images. The selected DF is 4, so the input images will have a size of 64x48.
6.1. LOCAL FEATURE OPTIMIZATION

CHAPTER 6. AGE ESTIMATION

6.1.2 Gradient

Downsampling factor

In the Gradient, the downsampling factor (DF) does determine the number of output features, thus it is important to optimize it. As we explained in previous sections, the number of features generated by the gradient is twice the amount of input image pixels. The Figure 6.5 shows the behaviour of different systems varying the DF, for various regressors.

![Figure 6.5: AUC of REDF in Gradient in function of the Downsampling Factor (DF), in test.](image)

Observe that while the DF increases, the AUC in REDF decreases, thus the error increases. However there is not a big difference in AUC value between DF 1 and 12, but the number of features is reduced from 98,304 to 704 ($\left\lceil \frac{256}{12} \right\rceil \cdot \left\lceil \frac{192}{12} \right\rceil \cdot 2$). Therefore, we choose 12 as a DF for the gradient.
6.1.3 Hessian

Downsampling factor

The downsampling factor (DF) for Hessian is similar than in Gradient. It is important because the DF fixes the number of output features, and using fewer features, the regressor would be easier and faster to implement and to use. The number of output features in gradient is twice the amount of input image pixels; however in the hessian is three times, so using the same DF the hessian has 50% more features than gradient. The conduct of DF in Hessian is shown in Figure 6.6.

![Figure 6.6: AUC of REDF in Hessian in function of the Downsampling Factor (DF), in test.](image)

The tendency of the AUC is to decrease when DF increases. But it starts to decrease considerably from DF 16. Consequently, we select 16 as a Hessian DF, so the output features are $576 \left( \left\lceil \frac{256}{12} \right\rceil \cdot \left\lceil \frac{192}{12} \right\rceil \cdot 3 \right)$.

6.1.4 Histogram of Oriented Gradients

To optimize the HOG parameters, we followed the same method that we have done in gender. We start assuming that the best parameters are those obtained in gender, i.e. HOG with 9 bins (B), 10 windows (W) and without downsampling factor (DF). Then we changed B with a fixed W and DF, so we have the optimal value B for age regression. The next step consists in use the obtained B and optimize W with a fixed DF, thus we obtain the optimal W for age. Finally, using the previous B and W, we studied the behaviour of the DF.

Number of bins

The number of bins (B) is one of the parameters that determine the number of HOG features, so it is important to optimize it, and use as less as possible in order to reduce the features. As mentioned above, we fixed W and DF to 10 and 1 respectively. Figure 6.7 shows the behaviour of B.

Observe that the AUC increases with B. Using few bins the AUC is small, thus the error is big. However, the difference between using 5 or 20 bins is practically indistinguishable. For
this reason, 5 bins are selected as an optimal B. Using 5 bins the angle range (from $-\pi$ to $\pi$) is divided in 5 parts, that is, each bin represents $\frac{2\pi}{5}$ radians (72 grades).

**Number of windows**

The number of windows (W) is the other parameter, together with B, that determines the number of output features. W represents the number of windows for each row and column that the image is split in. In order to do an easier optimization, we assume that the image splitting is done with the same number of windows for row and column, but it could be possible that these values were not the same. Because the non-quadrangular image, the cells are rectangular with a $\frac{3}{4}$ aspect ratio.
Increases while \( W \) increases too. The higher value of AUC is obtained using 14 windows, and then the AUC is practically constant. This is the reason why we select 14 as a number of windows. Note that knowing \( B=5 \) and \( W=14 \), the number of output features is also known: 980 \((B \cdot Wdot2)\).

**Downsampling factor**

As it happens in pixel difference, the downsampling factor does not determine the number of output features. However, it is good to know the conduct of it, because having the same error using different image sizes, our estimation can be done using smaller images and the system will be more robust (because it would not be necessary to have facial images of 256x192 for example).

The results of the study of the DF behaviour can be seen in Figure 6.9. It is done with \( B=5 \) and \( W=14 \), obtained in the previous sections.

![Figure 6.9: AUC of REDF in HOG \([B=5, W=14]\) in function of the DF, in test.](image)

Observe that the AUC decreases while the downsampling factor increases. However, from 1 to 5 the AUC is almost constant. That is because, although the downsampling implies the loss of some frequencies (high frequencies), the important image contours and transitions remain in the downsampled image, and the histogram is quite similar. The DF is 2, because is the higher value and it do no determines in the number of features.

**6.2 Feature optimization**

Once each feature is optimized, the next step is a global optimization in order to know how important each feature is, and which do we have to use to implement the regressor. The implemented method is exactly the same that in gender, with the mentioned evaluation method modification (in gender we evaluated the estimation with the misclassification error, and the best classifier was the one that gave the lowest error, but in age estimation the evaluation is done with the AUC, thus the higher value indicates which the best estimation
6.2. FEATURE OPTIMIZATION  

is). Observe in the Table 6.2 the type of features with the selected parameters. There are nine different features.

<table>
<thead>
<tr>
<th>Type</th>
<th>Feature name (Abbr.)</th>
<th>Parameters</th>
<th>Number of features</th>
</tr>
</thead>
<tbody>
<tr>
<td>Low level</td>
<td>Pixel difference (PD)</td>
<td>Num diff.=1,000</td>
<td>1,000</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DF=4</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Gradient (Gr)</td>
<td>DF = 12</td>
<td>704</td>
</tr>
<tr>
<td></td>
<td>Hessian (He)</td>
<td>DF = 16</td>
<td>576</td>
</tr>
<tr>
<td></td>
<td>HOG (Ho)</td>
<td>Num Bins = 5</td>
<td>980</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Windows = 14</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>DF=2</td>
<td></td>
</tr>
<tr>
<td>High level</td>
<td>Wrinkles (Wr)</td>
<td>-</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>Moustache (Mo)</td>
<td>-</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>Beard (Be)</td>
<td>-</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>Hair (Ha)</td>
<td>-</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>Gender (Ge)</td>
<td>-</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 6.2: Summary of the available features in age and their parameters.

Firstly the estimation is done just using the features one by one, and the best one is selected (the one that gives higher AUC of REDF). The next step is trying to know which feature added to the previously selected improves the most the estimation. Thus, the best two features are known. This step is repeated until the AUC of REDF does not increase, thus the estimation does not improve. The Figure 6.10 shows the results of the age estimation using one feature.

![AUC of REDF using one type of feature](image)

Figure 6.10: AUC of REDF using one type of feature to estimate the age, in test.

The feature that gives the highest AUC is HOG for all the regressors. The higher AUC is obtained with a single Tree, which has an AUC of 9.55, then the 10-Random Forest with an area of 9.53 and finally the Fern with 9.42.

The next step consists in trying to know which feature helps the most the HOG to do a better estimation. Hence, two features are used. In the Figure 11 is shown the difference of AUC between HOG and HOG with another feature added.
Observe that some features do not improve the estimation. Otherwise, using HOG the difference is 0 because the features are the same. However, there are some features that improve the age estimation. These features are different for each regressor: the Wrinkles are the ones that added to HOG improves the most the performance using a Tree; on the other hand the Hair is the one that increases the most the AUC when Random Forest or Fern are used.

The selection method is repeated using three features, the two selected before and a new one. In the Figure 6.12 the improvement of adding a new feature to the selected in the last step for each regressor is plotted. Then, the difference is calculated and plotted.

Note that any feature makes the AUC increase. We should obtain, at least, the same AUC as using the two selected features because there is exactly the same information (and other extra information added). However, this AUC reduction may happen because the k-fold (the sub-training-set is not the same for each k-fold, thus the regressor change and, consequently, the AUC changes). Despite the k-fold variance, the estimation does not improve, therefore we
6.3 Preprocessing optimization

Once we have the local features optimized and the selection of which one will be used, the next step is to study the image preprocess. In the Figure 6.13 it is shown the preprocessing diagram. As we said at the beginning of this chapter, we assume that the optimal crop and resize parameters for age estimation are the same than the obtained in gender. Therefore, only the importance of the histogram equalization is studied.

In order to know how the histogram equalization improves the estimation we use the selected parameters for each feature and a feature re-extraction without histogram equalization is done. The Table 4 shows the error of do not equalize the image histogram before the feature extraction. Note that the difference is practically 0, so we do not need the histogram equalization because is a useless step. As we said in gender, we suppose that the histogram equalization is not needed because we are using basically HOG, and the HOG of an image and the HOG of the same image with the equalized histogram is very similar because the contours do not change and HOG is invariant to illumination changes.

<table>
<thead>
<tr>
<th>Number of used features</th>
<th>Tree</th>
<th>Random Forest</th>
<th>Fern</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 feature</td>
<td>9.519 (Ho)</td>
<td>9.5271 (Ho)</td>
<td>9.4158 (Ho)</td>
</tr>
<tr>
<td>2 features</td>
<td>9.5712 (Wr)</td>
<td>9.5470 (Ha)</td>
<td>9.4737 (Ha)</td>
</tr>
<tr>
<td>3 features</td>
<td>STOP</td>
<td>STOP</td>
<td>STOP</td>
</tr>
</tbody>
</table>

Table 6.3: Selection of the best features for each classifier using the maximum AUC of REDF criterion.

<table>
<thead>
<tr>
<th>Tree: HOG+WR</th>
<th>Random Forest: HOG+HA</th>
<th>Fern: HOG+HA</th>
</tr>
</thead>
<tbody>
<tr>
<td>HIST. EQUALIZATION</td>
<td>9.5712</td>
<td>9.5470</td>
</tr>
<tr>
<td>NON HIST. EQUALIZATION</td>
<td>9.5721</td>
<td>9.5237</td>
</tr>
<tr>
<td>DIFFERENCE</td>
<td>-0.0009</td>
<td>+0.0233</td>
</tr>
</tbody>
</table>

Table 6.4: Comparison between estimate with histogram equalization or without it.
6.4 Regressor optimization

Once the local feature parameters optimization, features selection and pre-processing study are done, the regressor has to be optimized. As said in the beginning of the chapter, for make the optimization faster we use Random Forest with 10 trees; however more trees can be used in order to have a better estimation. Therefore an experiment is purposed to know the influence of using more trees in Random Forest. The features and the pre-processing are the optimal obtained in previous sections (HOG-B5-W14-DF2 and Hair extracted from an image without the equalized histogram). The Figure 6.14 shows the AUC of REDF training the Random Tree with the same features but with different number of trees.

![Figure 6.14: AUC of REDF using different number of trees in Random Forest.](image)

We expected that using more trees the regressor would correctly predict more data. However, it is seen in the Figure 6.14 that since using 10 trees the AUC is practically constant, using more trees the regression does not improve. Consequently we still use 10 trees for Random Forest.

6.5 Results

Once all the estimation chain (pre-processing, features, features selection and regressors) is optimized, the global results are obtained. We have used AUC of REDF to compare between to estimators because is an easy method where the comparison is done with real values. However, the AUC is an approximation to guess which estimator is better. It is possible to have a curve nearest to the (0, 1) point than other but with a lower AUC. Another possible case is an estimator with very good results but some high outliers. Then, the AUC should be higher, but maybe we prefer this than another without outliers but with more relative error in all the samples. For these reasons, we analyse the obtained results with another method, which is more visual to compare between two estimators.\$\$

The proposed method is a scatter plot or scattergraph where the values of two variables are displayed: the age and the predicted age. A collection of points is plotted, the truly age determines the position in the horizontal axis and the predicted age determines the vertical position. The optimal cloud of points is which all the points are over the line x=y, i.e. when
the predicted age is exactly the same than the true age.

The scatter plot displayed in the Figure 6.15 is computed in order to compare between the estimators, thus the train and test is plotted for each estimator.

![Scatter plot of the estimations](image)

Figure 6.15: Scatter plot of the estimations

Observe in the plots displayed in the Figure 6.15 that the Tree is overfitted in train, thus in test the results have more error. However the train error obtained using Random Forest and Fern is higher, but then, in test, the results are better. Note that during all this chapter the estimator with better results (higher AUC) was Tree, however in Figure 6.15 can be seen that Random Forest has a better performance.

We can see in the scatter plots a hole in the true axis between 6 and 17, result of the lack of data in the database in this range of ages. It also can be seen that the people who
are about twenty have a predicted age higher than twenty. Similarly happens in the other extreme, with the oldest people where the predicted age is always lower than what it should be. This behaviour is because the regression; as it is explained in the section 2.4.3 Random Forest, when the regression is trained, the predicted age in one leaf is obtained with the mean of all the ages of the samples that are in the leaf. Therefore, when just a sample aged higher than 17 is brought to a leaf that should be 17, the predicted age in that node increases, and the 17 value will never be an output of the system. Exactly the same happens with the older people.

Note that the prediction of the child people is not very good. We suppose that is because the images of the kids came from another database, where the images are took in the nature, thus the statistics of the database is very different than the Morph dataset where the frontal images are took expressly for the database creation. We decide to remove the Gallagher dataset and train the estimator again with only the Morph database.

As can be seen in the Figure 6.17 the prediction is not bad, however we can observe a bias problem, because the cluster should have the same orientation as the optimal line $x=y$ (black line in Figure 6.17), but it has not. To solve this problem we try to estimate the orientation of the points cluster and correct it in order to have the cluster over the optimal line. Obviously this prediction has to be done using the training data, thus the first step is compute the line that fits the cluster using the `polyfit()` Matlab function with an order 1 polynomial.

The Figure 6.17 also has the train fitting line (plotted in green), and the test fitting line (in yellow). Note that the two lines are quite similar, thus is a good approximation.

![Figure 6.16: Scatter plot of the prediction using 10-Random Forest with the MORPH subset](image)

Figure 6.16: Scatter plot of the prediction using 10-Random Forest with the MORPH subset

The next step is to correct the points cluster with the obtained line. That correction makes the prediction of young people younger and the old people older. This correction is displayed in the Figure 17. Observe that in train, when the correction is done, the optimal line and the fitting line of the cluster are exactly the same. In addition, in the corrected test, the correction is quite good and its fitting line is approximately the same as the optimal line (because the fitting lines of the train and test are also similar).
6.5. RESULTS

CHAPTER 6. AGE ESTIMATION

Figure 6.17: Scatter plot of the correction of the prediction using 10-Random Forest with the MORPH subset

The correction is done with the equation

\[ \text{new}_{\text{age}} = 1.1915 \cdot \text{pred}_{\text{age}} - 7.3828 \]

Note that the tendency of this equation is to make younger the people younger than 38, and older the range older than 38. Figure 6.18 shows the improvement of the correction. The discontinuous line represents the curve with the bias correction, and note that it is higher than the continuous one, thus the prediction is better.

With the REDF curves it is easier to understand how good the prediction is. Observe that the 60% of the estimations have less than 0.2 of relative error (that is, for example, say to a 20-year-old person that he looks like 16 or 24; or mistake of 10 years to a person who is 50) and the 75% of the estimations have less than 0.3 of relative error (that is, for example, commit a mistake of 10 years someone who is 30).

Figure 6.18: REDF of the prediction using 10-Random Forest (corrected and non-corrected) with MORPH subset.

It is also interesting to know which the behaviour of the age estimation in function of some other variables such as gender, race or range of age is. It can be seen in the figure 6.19.
6.5. RESULTS

CHAPTER 6. AGE ESTIMATION

(a) REDF for different the gender

(b) REDF for different races

(c) REDF for different range of ages

Figure 6.19: Behaviour of the estimation in function of gender, race and age

Observe that neither the gender (6.19a) nor the race (6.19b) are relevant to the age estimation because the obtained curves are almost overlapped. The behaviour in range of ages (6.19c) is not the same: the relative error in the youngest and oldest people is higher. As mentioned, that happens because the regression, where some problems in the data extremes may happen.

To know how good the age prediction is, we would like to compare it with the prediction done by the humans. That is the reason why we made a quiz and we sent it to different people. In the quiz we asked to, using some facial images from the Morph Database, guess the age of the subjects. Doing so, a pretty good approximation to how humans would estimate
is obtained. Finally, in Figure 6.20 the comparison between the human estimation and the estimation done using the developed system is plotted. Observe that the two curves of the REDF are very close; therefore the behaviour of the estimation done with the proposed system is quite similar to the human estimation. It is difficult to do it better than what people do, because some people looks older or younger than what they are, thus we consider that the obtained results are very good.

Figure 6.20: Comparison between the REDF of the estimation done with our system and the human estimation.
Chapter 7

Conclusions

Gender and age estimation can be achieved from facial images, and we developed methods to do it. In order to make the estimation, we propose a chain that follows these steps: preprocessing, feature extraction and estimation (regression or classification). Then, in order to know how good the estimation is, we propose an evaluation and then, the best parameters and features are obtained for both, gender and age estimation.

For the gender estimation, HOG with 9 bins and 10 windows is the best feature found, and the best classifier is SVM with a linear kernel. The misclassification error in the Feret database is 2.61%. This performance is better than what we expected, having a quite low error. The gender estimation done by humans is close to 99% (misclassification of 1%). Thus, is difficult to have a better performance, but the obtained results are near to the human performance.

For the age estimation, the selected features are HOG with 5 bins and 14 windows and Hair. The best regressor is Random Forest with 10 trees. The final results of the system applied over the Morph dataset show that 60% of the estimated samples have less than 0.2 of relative error, and the 75% have less than 0.3. In order to know how good the results are, we run a test where humans estimate the age of some subjects from the Morph dataset. Then, we compare the performance of our system with human results, and we notice that the behaviour is almost the same. It is difficult for the system to estimate better than humans do, so we can conclude that our results are good. In both gender and age estimation, the image equalization is not needed because the principal feature used is HOG and it is invariant to illumination changes.

In both gender and age estimation the low level features, in particular HOG, are more useful than high level features (those that are more useful for the humans to do the estimation, such as Wrinkles).

It is important to note that the implemented system tries to predict the real gender and age. However, we work with facial images and these just give us information about the look of the person. Therefore is difficult to estimate which the real age of the person is, instead of how old looks him.

To adapt the system and make it useful for a real application, the only requirement is to train it with a dataset that really represents the real input images, because as we could see in the development, the results change depending on the dataset used. We also recommend having a training dataset with a bigger age range than the needed, because we know that the estimation in the edge ages is not as good as in the other age ranges.
Some further work based on that project could be done. The colour information of the images can be used, as well as other type of features such as eigenfaces. Also other estimators can be implemented in future projects, for example the adaptation of SVM to regression. Adapt the system for video inputs instead of still images, which may be closer to what real applications need, is another possible implementation. Since neither the age nor gender change significantly in what a video lasts, the prediction has not to be done at every single frame. The easiest way to do so would be taking just a single image from the video and proceed in the same way as done in this project, but since the video can have a lot of different images from the same subject, we can take advantage of that and try to take as much images from a single subject as possible, estimate the age and the gender for all of them, and return a value depending on the results from each image. Another interesting application would be implement the system for mobile devices, taking into account the camera quality, the computational complexity and the device memory. Doing so maybe the system performance may get worse, but the execution should be much easier, which can be useful in some cases.
Bibliography


