

University of Pisa

DEPARTMENT OF COMPUTER ENGINEERING

Master Degree in Computer Engineering



Design and prototyping of
a face recognition system on
smart camera networks

Supervisors

Prof. Giuseppe Amato

Prof. Fabrizio Falchi

Prof. Francesco Marcelloni

Candidate

Costantino Perciante

A.Y. 2014/2015

Abstract

The aim of this work is to design and develop a face recognition system running on smart camera networks. In many systems, these are passively used to send video to a recording server. The processing of the acquired data is mainly executed on remote and more powerful computers (or clusters of computers). In this thesis a distributed architecture was developed where computer vision algorithms are executed on smart cameras, which can exchange information to improve resource balance. A smart camera network has been defined specifying the roles that client nodes and a server have, and how nodes cooperate and communicate among them and with the server. Smart cameras, initially look for changes in the environment. When motion is detected, they perform face detection. Once a face is found, the camera itself processes it and tries to assess whom it belongs to, using a local cache of recognizers. This cache stores a portion of the whole information present on server side, and can be used to perform recognition tasks on the smart cameras. If a node is not able to identify a face it sends a query to the server. Finally, if the person's id can be determined, either by the server or the client itself, the occurrence of the correspondent recognizer is notified to the nearest nodes. Human faces that were not recognized, are stored on the remote server and can be manually annotated. Clustering algorithms have been tested in order to automatically group faces belonging to unknown people on server side and made the manual annotation easier. Extensive experiments have been performed on a freely available dataset to both assess the recognition performance and the benefits of using collaboration among cameras. Raspberry PI devices were used as camera network nodes. Various tests were performed in order to verify the efficiency of the face recognition approach on such devices.

Contents

Abstract	1
List of Figures	6
List of Tables	7
Abbreviations	8
1 Introduction	9
1.1 Contributions of this thesis	10
1.2 Thesis structure	11
2 Background and Related Work	13
2.1 Background	13
2.1.1 Face detection	13
2.1.2 Features for Face Recognition	20
2.1.3 Classifiers for Face Recognition	25
2.2 Related work	29
3 Camera network	32
3.1 Goals and assumptions	32
3.2 Client side software architecture	34
3.2.1 Stage 1: Motion detection	34
3.2.2 Stage 2: Face detection and face processing	36
3.2.3 Stage 3: Face recognition	42
3.2.4 Stage 4: Neighbors communication	43
3.3 Server side software architecture	44
4 Exploiting unrecognized faces	46
5 Experiments and results	51
5.1 Dataset of faces	51

5.2	Recognition performance	52
5.2.1	LBPH	53
5.2.2	PCA on LBPH feature	61
5.2.3	Cross-validation	65
5.3	Simulating a network of cameras	66
5.3.1	Alerting the whole network	68
5.3.2	Alerting the nearest neighbors	75
5.4	Exploiting unrecognized faces	83
6	Conclusions	85
6.1	Future work	86
	Bibliography	87

List of Figures

1.1	AmI and related research fields.	10
2.1	Basic Haar features used in the Viola & Jones framework for object detection. . .	14
2.2	Fast Haar feature evaluation using integral images	15
2.3	Initial haar features selected by AdaBoost	16
2.4	Cascade of weak and strong classifiers	17
2.5	LBP operator example	17
2.6	LBP texture primitive examples.	18
2.7	MB-LBP feature extraction example	19
2.8	MB-LBP feature samples	20
2.9	LBPH feature extraction	22
2.10	kNN example in which the green ball must be labelled whereas the labels of the blue and red objects are known.	28
3.1	Raspberry Pi B+ equipped with Camera Pi module.	33
3.2	Client software pipeline stages.	34
3.3	Frame differencing drawbacks: ghosting and foreground aperture, taken by [1] . .	36
3.4	Face detection executed on the Raspberry PI and applied on stream coming from Raspberry PI camera. MB-LBP and Haar trained detectors are compared while varying the frame size. The time needed by the former detector is always smaller.	38
3.5	Gamma and logarithmic correction examples	40
3.6	Histogram equalization example	41
3.7	Final processed frame: central eyes coordinates found (left), scaled and rotated face (center), elliptic mask applied (right)	42
3.8	Face classification process	42
3.9	Handshake for node communication	43
4.1	Manual annotation tool for unrecognized faces	46
5.1	Yale dataset samples	51

5.2	1NN accuracy and percentage of classified faces for different training set sizes (from top-left to bottom-right): 5, 15, 40 and 50 samples for class	54
5.3	1NN (Accuracy(%) vs Classified(%)) when the number of training set samples for each class increases	55
5.4	kNN accuracy and percentage of classified faces when $k = 3$ and the training set is enlarged	56
5.5	kNN with $k = 3$ (Accuracy(%) vs Classified(%)) when the training set size increases	57
5.6	Results of weighted kNN when $k_{opt} = 20$ and the training set size increases, considering confidence c_{wkNN_1}	59
5.7	weighted kNN for $k = 20$ (Accuracy(%) and Classified(%)) and the training set size grows	60
5.8	weighted kNN results for $k = 20$, considering c_{wkNN_2} confidence and different TS sizes	60
5.9	1NN combined with PCA feature reduction when varying the threshold t and the number of samples per class in the training set (15, 40, 50).	62
5.10	1NN combined with PCA (Accuracy(%) vs Classified(%)) when the number of training set samples per class increases	63
5.11	kNN algorithm when PCA is applied on LBPH feature (Accuracy(%) vs Classified(%)) and the training set size grows	63
5.12	weighted kNN combined with PCA feature reduction, varying confidence for $k = 10$ and training set size	64
5.13	weighted kNN combined with PCA (Accuracy(%) vs Classified(%)), for $k = 10$.	65
5.14	Cross validation for weighted kNN	66
5.15	Network simulation - alert messages sent to the whole network: average number of alert messages and classifiers sent by clients	71
5.16	Network simulation - alert messages sent to the whole network: average number of requests performed by clients.	72
5.17	Network simulation - alert messages sent to the whole network: average number of classifiers used, not used and received by nodes.	73
5.18	Network simulation - alert messages sent to the whole network: average number of classifications, succeeded classifications and correct classifications performed by clients.	74
5.19	Network simulation - alert messages sent to the whole network: number of classifications, succeeded classifications and correct classifications performed by the server.	75
5.20	Nearest neighbors of the red camera are reported in blue	75

5.21	Network simulation - alert messages sent to the nearest neighbors: average number of alert messages and classifiers sent.	77
5.22	Network simulation - alert messages sent to the nearest neighbors: average number of requests performed by clients.	78
5.23	Network simulation - alert messages sent to the nearest neighbors: average number of classifiers used, not used and received by nodes.	79
5.24	Network simulation - alert messages sent to the nearest neighbors: average number of classifications, succeeded classifications and correct classifications performed by clients	80
5.25	Network simulation - alert messages sent to the nearest neighbors: number of classifications, succeeded classifications and correct classifications performed by the server.	81
5.26	Network simulation - alert messages sent to nearest neighbors: cache of unknown people by the server	82
5.27	Clustering algorithms results	84
5.28	Example of wrong cluster	84

List of Tables

2.1	Eigenface example	24
2.2	Fisherface example	25
2.3	kNN example of confidence values for a fixed k when equation 2.27 is used.	28
3.1	Raspberry PI CPU usage while running face detection for different frame sizes.	34
3.2	Raspberry PI CPU usage and FPS while running the motion detection algorithm	37
3.3	Face detection executed on the Raspberry PI and applied on the video stream acquired by PI Camera module	39
3.4	Eyes detection time for LBP cascade classifier and Haar cascade classifiers	41
3.5	Services offered by clients and invoked by neighbors	44
3.6	Database entry description of a sample belonging to the training set	44
3.7	Services offered by the server	45
4.1	Already processed images sent by clients after local recognizer failure.	47
5.1	Time required for LBPH feature extraction and χ^2 distance evaluation	53
5.2	Accuracy for some k values while increasing TS and applying the weighted kNN algorithm	58

Abbreviations

TS	T raining S et
PCA	P rincipal C omponent A nalisis
PC	P rincipal C omponent(s)
LBP	L ocal B inary P attern
LBPH	L ocal B inary P attern H istogram
LDA	L inear D iscriminant A nalisis

Chapter 1

Introduction

Thanks to the miniaturization of electronics, computing and sensing devices have become smaller and faster and can be placed in the environment in which humans live, in order to help them in their daily tasks and activities. Such devices, that are going to completely disappear in the surrounding, share information over a (mainly wireless) network, and are able to understand events and take decisions in real-time, with very limited user interaction.

This field of research is known as Ambient Intelligence (AmI), that can be defined as [2]

“A digital environment that supports people in their daily lives by assisting them in a sensible way.”

Its paradigm is build on the following concepts: pervasive computing, context awareness and human-centric computer interaction. It is also related to other areas of research, such as Sensor Networks, Pervasive Computing, Artificial Intelligence (look at Figure 1.1).

In the Sensor Networks field, a lot of effort has been recently spent on the use of Smart Camera Networks. Smaller and cheaper smart nodes, equipped with cameras, can be installed in a wide variety of locations for different purposes. In fact, the information collected by cameras can be useful in different vision systems and applications, such as surveillance of indoor or outdoor spaces, healthcare, biometric recognition, detection of position of objects, environmental monitoring and so on.

In this work, we present a distributed system that uses smart camera networks. Despite the fact that these cheap and small devices can be placed almost everywhere, using them is challenging for real-time applications, since they have limited resources. The implemented software enables client nodes to detect faces and recognize people. Nodes cooperate and exchange information

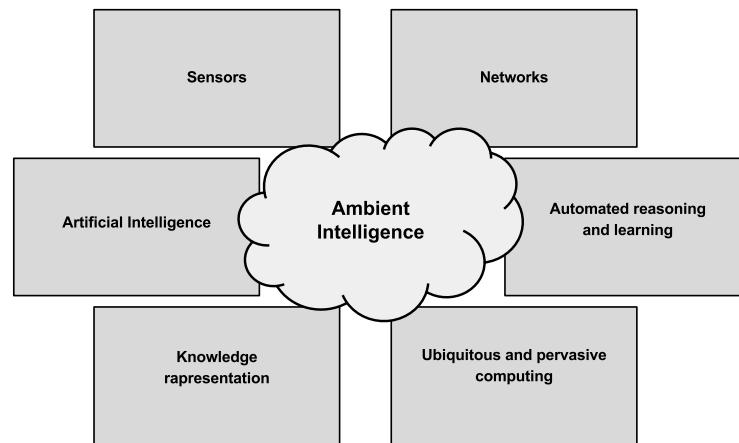


Figure 1.1
AmI and related research fields.

in order to improve both the scalability and the effectiveness of the recognition system.

The Raspberry PI platform equipped with the Pi Camera has been used for testing purposes. However, the system has been designed in order to use other smart cameras in which a Java virtual machine can run.

In the rest of this chapter, the main contributions of this thesis are reported. Finally, the structure of the entire document is briefly shown and discussed.

1.1 Contributions of this thesis

This thesis has three main contributions: 1) definition of a collaboration strategy among smart cameras to execute recognition tasks 2) definition of two clustering algorithms to group together unrecognized faces and 3) extensive experimentations of various face recognition strategies and camera cooperation strategies.

Strategies for camera collaboration

A camera network is a collection of smart nodes equipped with cameras able to communicate through a wireless/wired network. In section 2.2 of chapter 2, some examples of such systems are reported. The aim of our system, is to use the increasing power of those cameras to execute computer vision algorithms in order to better balance resources and scale-up with respect to a standard system in which cameras are just used to send stream to a recording server. The relationship between the unique identifier of a person that the system should be able to recognize and the information that let the system to recognize him/her is maintained on the central server. However, since nowadays smart nodes have enough CPU power and memory, the key idea is

to use them to directly execute the recognition algorithm and reduce requests to the server. A communication strategy has been defined in order to let nodes exchange useful information for face classification. Smart cameras store this information in a cache, whose content changes when nodes communicate. Thus, requests to the remote server are done only when a node is not able to recognize a detected face using the information in the local cache. The smart camera network is described in chapter 3.

Exploiting unrecognized faces

When a camera is not able to recognize a face, it makes a request to the server. If the server is also not able to recognize the face, the picture of the unrecognized face is stored at server side. A human operator can inspect and label them, but this task can be optimized if we cluster together faces that might belong to the same person. In this way, the human operator has to label entire clusters and/or correct errors occurred during the clustering process. Two different clustering algorithms for such task have been implemented and tested. Their description is reported in chapter 4 whereas the result of the tests is reported in chapter 5.

Recognition performance and network simulation tests

We have implemented and tested three algorithms, namely the 1NN classifier, the kNN and the weighted kNN classifiers. They have been extensively tested against a public available dataset. Different strategies for nodes communication have been simulated in order to understand how they, the size of the nodes' caches and the classifier algorithm used on nodes can affect the scalability of whole the system. We have compared these strategies a) each camera queries the server for face classification b) cameras perform face recognition and alert the whole network when recognize someone and, c) cameras perform face recognition but the alert message is sent to their nearest neighbors only. For all the approaches, we evaluated and compared the number of requests received by the server. For the last two strategies, the average number of classifiers and alert messages sent by clients, the number of classifiers sent by the server and the portion of classifiers that are used by nodes when varying the cache size have been evaluated.

1.2 Thesis structure

In Chapter 2 related work in which smart camera networks applications enhanced by computer vision algorithms are used are reported. The state of the art techniques for both face detection and facial feature extraction tasks are discussed. The classifiers used for performing face recognition relying on the previously extracted features are described. Finally, the implemented choices are described and motivated taking into account the reduced computing capabilities of smart nodes.

In Chapter 3 the main goals and assumptions of the network of cameras that we define are described. Nodes and server's activities and tasks are reported together the way they communicate among them. The entire pipelined process at node side, that starts with image acquisition and ends with face labelling is shown. Server's tasks and role are reported too.

In Chapter 4 two clustering algorithms, based on facial features, for collecting remote faces on server side are described. This task can help a human operator in the labelling process and automatically refine the knowledge the system has about an already known person or suggest the operator that a new person can be recognized.

In Chapter 5 are reported the experiments done together with the obtained results. In this chapter, the implemented classifiers are tested and the ability of the system to collect unrecognized faces has been evaluated. Furthermore, different strategies for nodes communication have been simulated and the related results are reported too.

In Chapter 6 the reached goals of the system are described and some of the possible enhancements and future work are discussed.

Chapter 2

Background and Related Work

Person identification in images, or frames coming by a video source, consists of a two steps pipeline: *face detection* and *face recognition*. In the following, different approaches for both tasks are reported and described. In the last part of this chapter are reported related work in which camera networks are used.

2.1 Background

2.1.1 Face detection

Face detection is a computer technology that allows a computer to identify human faces in digital images and video. Although humans can easily detect faces, this task is not trivial for a computer, due to the fact that human face is a dynamic object, with high degree of variability in its appearance. Different illumination conditions, facial poses and expressions, occlusion, rotation can affect a face detection algorithm. The solution to the problem involves segmentation, extraction, and verification of faces and possibly facial features from an uncontrolled background. In the literature, a lot of algorithms have been proposed to perform detection operation in real-time. A survey of these techniques is reported in [3].

In the following subsections, the **Viola and Jones framework** [4], the **LBP operator** [5] and one of its variant, namely the **Multi-Block LBP** [6], are briefly described. The original version of this famous framework uses Haar features. However, the same framework can be used to train classifiers based on the LBP operator or the Multi-Block LBP. In order to perform the training task of the classifier, a lot of positive samples (i.e., images that contain object the detector should be able to find in future images) and negative samples (i.e., images that *don't* contain object the detector should be able to find) are required. This task could take weeks for the Haar/LBP cascade classifier or days for the MB-LBP cascade classifier, but this heavily

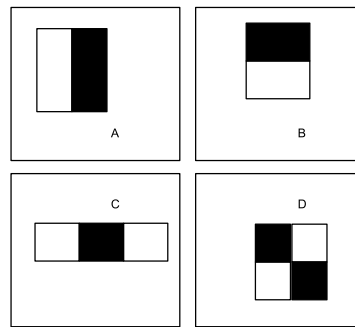


Figure 2.1

Basic Haar features used in the Viola & Jones framework for object detection.

depends upon the size of training set and the performance in term of alarm rate or true detection rate that the final classifier should reach. The OpenCV library offers pre-trained classifiers for faces, eyes, eyes with glasses, mouth, etc. Most of them are based on Haar features, however the library contains some classifiers for face detection that use the MB-LBP operator too. This feature makes the classifier faster and this property is exploited in the smart cameras of our system. A comparison between the Haar and MB-LBP cascade classifier on the Raspberry PI is shown in chapter 3.

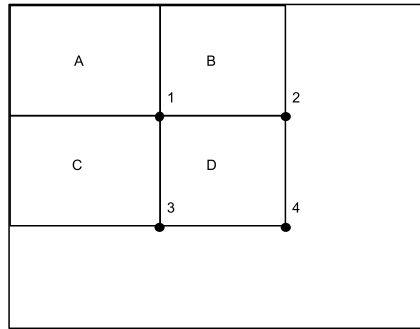
Viola-Jones object detection framework

Although it can be used to train detectors for a wide range of objects, it was mainly motivated by the face detection problem. It was proposed by Paul Viola and Michael Jones [4] and is based on a combination of

- a image representation called **integral image** which allows a very fast evaluation of the Haar features;
- a learning algorithm based on AdaBoost [7] in order to rapidly and efficiently classify faces with a reduced number of selected Haar features;
- a cascade of classifiers that allow to immediately discard non-face regions (such as background regions).

The result is a robust algorithm, able to reach very low false positives [4].

Haar features Three kinds of Haar features are used to scan the image and are reported in Figure 2.1. The value of the *two-rectangle feature*, is the difference between the sum of the pixels within two rectangular regions. The regions have equal size and shape and are vertically or horizontally adjacent (cases (a) and (b)). The value of the *three-rectangle feature* is given by the subtraction between (the sum of) the outside rectangles and the central one (case (c)).

**Figure 2.2**

Fast Haar feature evaluation using integral images. The value of the integral image at location 1 is the sum of pixels in A. At location 2 is $A + B$, at location 3 is $A + C$ and at location 4 is $A + B + C + D$. The sum of pixels in D can be quickly evaluated as $4 + 1 - (2 + 3)$.

Finally, for the *four-rectangle feature* the result is equal to the difference of diagonal pairs of rectangles (case (d)). Given that the minimum area in which the detector will look for objects is a sub-window of 24×24 pixels, a mechanism is necessary to reduce the number of features that can be applied in this region. This number is much higher than the number of pixels the window contains.

Integral images In order to quickly evaluate the values of those features, integral images have been introduced. It can be seen as an intermediate representation of the gray scale image I . The integral image I' at position (x, y) is defined as

$$I'(x, y) = \sum_{x' \leq x, y' \leq y} I(x', y'). \quad (2.1)$$

The sum of pixels in a rectangle of the original image I can be evaluated in constant time with four operations through I' (see Figure 2.2).

Learning a classifier Given a feature set and a training set of positive and negative images, a classification function must be learned. A variation of the AdaBoost is used to both select a small set of features and train the classifier (sometimes called weak classifier). On a 24×24 patch image, over than 180.000 rectangle features could be potentially applied and evaluated [4]. Unfortunately, the same operations need to be performed during detection and this is not feasible for a real-time system. Hence, the main idea is to search for a single feature that best separates positive and negative samples. So, for each feature, the optimal threshold classification function is found such that the number of misclassified examples is minimum. A weak classifier

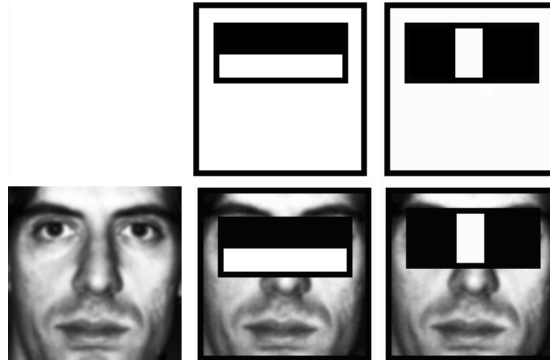


Figure 2.3

Initial haar features selected by AdaBoost for the classifiers focus on faces' properties such as darker eyes region with respect the region below the eyes and nose region.

$h_j(x)$ consists of a feature f_j , a threshold θ_j and a parity value p_j indicating the direction of the inequality sign:

$$h_j(x) = \begin{cases} 1, & p_j \cdot f_j(x) < p_j \cdot \theta_j \\ 0, & \text{otherwise} \end{cases} \quad (2.2)$$

The 24x24 pixels sub-window has been denoted with x . The initial feature selected by AdaBoost focuses on the property that the region of the eyes is darker than the one below the eyes. Whereas, the second selected feature relies on the fact that the eyes are darker than the nose region (see Figure 2.3). If more rounds are performed, the AdaBoost algorithm selects one feature among the 180.000 ones at each iteration. In this way, a strong classifiers $H(x)$ can be build using a combinations of weak classifiers

$$H(x) = \sum_j w_j \cdot h_j(x) \quad (2.3)$$

The features selection process is based on the Winnow exponential perceptron learning rule [8]. Classifiers that use 2,3,...,n features are build and they will be used in the last stages of the cascade of classifiers.

Cascade of classifiers A single feature classifier cannot perform classification with low error. Achieving low error rate requires more features for patch, and this means a lot of time for a single 24x24 rectangular area. A combination (a *cascade*) of week classifiers and slightly more complex classifiers is used to achieve lower error rates (See Figure 2.4) without increasing computation time. The key idea is to use simpler classifiers in the first stage of the cascade (they use

less features) which can discard immediately non promising regions of the image while spending more computation time in the last stage of the cascade (in which the classifiers use about 50 features). Non face regions are discarded immediately, and very low error rate could be achieved.

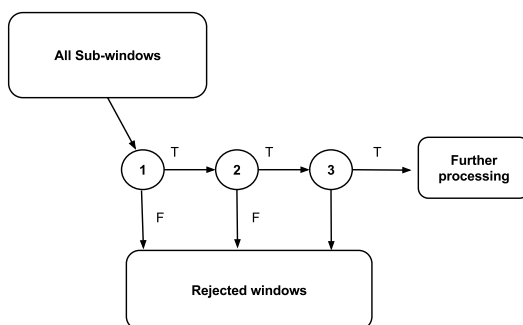


Figure 2.4

Cascade of classifiers: each region is scanned to assess if it contains a face. Regions that do not pass the first stage are immediately discarded whereas the ones that successfully pass this stage are re-scanned through a window in which more features are used. In the end, only the regions that successfully pass the entire cascade are retained.

The same authors improved their framework in order to take in consideration profile faces and face rotation in [9].

LBP operator

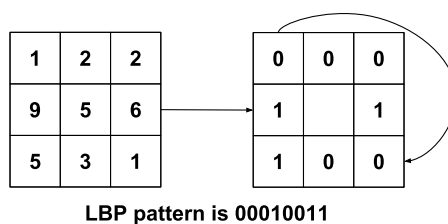


Figure 2.5

Basic LBP operator: the intensity values of the neighborhood are thresholded through the intensity of the central pixel in order to generate a binary pattern encoding the kind of area.

Haar features can be quickly evaluated through integral images. However, these operations can be still not feasible on mobile devices. In fact, the learnt cascade classifier still uses a lot of features in order to determine if a patch contains a face. To speed up computation, the use of the Local Binary Pattern [5] operator and its variant, the Multi-Block LBP was proposed. This type of feature performs very well in various applications, including texture classification and segmentation, image retrieval and surface inspection. Moreover, given the structure of the

feature, it can be used for both face detection and recognition (see subsection 2.1.2). The base version of the operator considers each pixel of the gray scale image and generates a binary pattern as shown in Figure 2.5. Every pixel and its neighborhoods are considered. Neighbors' intensity values are thresholded using the central pixel intensity one:

$$s(x) = \begin{cases} 1, & x \geq 0 \\ 0, & x < 0 \end{cases} \quad (2.4)$$

$$LBP(x, y) = \sum_{p=0}^{(p-1)} 2^p \cdot s(i_p - i_c), \forall (x, y) \in I \quad (2.5)$$

The total number of patterns is 2^8 , considering 8 neighbors. The operator was later extended to use different radii and number of neighbors. Let denote the generic operator considering P neighbors and R as radius, with $LBP_{P,R}$. Neighbors' coordinates can be found using the following formulas

$$x = x_c + R \cdot \cos\left(\frac{2\pi p}{P}\right) \quad (2.6)$$

$$y = y_c + R \cdot \sin\left(\frac{2\pi p}{P}\right) \quad (2.7)$$

with $p = 0, 1, \dots, P - 1$. If the neighbouring coordinate does not correspond to integer values, then bilinear interpolation is used for estimation of pixel value. A rotation-invariant LBP^{riu} has been also proposed. Rotation invariance is achieved by circularly rotating each bit pattern to the minimum value. For instance, the bit sequences 1000011, 1110000 and 0011100 arise from different rotations of the same local pattern and they all correspond to the normalized sequence 0000111.

Each of this generated pattern can be seen as micro-texton, encoding different types of areas such as edges, flat areas, corners and so on (Figure 2.6).

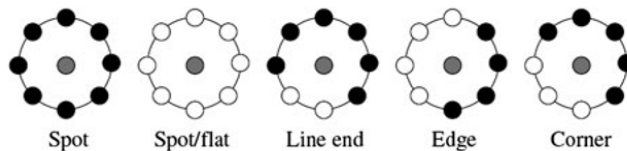


Figure 2.6
LBP texture primitive examples.

The most important properties of LBP features are their tolerance against monotonic illumina-

tion changes and their computational simplicity. This operator can be used for face recognition purposes encoding the whole face.

For face detection, the same algorithm proposed by Viola and Jones can be adopted. The main variation is the use of the LBP texture primitives instead of the Haar ones. The algorithm starts building the weak classifiers. In this case a single LBP feature is used to determine if a region x contains or not a face. Then, complex classifiers are built using a variation of the AdaBoost algorithm, namely the Gentle AdaBoost. The whole classifier is enhanced with a cascade of weak and strong classifiers.

Multi-Scale Block Local Binary Pattern

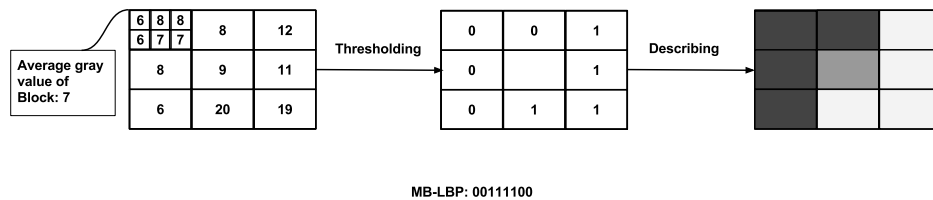


Figure 2.7

MB-LBP feature extraction example: the average intensity for each block is evaluated and then a binary pattern is obtained by thresholding neighborhood's values through the central block average intensity one.

The OpenCV library uses a variant of the LBP features, known as Multi-Scale Block LBP [6] for detection purposes. In fact, the LBP operator alone isn't able to make the face detection classifier faster, since Haar features can be evaluated fast as well through integral images. The basic idea of MB-LBP is to encode rectangular regions by local binary pattern operator. The MB-LBP features can also be calculated rapidly through integral image, while these features capture more information about the image structure than Haar-like. The real great advantage of MB-LBP is that the number of exhaustive set of MB-LBP features is much smaller than Haar-like features (about 1/20 of Haar-like feature for a sub-window of size $20 \cdot 20$). This fact makes both the classifier training and the face detection phase extremely fast with respect the Viola & Jones Haar features.

Whereas the LBP operator encodes information of a circular neighborhood of pixels, the MB-LBP encodes rectangle using a similar approach. Considering a neighborhood of rectangles of fixed weight and height, the central rectangle's average intensity is compared with those of its neighbors rectangles $\{r_0, r_1, \dots, r_7\}$, outputting a binary sequence

$$MB-LBP(x,y) = \sum_{p=0}^{(p-1)} 2^p \cdot s(r_p - r_c). \quad (2.8)$$

The function $s(x)$ has been defined in Eq 2.4. Figure 2.7 shows a simple example of the extraction of the MB-LBP binary patterns, considering rectangles of size 2×3 .

Totally, the number of binary patterns is 256, some of them are reported in Figure 2.8. Of course, this number is much higher if various scales are considered, anyway it is still lower than the corresponding number of Haar features for the same window patch.

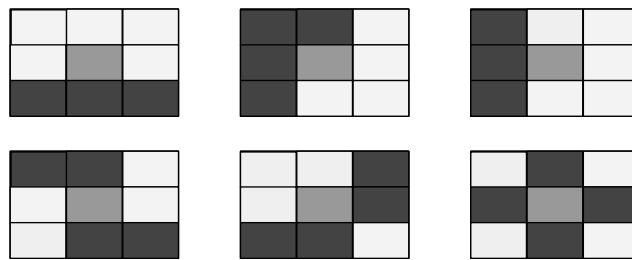


Figure 2.8
MB-LBP feature samples

2.1.2 Features for Face Recognition

Face recognition is still today an hot research topic. Several algorithms have been proposed to enable systems to recognize people, but this task still remains challenging. It is a classification problem in which a new detected face must be matched against an available dataset of already labelled faces. In order to efficiently represent those faces, algorithms are used to extract facial features. Among these, the Eigenface method based on PCA [10], the Fisherface method based on LDA [11] and Elastic Bunch Graph Matching (EBGM) that approximates a face through a deformable graph [12] have obtained success and are largely used in face recognition applications. More complex and sophisticated features based on Artificial Intelligence and its new research fields, such as Deep Learning technology [13], have been proposed too. These algorithms achieve great accuracy but are very computational demanding. Thus, in order to meet the requirements of the real-time applications and of the embedded computing boards, simple algorithms are required for fast feature extraction. Among these, the LBPH [14] has been investigated and is used in the system since it better fits the requirement of a dynamic system in which classifiers can be added with respect to the Fisherfaces model. Moreover, in [14] the authors show that LBPH outperforms the Eigenfaces approach.

These features are tools that can be used in face recognition systems since their aim is to

represent and capture the main features of a face and allow to the distinguish faces of different people. Since new facial features are continuously proposed in the literature, the system has been written in such a way that this component can be easily updated/modified. How these are used to accomplish the face recognition task is reported in chapter 5. In the following, we describe some of the existing ones.

Local binary pattern histogram

The LBPH feature [14] is based on the LBP operator shown in subsection 2.1.1. The idea behind using the LBP features is that the face images can be seen as composition of micro-patterns which are invariant with respect to monotonic grey scale transformations. Thus, initially the LBP operator is applied on the whole image. Then, in order to understand the distribution of regions encoded by LBP an histogram can be build as:

$$I\{A\} = \begin{cases} 1, & A \text{ is true} \\ 0, & A \text{ is false} \end{cases} \quad (2.9)$$

$$H = \sum_{x,y} I\{LBP(x,y) = \alpha\}, \alpha = 0, \dots, n-1 \quad (2.10)$$

where n is the number of patterns that the operator can generate and $LBP(x,y)$ has been defined in equation 2.5. For efficient face representation, spatial information should be retained, so the image is divided in $m \times n$ blocks $R_0, R_1, R_2, \dots, R_{n \times m-1}$ and for each of these blocks an histogram $H_{i,j}$ of the previously generated LBP patterns is evaluated using the above equations as:

$$H_{i,j} = \sum_{x,y} I\{LBP(x,y) = \alpha\}, (x,y) \in R_{i,j}, i = 0, 1, \dots, m-1, j = 0, 1, \dots, n-1. \quad (2.11)$$

Finally, these histograms are concatenated in order to build a single enhanced histogram encoding the facial features. Figure 2.9 summarizes the entire process.

The local binary pattern histogram encodes information at three levels of locality: the labels for the histogram contain information about the patterns on a pixel-level, the labels are summed over a small region to produce information on a regional level and the regional histograms are concatenated to build a global description of the face.

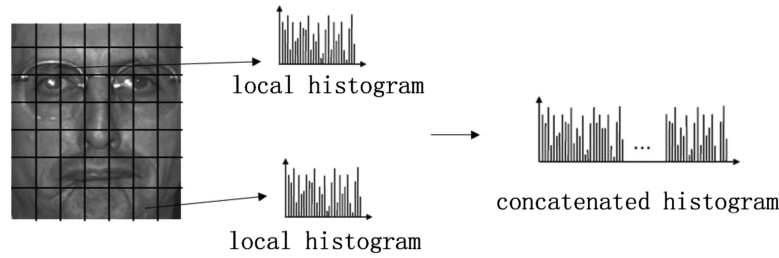


Figure 2.9

LBPH feature extraction for image divided in 7×7 regions (or patches). On each region the LBP operator is applied and an histogram of patterns distribution is evaluated. The LBPH feature is build by a concatenation of these histograms.

Some parameters can be chosen to optimize the performance of the LBPH. Ahonen et al. [14] also show how they affect the recognition capabilities. The first one is the LBP operator. Choosing an operator that produces a large amount of labels makes the histogram long and thus calculating the distance between features takes more time. On the other hand, using a small number of labels makes the feature vector shorter but more information goes lost. A small radius of the operator makes the encoded information in the histogram more local. Another parameter is the division of the images into regions $R_0, \dots, R_{n \times m - 1}$. The length of the feature vector becomes $B = (n \times m) \times B_r$, in which $n \times m$ is the number of regions and B_r is the LBP histogram length. A large number of small regions produces long feature vectors causing high memory consumption and slow classification, whereas using large regions causes more spatial information to be lost.

The Eigenface approach

The Eigenface method is based on the PCA algorithm for feature reduction. The key idea behind this algorithm is finding a set of orthogonal basis vectors of the training images. This allows to represent (and approximate) the original face image with its projection into a low dimensional space. Moreover, PCA provides an optimal linear transformation in the sense of least mean squared reconstruction error. Formally, let $X = \{x_1, x_2, x_3, \dots, x_n\}$ $x_i \in \mathbb{R}^d$ be a set of observations (e.g., training images or already extracted features, or ...). The algorithm follows these steps:

1. the mean samples value μ is evaluated as

$$\mu = \frac{1}{n} \cdot \sum_{i=1}^n x_i = \frac{1}{n} \cdot (x_1 + x_2 + \dots + x_n) \quad (2.12)$$

2. after that, the covariance matrix C is evaluated

$$C = \frac{1}{n} \cdot \sum_{i=1}^n (x_i - \mu) \cdot (x_i - \mu)^T \quad (2.13)$$

3. the eigenvalues λ_i and the eigenvectors w_i of C are evaluated

$$C \cdot w_i = \lambda_i \cdot w_i, \quad i = 1, 2, 3, \dots, n \quad (2.14)$$

4. the obtained eigenvalues and the corresponding eigenvectors are sorted in descendant order. The α principal components are the first α eigenvectors (*eigenfaces* in this context) relative to the largest eigenvalues. Let us call this set W

$$W = \{w_1, w_2, w_3 \dots w_\alpha\}. \quad (2.15)$$

Now, given W , the projection y_p of any new observation y can be evaluated using the following formula:

$$y_p = W^T(y - \mu). \quad (2.16)$$

Thus, the eigenface method proceeds this way:

1. every training set image of $h \times w$ pixels is reshaped in one-row or one-column vector;
2. the W matrix composed by eigenfaces is evaluated and the whole training set is projected into the subspace given by W ;
3. each new sample image q is projected into the subspace as well.

Table 2.1 shows some images of a face, their mean value and the eigenfaces extracted. A query image is then projected into the sub dimensional space and its reconstructions are shown while the number of eigenfaces used grows.

Despite the great space reduction, the eigenfaces approach suffers to changes in pose and illumination conditions of the faces. In fact, the principal components correspond to the components in the original image space with higher variance. If there is a change in the illumination condition, it is very likely that the principal components of these new images will be different. Another problem is that eigenface method described by equations (2.1), (2.2), (2.3), (2.4) must be done offline with an already available training set.

The Fisherface approach

The greatest lack of the Eigenface approach is that it doesn't consider the different classes in the training set. Hence, possible discriminative information may be lost considering only the components of higher variance of the original space. On the other hand, the Fisherface model, based on LDA, tries to find a linear transformation of data that maximize the between-class






Type of the image	Samples
Original samples (a)	
Mean value (b)	
Eigenfaces (c)	
Query (d)	
Reconstructed samples (e)	

Table 2.1

Eigenface example: a) a subset of the original images b) the mean value of these images c) subset of extracted eigenfaces d) query image and e) its reconstructed version while the number of eigenfaces used is incremented.

variance and minimize the within-class variance, in such a way faces belonging to the same person are clustered together in the new lower dimensional subspace and are far apart clusters containing faces of other people. Let denote with X , a set of images or observations (or features) $X = \{x_1, x_2, x_3, \dots, x_n\}$, $x_i \in \mathbb{R}^d$, belonging to C different classes. The algorithm evaluates the between-class and within-class scatter matrices S_b and S_w as follows:

$$S_b = \sum_{i=1}^C N_i \cdot (\mu_i - \mu)(\mu_i - \mu)^T \quad (2.17)$$

$$S_w = \sum_{i=1}^C \sum_{x_j \in C_i} (x_j - \mu_i)(x_j - \mu_i)^T \quad (2.18)$$

being N_i the number of elements for class c_i , μ_i the mean value for the class c_i , and μ the overall mean vector, that is $\mu = \frac{1}{C} \cdot \sum_i \mu_i$.

Then the algorithm looks for W_{opt} that maximizes the class separability criterion, defined as

$$W_{opt} = \arg \max_W \frac{|W^T S_b W|}{|W^T S_w W|}. \quad (2.19)$$

The solution for this optimization problem is given by solving the General Eigenvalue problem


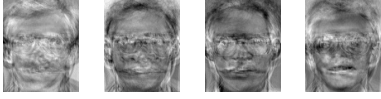
Type of the image	Samples
Original samples (a)	
Fisherfaces (b)	

Table 2.2

Fisherfaces approach example: a) samples of five subjects b) the four fisherfaces extracted by the algorithm.

$$S_b w_i = \lambda_i S_w w_i \quad (2.20)$$

$$S_w^{-1} S_b w_i = \lambda_i w_i \quad (2.21)$$

In [15] it has been shown that PCA feature reduction can be required in order to solve the problem. Again, this could lead to information loss. Table 2.2 shows the fisherfaces extracted from (a subset of) the ORL dataset ¹. Five subjects and their images have been processed. Notice that there can be at most $C - 1$ fisherfaces, if C is the number of available classes. The Fisherface approach finds face's features that allow to discriminate between persons and doesn't suffer from illumination conditions of the training set. The main problems, however, are still the required batch phase in order to train the model and the fact that the on-line update of the model is not supported. If samples of a new subject should be added, the whole process described by equations (2.17), (2.18), (2.19), (2.20), (2.21) must be executed again since a new fisherface needs to be find.

2.1.3 Classifiers for Face Recognition

In the previous section, algorithms to detect faces and extract facial features have been reported. The aim of these latter algorithms is to find facial features that can be used to let the system discriminate people. In order to do that, they are compared to assess, with a certain degree of confidence, if a new unlabelled face belongs to someone that system already knows.

This is a classification task that can be formulated as follows: given a set of labels or classes $\{c_1, c_2, c_3 \dots c_h\}$, a set of already labelled objects $\{o_1, o_2, o_3 \dots o_n\}$ (belonging to the *training set*,

¹<http://www.cl.cam.ac.uk/research/dtg/attarchive/facedatabase.html>

TS) and a query q (i.e., an unlabelled object), the problem is to assign a label to q .

In the literature, a lot of possible classifiers have been proposed. In this thesis, we test and compare: the first nearest neighbor (**1NN**), the k-nearest-neighbor (**kNN**) and the Weighted k-nearest-neighbor (**W-kNN**) ones that are described in the following. More complex classifiers such as Neural Network(**NN**) or Support Vector Machine (**SVM**) classifier have not been considered, due to the limited resources of the smart nodes, but they could be potentially applied on the server.

These algorithms rely on a distance (or dissimilarity) function $d(o_1, o_2)$, that allows objects comparison. It is usually defined as $d: \mathfrak{R}^N \times \mathfrak{R}^N \rightarrow \mathfrak{R}$, being N the dimension of the space in which objects lie. Different measures can be adopted, since they can perform better than others for a given feature.

1NN Classifier

Given an unknown face q and a set of known and labelled faces (o_i, c_i) , the **1NN** algorithm classifies q according to the label of the nearest object belonging to the training set for the query, in the feature space. Formally, the label returned by the algorithm is

$$1NN_C(q) = \{c_j : d(q, o_j) \leq d(q, o_i), \forall (o_i, c_i) \in TS\}. \quad (2.22)$$

In case of ties, different approaches can be followed such as peeking at random a label among the winners. The aim of the algorithms for facial features extraction is to generate patterns that lie near each other for the same faces and are far away from the features of different faces. Thus, a threshold t on the **1NN** search is used to discard too far away objects

$$1NN_C(q) = \{c_j : d(q, o_j) \leq d(q, o_i) \wedge d(q, o_i) \leq t, \forall (o_i, c_i) \in TS\}. \quad (2.23)$$

This threshold can force the response set to be empty. A degree of confidence of the response of the classifier is evaluated as

$$c_{1NN} = 1 - \frac{d(q, o_j)}{d_{max}} \quad (2.24)$$

where d_{max} is a distance used as normalization factor. The higher is the c_{1NN} value, the higher is the probability that object q has label c_j .

kNN Classifier

The kNN classifier is a simple machine-learning algorithm and a generalization of the $1NN$. It requires an integer parameter k to be chosen. The algorithm looks for the k nearest neighbors to the query q in the training set. Formally, the response set for the query is the following

$$kNN(q) = \{R \in TS \text{ s.t. } |R| = K \wedge \forall x \in R, y \in TS - R, d(q, x) \leq d(q, y)\} \quad (2.25)$$

Once this set has been evaluated, the *winner* class c_w is the most frequent one

$$kNN_C(q) = c_w = \max_{c_j} \{ |o_i \in kNN(q) |_{c_i=c_j} \} \quad (2.26)$$

Where $|\cdot|$ denotes the cardinality of a set. The label c_w will be assigned to the query. In case of ties, several approaches can be followed. In our implemented version of the algorithm, we choose among the set of winners (i.e., the classes that reach the same maximum score), the one that contains the object nearest to the query. A confidence value ranging in the interval $[0, 1]$, that is, as the name suggests, a degree of confidence of the response of the classifier is evaluated as:

$$c_{kNN} = 1 - \frac{\text{Number of elements of the second-most-frequent class} \in kNN(q)}{\text{Number of elements of the most-frequent class} \in kNN(q)} \quad (2.27)$$

If $c = 1$ all the elements in $kNN(q)$ belong to c_w . Otherwise elements of other classes are present in the response set. The higher is c_{kNN} , the higher is the probability that c_w is the correct label for q . Finally, notice that Equation 2.27 has a limited number of values for a given k value (see Table 2.3).

The k value can be difficult to choose since it often depends on the samples. When noise is present in the locality of the query instance, the noisy instance(s) win the majority vote, resulting in the incorrect class being predicted. Thus, a larger k could solve this problem. On the other hand, when the region defining the class, or fragment of the class, is so small that instances belonging to the class that surrounds the fragment win the majority vote, a smaller k could solve this problem.

Weighted kNN Classifier

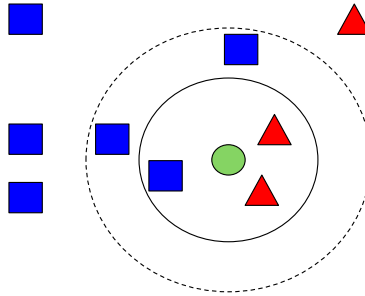
The kNN does not take directly into account the distances of the nearest k objects to the query. They are all treated in the same way and this can be the cause of misclassifications if

k	Set of values for c
2	$\{0, 1\}$
3	$\{0, \frac{1}{2}, 1\}$
5	$\{0, \frac{1}{3}, \frac{2}{3}, \frac{1}{2}, \frac{3}{4}, 1\}$
...	$\{\dots\}$

Table 2.3

kNN example of confidence values for a fixed k when equation 2.27 is used.

noise samples are near the query. Suppose, for example, a situation such as the one depicted in Figure 2.10.

**Figure 2.10**

kNN example in which the green ball must be labelled whereas the labels of the blue and red objects are known.

The green object is the unknown one that should be labelled, whereas the others belong to the training set and their labels are already known. Now, if $k = 3$ the green object will be classified as a red one by kNN . While, if we choose $k = 5$, the blue class will be the predominant one in the response set $kNN(q)$ returned by the algorithm. It is worth to note that the red objects are nearer than the blue ones. The key idea of the *weighted kNN* is to assign a *weight* or *score* to an object, according to its position with respect to the query. This score is assigned to each object belonging to the result set that the kNN would return and is evaluated as

$$s_{o_i} = 1 - \frac{d(q, o_i)}{d_{max}}, \quad (2.28)$$

where d_{max}^2 is used to normalize the real distance $d(q, o_i) \forall o_i \in kNN(q)$. Now, the algorithm evaluates the sum of this scores for the different classes present in the result set. That is, it calculates

$$z(q, c_j) = \sum_{o_i \in c_j} s_{o_i}, \quad \forall c_j \in kNN(q). \quad (2.29)$$

² d_{max} can be a known maximum value for the distance function we are using, or the distance of the farthest object to the query in the response set.

In the end, the label assigned to the query will be the one of the class that reaches the maximum value $z(q, c_j)$. Again, let us call it c_w . In this case, the confidence value returned by the algorithm has the following form:

$$c_{wkNN} = 1 - \frac{z(q, c_{w_2})}{z(q, c_w)}, \quad (2.30)$$

where we denoted with c_{w_2} the class that obtained the second higher score in $kNN(q)$. This classification confidence can be used to decide whether or not the predicted label has an high probability to be correct. Another kind of confidence we analyze is the following:

$$c_{wkNN_2} = \frac{\sum_i s_{o_i}}{k} \quad \forall o_i \in c_w \quad (2.31)$$

This value expresses the probability that q belongs to c_w . The k factor at the denominator is needed in order to have a value ranging in $[0, 1]$. The weighting schema reduces the probability of ties. Anyway, they can be handled with the same approach discussed so far.

2.2 Related work

In this section are reported some examples of works in which camera networks and computer vision algorithms are used.

Healthcare, teleimmersion, and surveillance

Ching et al. [16] discuss some motion-capture applications and systems, based on marker and markerless technology useful in various domains for human or object movement tracking. A review of other aspects, such as camera calibration is considered, for a 3D model reconstruction. Human movement tracking can be used in *healthcare* (for the initial diagnosis of movement-related disorders, patients remote monitoring), *teleimmersion* (3D videoconferencing, virtual presence, collaborative work and education) and for the *surveillance of public/private spaces* (human recognition, person reidentification). The main focus is on marker-less technology because it doesn't require markers to be put on human body or 3D scanners to be used. Hence, applications can be used for a wider area of situations and fields. However, this introduces the need of an accurate camera calibration in order to handle different environmental situations (low resolution of the cameras, overlap of camera views and so on).

In teleimmersion, different cameras viewpoints can be used to reconstruct a 3D model of the environment. These high resolution 3D videos are created for remote users interaction and

collaboration. Smart camera networks which are able to recreate a 3D model can be divided in a) passive reconstruction systems, in which multiple 2D cameras videos are combined together and b) active reconstruction systems, in which specialized 3D cameras hardware are directly used.

In video surveillance systems, multi camera networks help in facial gender identification, activity recognition and anomaly detection. A standard surveillance system usually performs a four-stage processing pipeline: spot the mover (*detection phase*) through background subtraction, computation of mover position in actual frame (*localization*) and in the next time interval (*tracking*), identification of the mover (*verification and recognition stage*).

Surveillance of public spaces

Abas et al. [17] report a taxonomy of wireless visual sensor networks (WVSN) for indoor and outdoor surveillance purpose that try to balance energy efficiency and application performance requirements. In contrast to sensor networks that report physical quantities represented by scalars (temperature, velocity, humidity, ...), sensor camera networks provide richer information such as audio and video. Unfortunately, this is achieved at higher energy, bandwidth and processing costs. Thus, the main challenge is the design of WVSNs able to reach efficiency, costs and application service requirements tradeoff. Already existent systems such as Citric, HuSIMS, OmniEye, Wi-FLIP, CamInSens, MeshEye are compared taking into account their energy efficiency, bandwidth efficiency, use of multi sensors, wireless technology, computer vision algorithms, the system software, whether cameras overlap and how transmission security is achieved.

The authors introduce also SWEETcam, a Solar Wi-Fi Energy Efficient Tracking camera system that tries to balance efficiency and costs requirements using solar panel energy. The Raspberry Pi equipped with Pi Camera module attached through CSI port to the main board is chosen as smart node. The computer vision software is developed in C/C++, through the OpenCV library. To save power SWEETcam uses the MSP430 micro controller to safely start and shut down the Raspberry PI. The controller has a passive infrared motion sensor (PIR) used to detect motion. It wakes up the system if something is perceived, or puts the system in sleep mode.

Person reidentification using spatiotemporal appearance

Gheissari et al. [18] present a new approach for the person reidentification problem over a smart camera network, based on the fact that people appearance in public spaces is almost the same in a given temporal window. Thus, instead of the face or other passive biometrics (such as gait), they consider the use of the overall appearance of the individual. In order to do that, the

system should be able to a) establish correspondence between parts and b) generate signatures that are invariant to changes of illumination, pose and dynamic appearance of clothing. The signature generation relies on a novel spatiotemporal segmentation approach that uses a new edge detection algorithm, that rejects temporally unstable edges, and color histograms in the HSV space. Two approaches have been proposed to generate correspondences a) a model based approach that fits an articulated model (through a decomposable triangulated graph) to each individual to establish a correspondence map, and b) an interest point operator (the Hessian Affine invariant) approach that nominates potential correspondences. They show that the model based approach is able to reach a higher true detection rate.

Chapter 3

Camera network

In this chapter the distributed face recognition system running on smart camera networks proposed in this thesis is presented and described. The main assumptions and goals while designing the system are reported. Client and server software architectures are shown.

3.1 Goals and assumptions

The main goal of the presented work is to design and implement a system able to:

- detect and recognize human faces;
- distribute across the nodes information in order to scale with respect the number of cameras the recognition task.

Application scenario We assume that cameras are fixed and they can communicate over a wired or wireless network. Each node has a list of available neighbors so that no discovering phase is needed. They continuously acquire a video stream looking for changes in the environment. When something happens, the face detection algorithm is turned on and node tries to detect faces. In order to reduce false positives and enabling face alignment, some face's landmarks are detected too. In particular, eyes' positions will be later used to align faces, enabling a robust face recognition phase. In fact, in-plane face rotation can be compensated and faces are transformed to reduce illumination and pose changes, allowing the system to compare same areas of faces during the recognition phase. If a node is not able to recognize a person using the locally cached classifiers, the central server is queried. If the server is able to recognize the incoming face, it sends back the unique identifier of the related person. The node can later ask for its classifier that is also sent to the nearest neighbors in order to reduce future requests to the server. This classifier contains facial features extracted by the LBPH algorithm. This algorithm has been chosen since it better fits the requirement of a dynamic system in which

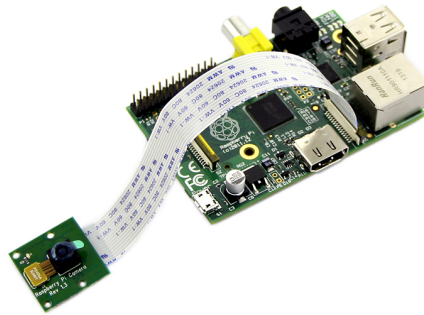


Figure 3.1
Raspberry Pi B+ equipped with Camera Pi module.

classifiers can grow (adding more images of an already present person or a new one) than the Fisherfaces algorithm. The latter creates a model for a given set of classes and it must be evaluated from scratch if a new person is added. Moreover, the authors of the LBPH [14] show that their algorithm outperforms the Eigenface method.

Future goals In the current system, in order to have a robust face recognition, faces' images are preprocessed in such a way that the position of the eyes is fixed in the transformed face. This requires both eyes to be present and limits the recognition capabilities if faces are rotated too much (e.g., when one of the eyes is not present at all). Of course, being able to recognize people in such different positions can be extremely important and useful in a surveillance application, since cameras can be positioned in many different ways and faces are not always captured frontally. A 3D modeling of the face is required for out-of-plane rotations and full face reconstruction. Furthermore, different illumination conditions and camera's resolutions should be taken into account, since they can affect both detection and recognition stages.

Computing platform Both the client and the server applications are written in Java. This ensures great portability and allows client software to run on different camera platforms, in which a Linux OS usually runs. Computer vision algorithms are implemented using the OpenCV¹ library. Moreover, some of the already available algorithms bundled in the library have been tested or used. Communication among nodes has been implemented using RESTful technology that runs over HTTP, using JETTY² as web server. The latter has been chosen since it allows to deploy and run the web server within the java application, whereas standard web servers often use the opposite approach and needs to be installed separately.

Raspberry PI The Raspberry Pi B+³ is a low-power, low-cost, credit-card sized computer with a 700MHz ARM processor, 512Mb of RAM and VideoCore IV GPU. Smart camera can

¹<http://www.opencv.org/>

²<http://www.eclipse.org/jetty/>

³<http://www.raspberrypi.org/>

be attached to the main board using the standard USB port or through CSI (Camera Sensor Interface), which enables a fast visual data acquisition while consuming less power with respect to USB interface. Its specifications make Raspberry PI board a good candidate as smart node device.

3.2 Client side software architecture

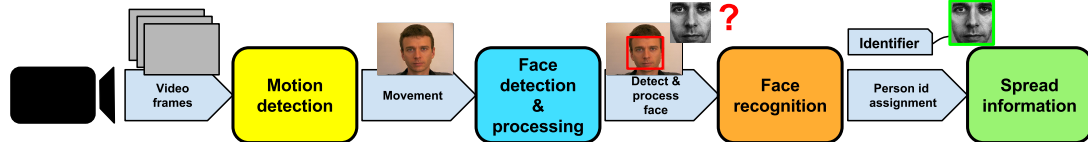


Figure 3.2
Client software pipeline stages.

Smart node software has been written as a pipeline of processing steps applied to the incoming video stream, as reported in Figure 3.2. A motion detection algorithm is applied and only when a movement is perceived the face detection algorithm is turned on. Face detection is a heavy task, so it is better to avoid if no faces are present at a certain time. When faces are found, they are preprocessed and sent to the next stage, in which a classification algorithm is locally applied using the stored classifiers. The latter stage could fail and a remote request could be needed. Finally, if an id can be assigned to the detected face, this information is spread among node's neighbors, together with the classifier of this person, for the future recognition of that face, allowing to reduce server involvement.

3.2.1 Stage 1: Motion detection

Face detection in frames coming by a video source is discussed in the next section. In Figure 3.4, the time needed for finding faces in frames of different size is reported, taking into account both MB-LBP and Haar features trained classifiers. It is also described how face detection time can be enhanced by frame downsampling technique. However, these operations still require a high CPU usage (see Table 3.1) that can be reduced if there is no face to be detected at a certain time through a motion detection algorithm.

Frame size	CPU usage (%)
320 × 240	70.3 %
640 × 480	90.0 %
960 × 720	96.0 %

Table 3.1
Raspberry PI CPU usage while running face detection for different frame sizes.

A trigger mechanism for face detection

Motion detection allows a system to determine if something in the environment is changing and can be applied in order to understand if faces are present in the scene. The output of this algorithm is a binary mask that specifies background and foreground pixels. Although the face detection could be performed only on the area in which a movement has been detected, this approach is not able to handle incoming faces in other areas of the frame. Hence, the face detection has to be performed on the whole frame. The key idea to reduce CPU usage on long run is to combine both kinds of detections and use motions as a trigger mechanism for performing face detection. When a motion is perceived, the motion detection algorithm stops and the face detection on the whole frame is applied. If no faces are found after a certain time, the motion detection algorithm is turned on again. Although the literature is plenty of algorithms and approaches for motion detection, most of them are based on **background subtraction** technique [1]. The idea behind this approach is that of detecting the moving objects through the difference between the current frame and a reference frame called 'background model' or 'background frame'. The output of this algorithm is a foreground mask in which a pixel is zero-valued if it is considered background, non-zero otherwise. A lot of techniques have been proposed to update the and maintain the background model in order to be robust to different kind of changes in the environment [19]. The simplest model of the background that can be maintained is the previous frame(s) or a frame captured when no other objects but background is presented in the scene. This technique is called *frame differencing*, since a difference between the actual frame $I(t)$ and the background frame B is evaluated. After that, if a pixel in the new image has a distance farther than a given threshold t from the background image, the pixel is considered a foreground pixel. The higher is the threshold value t , the higher is the admitted movement. Morphological filters (such as dilatation and erosion) can be applied in order to remove unwanted noise and single pixels recognized as foreground or background in a blob region of background or foreground pixels respectively. When the background model matches the previous frame(s), the method is able to adapt to changes in the environment. However, some problems that arise for particular object's speed and frame rate are present: *ghosting* and *foreground aperture*, shown in Figure 3.3. Collins et al. [20], resolve the ghosting problem through a combination of three frames. Differences between frames at time t and $t-1$, t and $t-2$ is performed and an AND logical operator is applied to the results. Finally, the thresholding method is applied to this new image. Foreground aperture problem has been faced in [21] by the same authors.

In our system, client node starts looking for changes in the environment, applying frame differencing and evaluating $diff(t) = |I(t) - I(t-1)|$. Where $I(t)$ is the current gray frame and $I(t-1)$ the background gray frame. A threshold is applied for each pixel of the $diff(t)$ frame

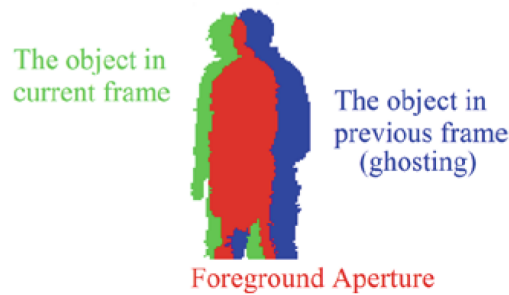


Figure 3.3

Frame differencing drawbacks: ghosting and foreground aperture, taken by [1]

in order to set the value of those pixels lower than the threshold to zero. This way they are recognized as background frames. Morphological erosion algorithm is then applied in order to reduce unwanted noise and holes in the foreground mask. Different problems can generate false alarms, such as fast light conditions changes. The standard deviation over the pixels of this new mask is evaluated, since higher value of the standard deviation often means no real movement but abrupt changes in the image. A different threshold t_2 can be applied to discriminate these cases. Finally, the number of changed pixel is evaluated and if it is over a certain threshold t_3 the face detection algorithm is turned on.

This algorithm has been implemented but its usage will be better exploited in future works, since a smart approach should be find and applied to schedule different activities on a single low-power CPU. Another more efficient approach would be to use a hardware sensor for detecting motions, allowing to

1. put in sleep mode or shutdown the Raspberry if no motion is detected and/or there are no tasks to be performed;
2. send an hardware interrupt if motion is detected to wake up the node.

In this way, the power consumption can be heavily reduced. Of course, a node must be awakened also if its neighbors or the server need to communicate with it. Finally, the FPS reached by the motion detection algorithm and the CPU usage for different frame sizes is reported in Table 3.2.

3.2.2 Stage 2: Face detection and face processing

When movement is perceived by a camera in the previous stage, the face detection algorithm is turned on and is applied over the entire frame, in order to check if faces are present. Even if it could be applied on the area in which the movement has been perceived, doing so incoming

Frame size	CPU usage (%)	FPS
320 × 240	42.3 %	22
640 × 480	56.2 %	9
960 × 720	74.5 %	5

Table 3.2

Raspberry PI CPU usage and FPS while running the motion detection algorithm. CPU usage can be reduced of roughly 30 % with respect the face detection algorithm (see Table 3.1).

faces in other areas of the frame would be lost. If no face is found, the next frame is analyzed until the face detection mode is turned off. In case faces are found, eyes detection is applied and faces are extracted for further processing (alignment, histogram equalization, noise reduction). Parameters can be tuned in both face and eyes detection phases, allowing to find more faces (eyes) consuming computational resources or being faster, dramatically reducing the computations needed while missing faces/eyes. In the experiments, they have been set in order to have a good tradeoff between these extreme cases but can be changed according applications needs. In the following, these parameters and how they affect the detection process are reported. Finally, the preprocessing stage that follows the face detection is reported too.

Face detection using OpenCV on the Raspberry PI

Face detection algorithms have been discussed in section 2.1.1. In OpenCV, the output of these algorithms after the training phase is a .xml that stores the parameters of the trained classifier. Applications that use OpenCV can load this file in order to perform face detection in images or video frames through the function

```
ObjectDetect.detectMultiScale(Mat image, MatOfRect rect, double
    ↪ scaleFactor, int minNeighbors, int flags, Size minSize, Size
    ↪ maxSize);
```

The Mat type stores internally the frames' pixels whereas MatOfRect is the return value of the method, in which bounding boxes for the detected objects are faces. The function requires also other parameters that can affect both the accuracy and the time spent for detection. Specifically

- **Size minSize/maxSize** it is the minimum/maximum possible window size, thus objects smaller/larger than that will be ignored. The minimum value is equal to the window that is used to train the classifier (in OpenCV it is a 24 × 24 window);
- **scale factor** as earlier stated, a cascade classifier is applied over a fixed size window that scans the image. With this approach, objects larger than the sub window will not be detected. In order to find them two approaches can be followed: sub-window enlargement or image pyramid construction through initial image rescaling. This latter operation is

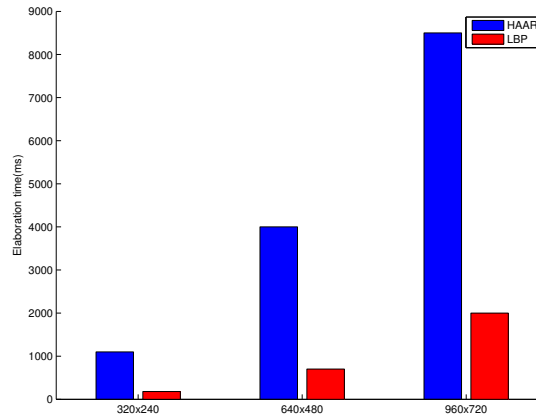


Figure 3.4

Face detection executed on the Raspberry Pi and applied on stream coming from Raspberry Pi camera. MB-LBP and Haar trained detectors are compared while varying the frame size. The time needed by the former detector is always smaller.

preferable since it can be performed at lower cost. The number of images of this pyramid is related to the maximum/minimum size and the scale factor. This value ranges between 1.01 and 1.5. When the smallest value is used, the detection process is slow but more faces can be detected, whereas a lot of faces are missed if the larger value is chosen;

- **minimum number of neighbors** while scanning the image, the same face can be found in more adjacent sub windows. The higher the number of overlapping windows, the higher the confidence that it is a real face/object. A standard value for this parameter is 3;
- **other flags** often used to speed up the whole detection process. For instance, an edge detector can be applied to determine flat areas that have low probability to contain objects, hence they will not be considered.

Figure 3.4 reports the time needed for face detection on the Raspberry Pi, while varying the frame size and with a single face to be detected using MB-LBP and Haar trained detectors available in OpenCV.

For small frame's sizes, face detection can be applied in around two hundred milliseconds using the MB-LBP detector, that results to be faster in each of the three situations. Unfortunately the smaller is the frame, the lower is the probability to detect faces far from the camera. Anyway, the following considerations must be taken into account. In order to boost the recognition phase, extracted faces need to be processed in order to align them and compensate illumination problems. The alignment phase relies on eyes coordinates estimation. Hence, after face detection eyes detection is performed through trained classifier for eyes using the already seen method `detectMultiScale()`. Independently from the frame size, the minimum face that can be detected must have size 24×24 , since the classifiers have been trained using that window. Eyes detectors, instead, have been trained using a 20×20 pixels window. Due to this fact and

Feature	Frame size	Minimum size window	FPS
MB-LBP	640×480	(50, 60)	3.8
MB-LBP	320×240	(25, 30)	5.9

Table 3.3

Face detection executed on the Raspberry PI and applied on the video stream acquired by PI Camera module. The detection is faster for the downsampled frame version in which a scaled minimum window is used.

taking into account the structure of the face, eyes could be detected in a face or equivalently in a window that is at least 50×60 pixels. This window can scan a 640×480 pixels frame or an even larger frame, but a smarter approach is to use a 320×240 downsampled frame and scan it with a 25×30 . Once faces are detected, the obtained bounding box can be rescaled in the original frame, in which eyes detection can be directly applied. Eyes detection should be applied at full size, because the probability to find eyes becomes higher. Table 3.3 shows that this approach achieves higher FPS.

Image processing

A lot of techniques have been proposed to contrast light and head pose variations with the aim to achieve face normalization. However, these are only a part of the many problems that a recognition system has to solve. Faces captured at different scales and/or by different cameras, aging effect, occlusion and different expressions make the recognition process even harder.

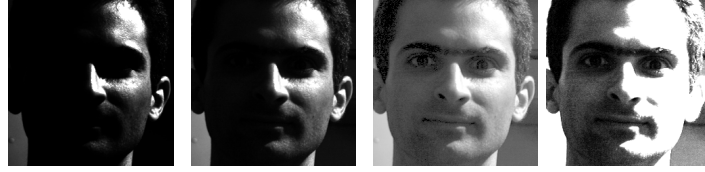
The Gamma intensity Correction (GC), the Logarithm Transform (LT) and the Histogram Equalization (HE) methods can be applied for the lighting variations problem [22] [23]. However, the first two approaches require parameters to be chosen which in turn depend on the light under which faces are captured and so they are difficult to tune. The HE approach instead, doesn't require parameters to be shown, thus it is used in the current version of the system. Moreover, it can be used to handle different light conditions to the two halves of a face.

Gamma Correction This method allows to change pixel's intensity through the following (power law) formula

$$f(I(x,y)) = c \cdot I(x,y)^{(1/\gamma)}, \quad \gamma, c \in \mathfrak{R}. \quad (3.1)$$

The whole image's illumination changes according to the γ and c values. A high value of γ makes brighter the image.

Logarithmic Transformation In this case, a logarithmic function is used to transform pixel

**Figure 3.5**

Gamma and logarithmic correction examples: (from left to right) $\gamma = 0.5$, $\gamma = 1$ (original image), $\gamma = 3$ and logarithmic image with $c = 10$

intensity such that

$$f(I(x, y)) = c \cdot \log(I(x, y) + 1), \quad c \in \mathfrak{R} \quad (3.2)$$

The original image, its gamma and logarithmic versions are reported in Figure 3.5.

Histogram Equalization Histogram equalization tries to normalize light distribution over the image, in order to improve image's contrast and brightness. The key idea is to change the actual pixels' intensity distribution to another, more uniform, distribution. A gray scale image can be considered, whose pixels' values are in the range $[0, 255]$. The occurrences of each intensity value are evaluated and the obtained histogram is normalized in order to obtain a probability mass function (*p.m.f.*). Thus, the cumulative distribution function (*c.d.f.*) can be evaluated as

$$H(i) = P(p_{(x,y)} \leq g_i) = \sum_{0 \leq j < i} h(j), \quad (3.3)$$

where $h(j)$ is the probability that pixel at coordinates (x, y) has intensity value g_i . This *c.d.f.* is now used as Look-Up-Table to transform the original pixel value v , to a new value v'

$$v' = \text{round}\left(\frac{\text{cdf}(v) - \text{cdf}_{\min}}{W \cdot H - \text{cdf}_{\min}} \times (L - 1)\right). \quad (3.4)$$

L denotes the number of gray values, cdf_{\min} is the minimum non-zero value of the cumulative distribution function, W and H the width and height of the original image respectively.

In real world scenarios, it is common to have different light conditions to the two face's halves. The face image reported in previous figures is an example of such situation. Histogram equalization can be applied to both parts separately, although the strong light effect cannot be removed totally.

**Figure 3.6**

Histogram equalization example: (from left to right) original image, equalized, left and right equalized separately.

Head pose problem can be solved through facial landmarks (such eyes' centers, mouth, tip of the nose) coordinates estimation and geometric transformations. Complex methods allow to compensate several types of out-of-plane face rotations through a reconstructed 3D model of the face [13]. Finally, a representative training set which contains faces captured under different poses, is also a powerful mean to enhance face recognition's accuracy, if no or simple geometric transformation is applied.

In the following, a transformation of the 2D captured face is reported, based on [24]. A proper 2D alignment, often relies on several facial features detections. Since detections are costly operations to perform on a low power CPU, only eyes' centers coordinates will be extracted. After that, a geometrical affine transformation is applied in order to

- rescale the captured face to a fixed size;
- compensate in-plane rotations of the face;
- translate the face in such a way eyes are at fixed position;
- crop non interest part of the face image, such as hair, forehead, ears, and chin.

Feature	time (ms)
LBP	180
Haar	410

Table 3.4

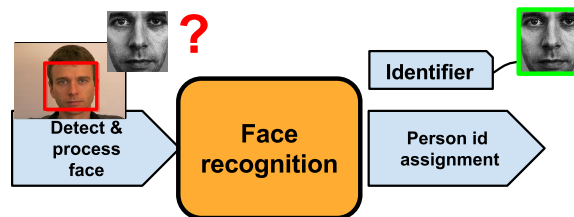
Eyes detection time for LBP cascade classifier and Haar cascade classifiers, on Raspberry PI, applied to image in Figure 3.6

A trained LBP cascade classifier is used to speed up the eyes detection process (see Table 3.4). Moreover, due to the structure of the face, eyes detection can be applied only on the top-half of the extracted area containing the face, reducing the search time. Finally, an elliptic mask can be used to remove the remaining background. The equalized image in Figure 3.6 is transformed as reported in 3.7.

**Figure 3.7**

Final processed frame: central eyes coordinates found (left), scaled and rotated face (center), elliptic mask applied (right)

3.2.3 Stage 3: Face recognition

**Figure 3.8**

Classification process: after face detection in video frame, the face is extracted and processed in order to normalize its appearance and to extract facial features. Then, a classifier is applied to assess if similar features are present among the ones of already labelled faces.

In this stage, a node tries to recognize detected faces using the information it has in its internal cache. This cache is handled using a LRU policy, and contains features of the last recently seen people by the camera and its neighbors. An initial list of features is obtained at bootstrap from the server but it dynamically changes when nodes exchange information. So, features from the detected faces are extracted and a classification algorithm is applied in order to check if a good match exists locally. At each time, nodes and server can have different information since node's cache is supposed to contain less classifiers; in the case in which the node isn't able to recognize a face, it asks to the server. The server receives the already aligned and cropped face image and checks if it is able to recognize it. In case it succeeds, it sends back the id of the recognized person to the node. If the classifier for this person is needed by the client, it could later ask it. It could happen that the classifier is available in node's cache but the local face recognition failed. The main reason for such a situation is that more features for that face have been added on server side by a human operator. In this case, node deletes the already present classifier and asks for the new one.

3.2.4 Stage 4: Neighbors communication

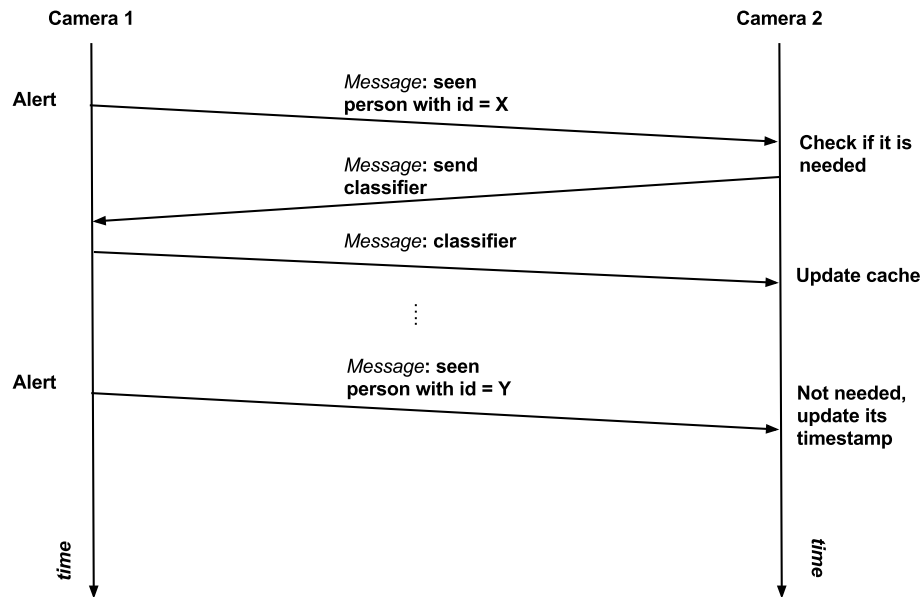


Figure 3.9

Handshake for node communication: in the first case (top) the classifier is needed by neighbor Camera 2 whereas it is not needed in the second case (bottom) but its timestamp is updated since it could be used in a short time.

In this stage, the information about the just recognized person is spread among neighbors' node. Since a neighbor could already have this classifier, a handshake is introduced in order to reduce data to be exchanged, see Figure 3.9. Thus, a node tells the others that it has just seen person with identifier X and that it has available the classifier for that person. Neighbors check if a classifier with this identifier is present in their own caches and if it is so, the classifier is not needed. However, in these caches, that classifier will be moved and considered as the most recently used so that it won't be discarded as next. Neighbors that do not have it, can later ask for this classifier. Different communication strategies for the alert messages have been simulated since the information can be sent to the overall network of cameras, to the nearest neighbors or to no one. More details and results of such simulations are reported and discussed in section 5.3. Nodes communicate through the HTTP protocol, using a simple stateless technology known as REST (REpresentational State Transfer). Parameters can be put in POST or GET requests by clients and the http server can return data through standard and independent machine format such as JSON or XML. Clients expose a simple interface through which they offer services that are reported in Table 3.5.

NODE	SERVICE NAME	DESCRIPTION
Client	/seenPerson	A client invokes this method exposed by its neighbors to inform them that it has seen a given person. A JSON string encoding the ID of this person is sent as parameter.
Client	/giveMeTheClassifierOf	A client invokes this method to require a classifier to a neighbor. The id associated to this classifier is passed as argument and the classifier (encoded as JSON string) is sent back.

Table 3.5

Services offered by clients and invoked by neighbors

3.3 Server side software architecture

FIELD	DESCRIPTION
ID	Unique id assigned to this sample
PERSON ID	Id of the person to who this face belongs
TIMESTAMP	Time in which this feature has been generated
IMAGE URL	Path of the original image from which the LBPH feature has been extracted (for debugging purposes)
FEATURE	Binary serialization of the LBPH feature of this sample

Table 3.6

Database entry description of a sample belonging to the training set

A server is needed in order to maintain and distribute the knowledge base of the whole system. Its internal SQLite⁴ database relates the features of the face of a given person to his/her unique identifier. Each entry of this database has the structure reported in Table 3.6. Moreover, the server waits for remote requests and apply a classification algorithm in order to assign an identifier to the incoming face, when a node is not able to do so with its local information. It has also the role of storing the faces of unrecognized people detected by cameras. These faces can be manually annotated by an operator as belonging to a already known person or a totally new one. Unsupervised algorithms have been tested in order to automate such task and their resulted are reported in section 5.4. Finally, when a node wakes up, the server is queried for a configurable number of recent classifiers, so that client's cache is never empty. Future work will try to enhance server's tasks in different directions. Server's services reachable through HTTP

⁴<https://www.sqlite.org/>

NODE	SERVICE NAME	DESCRIPTION
Server	/classify	Clients invoke this method to ask server classification of a detected face. The encoded face is sent as parameter.
Server	/giveMeTheClassifierOf	Clients invoke this method in order to request a classifier. The ID of the person is passed as argument and the JSON encoded classifier is sent back.
Server	/giveMeLatestUsedClassifier	Method invoked by clients in bootstrap phase in order to get a list of classifiers of the last seen people

Table 3.7*Services offered by the server*

are reported in Table 3.7.

Chapter 4

Exploiting unrecognized faces

Smart nodes try to recognize faces they detect but when the local classifier fails, a remote request is performed. In this case, they send this image to the server asking for prediction. Even if they could send the LBPH feature related to it, avoiding feature re-extraction, doing so doesn't allow a further analysis of this information. The first operation that the server does is to re-extract the feature and try to classify the face it encodes. However, in the case in which the server itself isn't able to recognize the person within the image, a human operator can inspect and label the original face images. This task can be optimized using clustering algorithms through which the system organizes this potentially huge amount of data coming by clients. In this way, the human operator has simply to label entire clusters and/or correct errors occurred during the clustering process.

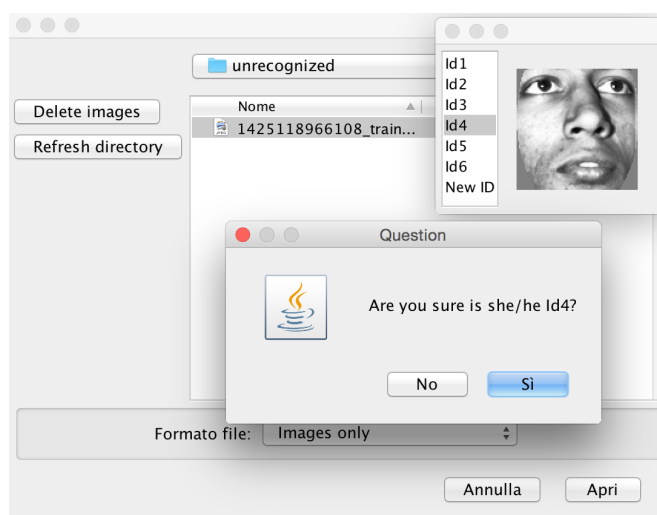


Figure 4.1

Manual annotation tool for unrecognized faces. The panel gives the opportunity to associate this image with an already present person or to add a new subject and associate the face with him/her.

Faces sent by clients can belong to unknown people or to already known ones, since it can

happen that the local client classifier fails if faces are captured under new poses or illumination conditions or expressions. Without a proper preprocessing of these images an operator can easily get bored, since he is forced to use a manual annotation tool (see Figure 4.1). In order to help him, two clustering algorithms have been implemented and tested and a) the error produced while they connect images in clusters and b) the number of clusters for person has been evaluated, while varying a distance threshold t .


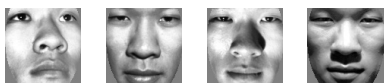
Class label	Samples
YaleB11	
YaleB12	
YaleB13	
...	...

Table 4.1

Already processed images sent by clients after local recognizer failure.

Table 4.1 shows some samples images to be clustered. Those images are supposed to come from clients, hence all the detection and alignment processes have been already performed.

The threshold t is applied between LBPH features extracted from the images, but it has a different meaning in the two approaches. Specifically, let $O = \{o_i, l_i\}$ the set of labelled samples. The entire dataset is shuffled and objects are collected in clusters following the two algorithms described in the following.

The distance between an object o_i and a cluster C_j is defined as

$$d(o_i, C_j) = \min_x \chi^2(o_i, o_x), o_x \in C_j. \quad (4.1)$$

First clustering algorithm In this case, a new coming image is attached to the cluster that contains the more similar (1NN strategy) face. In case of ties, it is randomly associated to one of those clusters. If the distance returned by 1NN with respect a certain cluster C , let us denote it with $d(C, q)$, is higher than t , the creation of a new cluster containing the new element q is forced. With this approach an object belong to only one cluster, hence no cluster merging techniques are exploited. The algorithm schema is reported here.

Algorithm 1 Clustering algorithm

```

 $C = \{\emptyset\}$  ▷  $C$  is the current set of clusters
for all  $\{o_i, l_i\} \in O$  do
   $l^* \leftarrow \emptyset$  ▷  $l^*$  will be the label of the nearest cluster (if any)
   $d_{min} \leftarrow +\infty$ 
  for all  $C_j \in C$  do
     $d \leftarrow d(o_i, C_j)$ 
    if  $d < d_{min}$  then
       $d_{min} \leftarrow d$ 
       $l^* \leftarrow j$ 
    end if
  end for
  if  $(l^* == \emptyset) \vee (d_{min} > t)$  then
     $C_{new} \leftarrow \text{new Cluster}(o_i)$ 
     $C \leftarrow C \cup C_{new}$ 
  else
     $C_j \leftarrow C_j \cup o_i$ 
  end if
   $O = \{O\} \setminus \{o_i\}$ 
end for

```

Second clustering algorithm In this second case, a coming image can be useful to merge two or more already existent clusters. This operation could hopefully merge clusters containing images of the same person but in different poses or under different illumination conditions, expressions and so on. This time, the algorithm consider an already present cluster if $d(o_i, C) < t$. If none of them is returned, a new cluster is created. The algorithm schema is reported here.

Algorithm 2 Merge clustering algorithm

```

 $C \leftarrow \{\emptyset\}$  ▷  $C$  is the current set of clusters
for all  $\{o_i, l_i\} \in O$  do
   $L \leftarrow \{\emptyset\}$  ▷  $L$  is the set of nearest clusters for  $o_i$ 
  for all  $C_j \in C$  do
     $d \leftarrow d(o_i, C_j)$ 
    if  $d < t$  then
       $L \leftarrow L \cup C_j$ 
    end if
  end for

```

Algorithm 3 Merge clustering algorithm (continued)

```

if  $L$  is empty then
     $C_{new} \leftarrow new\ Cluster(o_i)$ 
     $C \leftarrow C \cup C_{new}$ 
else
     $C_{new} \leftarrow new\ Cluster(o_i)$ 
    for all  $C_j \in L$  do
         $C_{new} \leftarrow C_{new} \cup C_j$ 
    end for
     $C \leftarrow C \cup C_{new}$ 
end if
 $O = \{O\} \setminus \{o_i\}$ 
end for

```

For both of them, the error is evaluated using *macro* and *micro* averaging techniques, each time a new image is given as input to the algorithm and considering as label of a cluster the object's label more frequent in it. This value is found using the *majorLabel()* function, reported in Algorithm 4. It simply counts the number of times a certain label l_i is present in C_j , returning the one that reaches the highest number of occurrences. If C_j is an object, its label will be returned.

Algorithm 4 Subroutine *majorLabel()*

```

function  $L(C_j)$ 
     $L_s \leftarrow \{(l_i, n_{l_i}), (l_i, o_i) \in C_j\}$   $\triangleright L_s$  is the set of pairs  $(l_i, \text{number of objects with label } l_i \in C_j)$ 
     $l^* \leftarrow \{l_a : n_{l_a} > n_{l_b}, \forall (l_a, l_b) \in L_s\}$ 
    return  $l^*$ 
end function

```

The micro and macro average error metrics are evaluated using the following notations. Let us call n the number of built clusters at a given time instant, TP_i the number of samples correctly assigned to cluster c_i , FP_i the number of wrongly assigned object to c_i and P_i the total number of elements of cluster c_i .

Micro average error

$$mAE = \frac{\sum_{i=1}^n FP_i}{\sum_{i=1}^n P_i} \quad (4.2)$$

It evaluates how many objects have been put in clusters whose major label is different than the object's label.

Macro average error

$$MAE = \frac{\sum_{i=1}^n FP_i}{\sum_{i=1}^n P_i} \cdot \frac{1}{n} \quad (4.3)$$

Macro average measure gives the same weight to big or small clusters whereas micro measure gives a better idea of the error committed while building clusters.

The result of the tests related to these algorithms are reported in section 5.4.

Chapter 5

Experiments and results

In this chapter are reported a) the performance obtained by the proposed classifiers in section 2.1.3 b) simulation tests of a smart camera network in which the developed software runs and c) the tested clustering algorithms for collecting faces of unrecognized people proposed in the previous chapter.

5.1 Dataset of faces

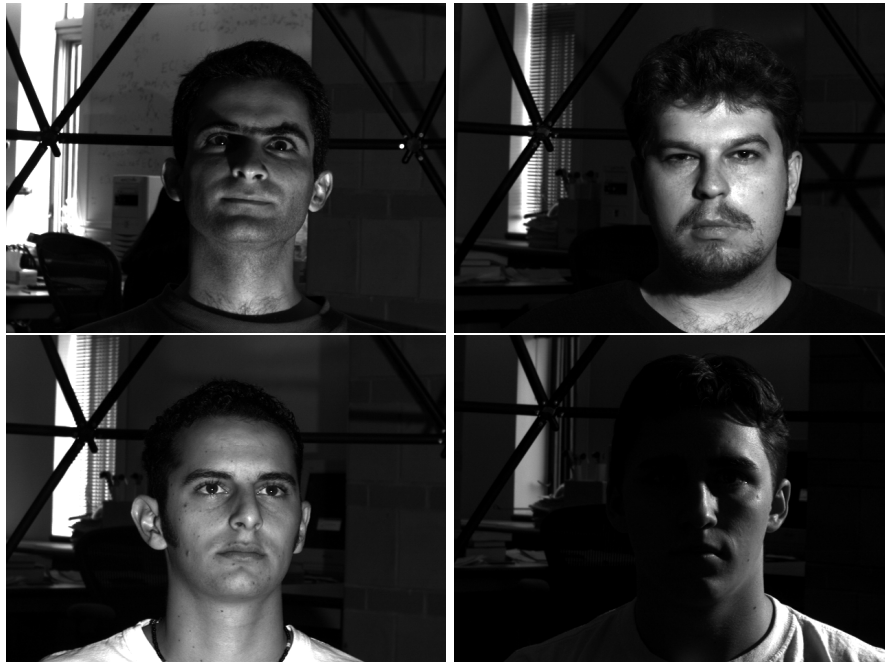


Figure 5.1
Yale dataset samples

Recognition and clustering tests have been performed using the *Extended version of the*

Yale Database [25], freely available¹. The dataset contains 16128 images of 28 humans captured under different and challenging illumination conditions and frontal poses. Samples of the dataset are reported in Figure 5.1. The Yale dataset is widely used in the literature for faces preprocessing techniques tests and face detection and recognition tasks evaluation [22] [26] [27].

5.2 Recognition performance

From the pattern classification point of view, a usual problem in face recognition is having few, possibly one, training sample(s) per class and a huge number of faces to be classified. Thus, from the whole Yale Dataset, 350 images for each person have been taken. Among these, 300 are used as test set and the remaining ones are used for the training set of each person. The LBPH feature has been chosen to extract facial feature. There are some optimization that can be done for this feature such as the radius and the neighborhood of the LBP operator, and the number of vertical and horizontal patches that cover the whole face. In [14], it has been shown that the performances of the LBPH are quite stable with respect to the number of patches (or regions) in which the image face is divided for evaluating the histograms of distribution of the LBP labels. However, a good tradeoff is to use a grid of 8 horizontal and vertical patches. For the LBP operator, the standard one with radius 1 and neighborhood equals to 8 captures enough local information of the face, hence this one will be used. The same authors also show that the Chi-Square distance, defined as

$$\chi^2(H_1, H_2) = \sum_i \frac{(H_{1_i} - H_{2_i})^2}{(H_{1_i} + H_{2_i})} \quad (5.1)$$

gives better results than the Histogram intersection or Log-likelihood statistics for LBPH features comparison. The *1NN*, *kNN* and *weighted kNN* classifiers, described in section 2.1.3, have been tested and their accuracy is reported. For each experiment, we underline the threshold or the *k* parameter, the type of confidence in use and so on. In order to assess the goodness of the results, *cross-validation* is performed.

Furthermore, the Principal Component Analysis algorithm is applied on the LBPH feature in order to reduce the feature length we get. The dissimilarity measure after PCA is the Euclidean distance, defined as

$$d(x, y) = \sqrt{\sum_{i=1}^N (x_i - y_i)^2}, \quad x, y \in \mathbb{R}^N. \quad (5.2)$$

¹<http://vision.ucsd.edu/~leekc/ExtYaleDatabase/ExtYaleB.html>

LBPH feature extraction	$\chi^2(H_1, H_2)$
≈ 30 ms	≈ 9 ms

Table 5.1

Time required for LBPH feature extraction (100 × 100 pixels image as input) and χ^2 distance evaluation on the Raspberry PI

First, the experiments with the LBPH feature will be shown. Then, we will apply the Principal Component Analysis to this feature.

Table 5.1 summarizes the LBPH feature extraction time and the χ^2 distance evaluation on the Raspberry PI. In order to compare the results, the obtained **accuracy**, defined as

$$a = \frac{\text{number of objects correctly classified}}{\text{total number of classified objects}} \quad (5.3)$$

as well as the number of elements **classified** is considered. The latter is useful since it is not always equal to the test set cardinality. In fact, a tradeoff between accuracy and number of classified faces is obtained by thresholding the confidences of the classifiers.

5.2.1 LBPH

1NN

In Figure 5.2 the results of the 1NN classifier are shown for different training set sizes and for different values of the threshold t . The first nearest neighbor algorithm has been applied varying:

- the threshold t for the distance $d(q, o_i)$ in the range $[0, 40]$. While scanning the training set an object o_i won't be considered if $d(q, o_i) > t$;
- the number of training samples per class from 5 up to 50 with steps of 5.

A confidence for the response of the 1NN classifier can be evaluated as

$$c_{1NN} = 1 - \frac{d(q, o_j)}{d_{max}} \quad (5.4)$$

where o_j denotes the nearest face to the query q . This value is used to understand how much the classifier is confident of the label that is assigned to the query. However, since the confidence is related to the distance of the nearest object $d(q, o_j)$, the following plots are reported considering this distance and the threshold t applied on it.

When the threshold t increases, the classifier is able to label more faces but the accuracy obtained decreases. For a threshold lower than 20, an accuracy near to one is achieved. However, for the same threshold and an higher number of samples in the training set, more faces can be classified with an increasing accuracy. For example, if 5 samples for each class are available in the training set and we are using a threshold equals to 20, the classifier is able to label 30 faces on 100 with an accuracy of 0.9. If the same threshold is used but the number of training samples per class grows to 50, the 70% of the faces can be classified with an accuracy of around 0.95. Finally, for higher threshold values, the classifier labels each face while the accuracy remains constant since the distance of the nearest object from the query doesn't change anymore. Thus, if an accuracy higher than 0.9 is required, a threshold lower than 20 must be used.

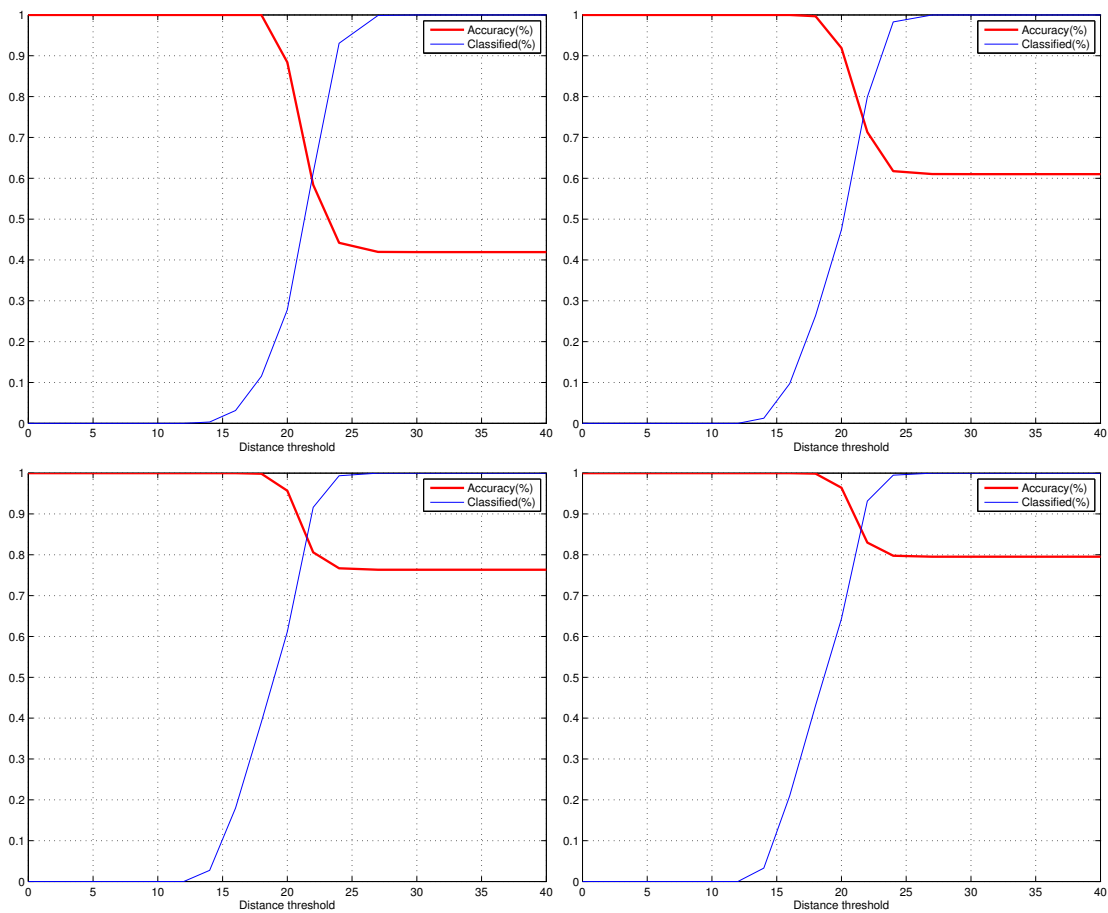


Figure 5.2

INN accuracy and percentage of classified faces for different training set sizes (from top-left to bottom-right): 5, 15, 40 and 50 samples for class. For the same threshold t , if the number of samples in training set is incremented more faces can be labelled and with higher accuracy.

Figure 5.3 relates the accuracy and the percentage of classified objects considering the number of samples in the training set for each class. For example, from this plot we can deduce that if we want to be able to classify 8 times on 10 (that is, 80 % of classified elements), and we need

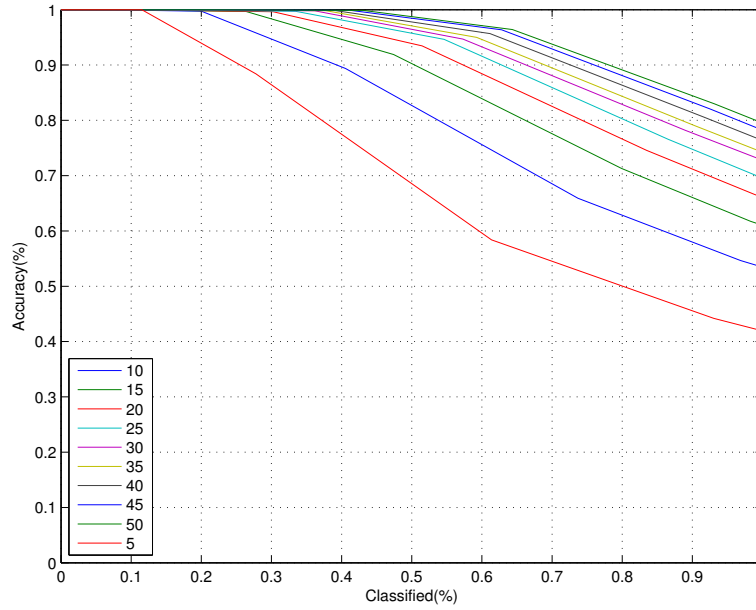


Figure 5.3

INN (Accuracy(%) vs Classified(%)) when the number of training set samples for each class increases. The legend in the bottom-left part of the plot shows the number of samples for each class of the training set. More faces can be classified with higher accuracy when the samples number increases.

high accuracy (suppose 90%), 50 samples per class are a good starting point for the TS.

kNN

The kNN algorithm has been applied varying:

- the k parameter in the range $[0, 100]$;
- the number of training samples per class from 5 up to 50, with steps of 5.

A confidence for the response of the kNN classifier can be evaluated as

$$c_{kNN} = 1 - \frac{\text{Number of elements of the second-most-frequent class} \in kNN(q)}{\text{Number of elements of the most-frequent class} \in kNN(q)} \quad (5.5)$$

where the set $kNN(q)$ is evaluated through equation 2.25. The k_{opt} , for which the highest accuracy can be reached, changes for different sizes of the training set from 2 to 3. For training set sizes lower than 45 samples per class, the k_{opt} is equal to 2 and the accuracy is slightly higher than the one reached by $k = 3$, whereas for 45 samples and 50 samples for class the accuracy when $k = 3$ is used is slightly higher than the one for $k = 2$. If the confidence c_{kNN} is not taken into account, the kNN always classifies each element of the test set.

Now, we consider the confidence expressed by equation 5.5 whose values are reported for some k in Table 2.3. In Figure 5.4 is reported how the accuracy and the percentage of classified elements change when the accuracy grows. The same behavior shown by the 1NN classifier in Figure 5.2 holds for the kNN classifier: when the number of samples in the training set of each class grows, more faces can be labelled and with higher accuracy, for a given fixed confidence. On the other hand, if the confidence required grows, the accuracy grows as well whereas the number of faces classified decreases, for a fixed training set size. The main problem however, is that even if 50 faces per person are used in the training set, the kNN classifier is not able to reach 100 % of accuracy.

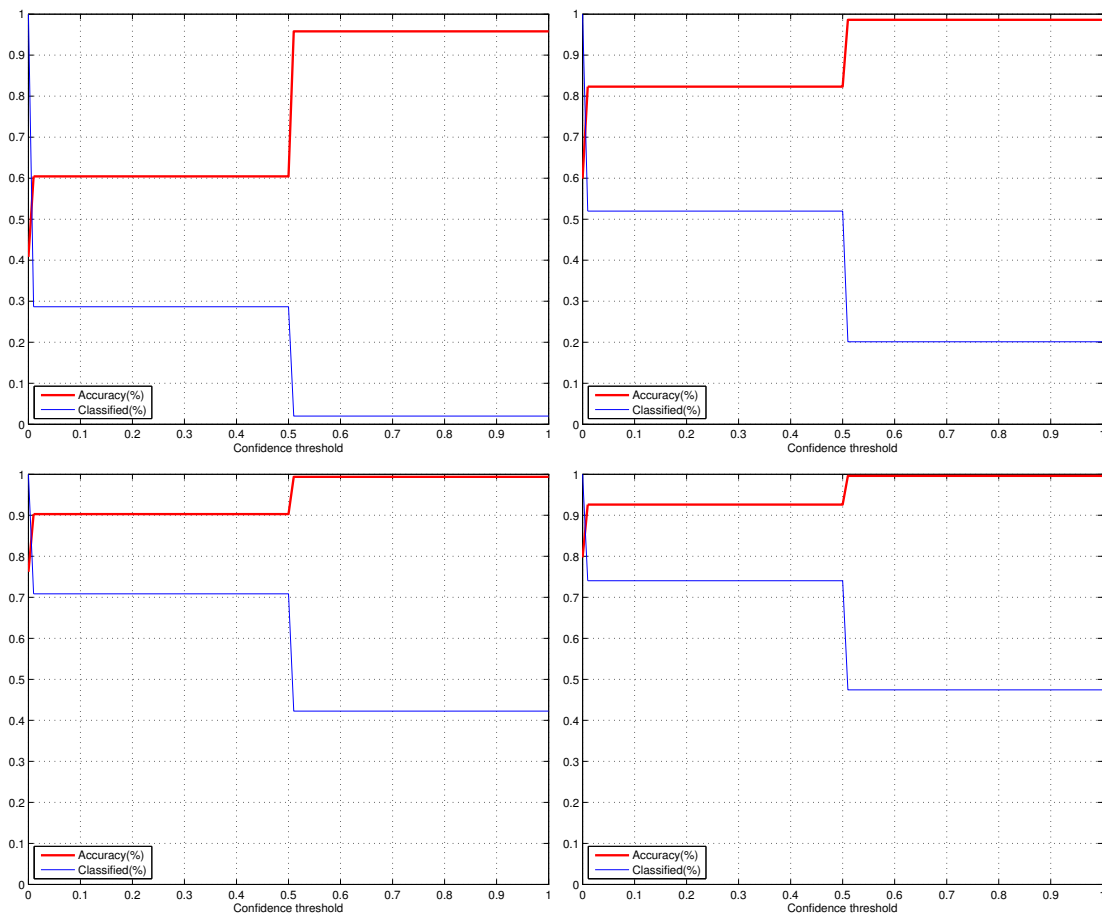
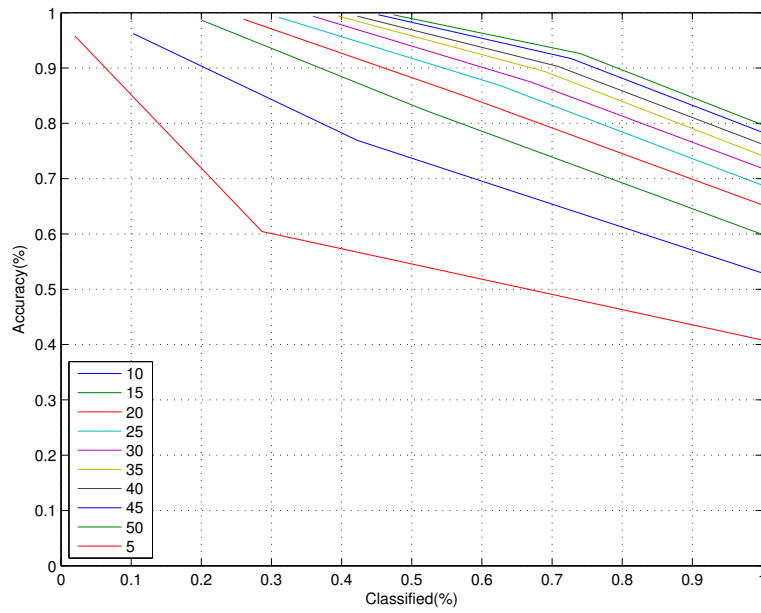


Figure 5.4

kNN accuracy and percentage of classified faces when $k = 3$, the confidence grows and different training set sizes are used (from top-left to bottom-right): 5, 15, 40 and 50 samples for class. When the confidence grows, the accuracy of the classifier grows as well whereas the number of classified faces decrease for a fixed training set size. Moreover, when the number of samples in training set is incremented more faces can be labelled and with higher accuracy for a fixed required confidence.

Figure 5.5 shows when $k = 3$ has been fixed, how the percentage of classifiers and the accuracy reached are related. This plot can be compared with the one in Figure 5.3: in this case the algorithm is not able to reach the 100% of accuracy.

**Figure 5.5**

k NN with $k = 3$ (Accuracy(%) vs Classified(%)) when the training set size increases. The legend in the bottom-left part of the plot shows the number of samples for each class of the training set. Note that 100% of accuracy is never reached with these training set sizes.

Weighted kNN

	k					
TS images		5	7	10	20	30
5		0,4192	0.42025	0.41844	0.40773	0,3936
10		0,5399	0.54165	0.53734	0.51384	0,5017
15		0,615	0.62022	0.61994	0.60882	0,5932
20		0,666	0.66861	0.66945	0.66083	0,6509
25		0,7014	0.70449	0.70519	0.69629	0,6894
30		0,7299	0.73425	0.73522	0.7298	0,7206
35		0,7489	0.75414	0.75817	0.76053	0,7514
40		0,7709	0.77458	0.78125	0.78612	0,7761
45		0,7916	0.79586	0.79808	0.80378	0,7970
50		0,8035	0.80809	0.81129	0.81602	0,8108

Table 5.2

Accuracy for some k values while increasing TS (best values are in bold) and applying the weighted kNN algorithm. Note that the accuracy obtained for the k_{opt} when the training set size grows doesn't change too much. Thus, $k_{opt} = 20$ is used in the following for the weighted kNN, although for efficiency $k = 7$ could be used.

The weighted kNN has been applied varying:

- the k parameter in $[0, 100]$;
- the number of training samples per class from 5 up to 50, with steps of 5.

Moreover, two kind of confidences reported here for ease of reference, and defined in section 2.1.3, have been analyzed

$$c_{wkNN_1} = 1 - \frac{z(q, c_{w_2})}{z(q, c_w)} \quad (5.6)$$

$$c_{wkNN_2} = \frac{z(q, c_w)}{k} \quad (5.7)$$

The confidence expressed by c_{wkNN_1} and c_{wkNN_2} assume infinite values in the $[0, 1]$ interval. Due to this, the ties probability in *weighted kNN* is very low. c_{wkNN_1} is a measure of confidence of the incertitude the classifier has about the first and the second class with higher score among the others. c_{wkNN_2} instead, conveys the probability that q belongs to the class with absolute higher score. Of course, for face recognition purposes the former matters. The k_{opt} depends

upon the size of the TS. For a small TS, a small value of k gives better performances. If TS grows, the best k grows as well, until it reaches a maximum value of 20. Since the accuracy obtained by these k values is quite the same (see Table 5.2), we analyze only how the accuracy and the number of classified elements change if $k_{opt} = 20$, when the confidence c_{wkNN_1} grows from 0 to 1 (see Figure 5.6), even if for efficiency reasons $k = 7$ should be chosen. Whereas, both the accuracy obtained and the number of classified elements are considered in Figure 5.7 for training set sizes of 5, 15, 40 and 50 samples.

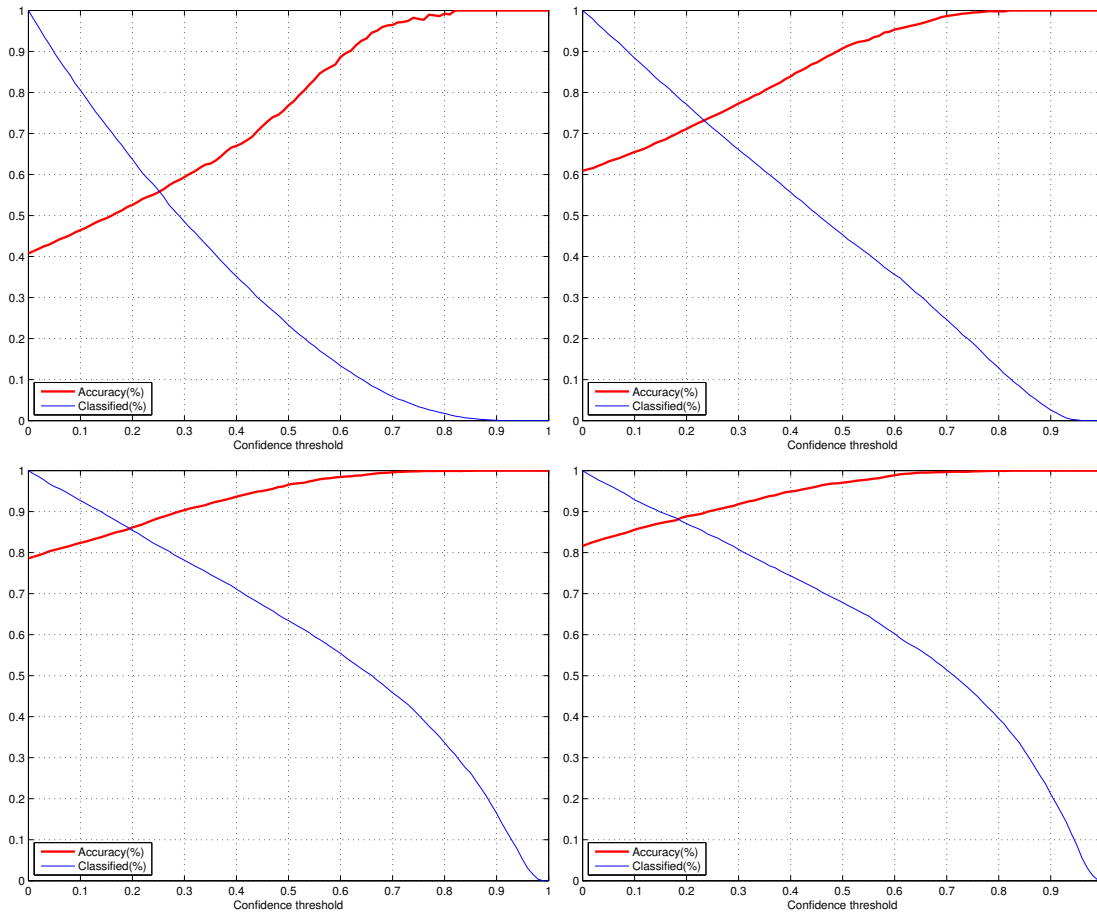


Figure 5.6

Results of weighted kNN when $k_{opt} = 20$ and the training set size increases, considering confidence c_{wkNN_1} . Again, the higher is the required confidence, the higher is the reached accuracy but the lower is the number of classified elements, when the training set size is fixed. If the latter grows, a higher percentage of elements can be classified whit higher accuracy, for a fixed confidence value.

We have seen that 1NN classifier is preferable than kNN . If we compare Figure 5.7 with Figure 5.3, the former classifier is able to classify more objects for a given accuracy and a fixed training set size. For instance, the weighted kNN with 50 samples per class in the training set, is able to classify roughly 60% of faces whereas the 1NN with the same configuration only 40%. Hence, the weighted schema allows us to classify better than the other classifiers.

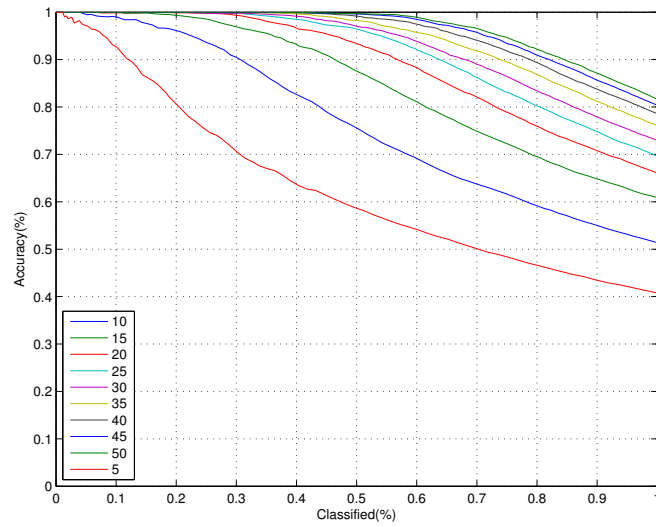


Figure 5.7

weighted kNN for $k = 20$ (Accuracy(%) and Classified(%)) and the training set size grows. For each curve, the number of training set samples used is reported in the legend (bottom-left of the plot).

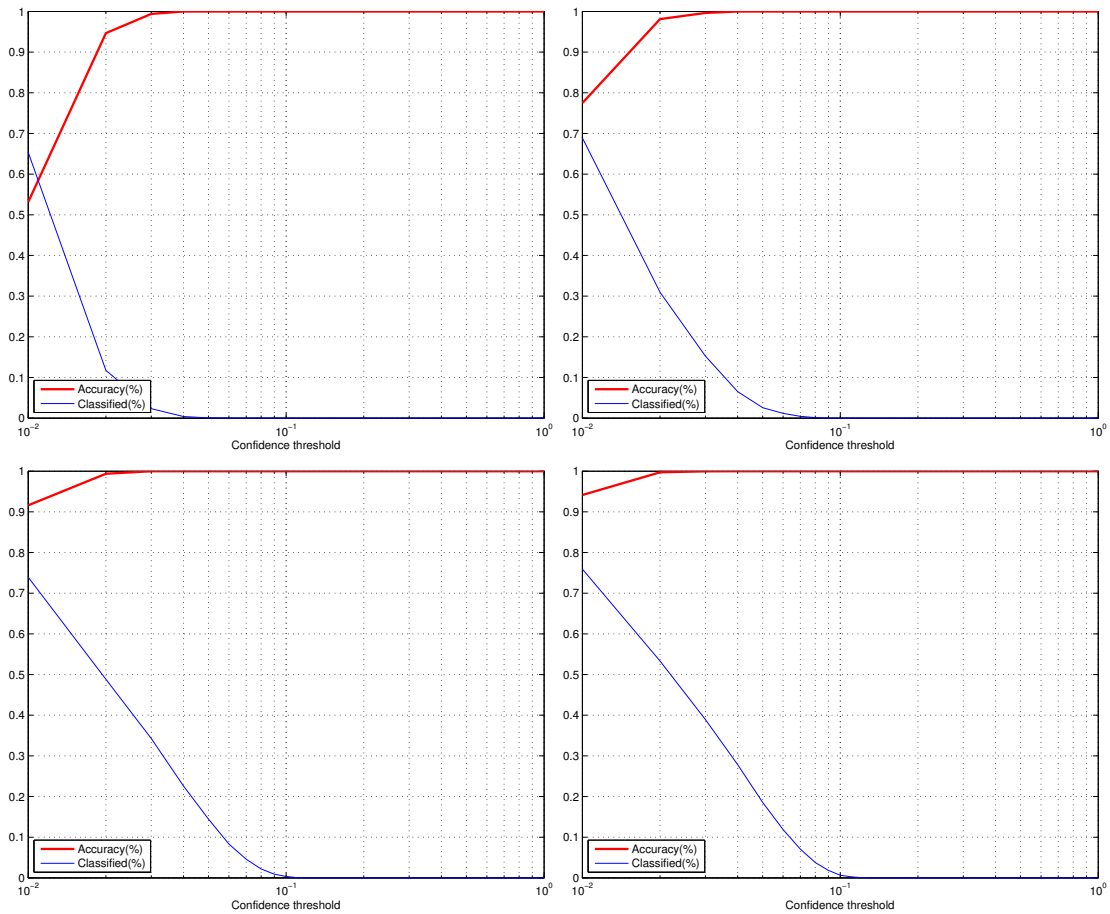


Figure 5.8

weighted kNN results for $k = 20$, considering c_{wkNN_2} confidence and different TS sizes. Plots are in semi-logarithmic scale since for a low confidence a high accuracy is reached, whereas the number of classified faces rapidly decreases.

Finally, Figure 5.8 shows how the accuracy and the classified elements change when c_{wkNN_2} varies for the same different training set sizes of the previous plot 5.6. Notice that the meaning of $c_{wkNN_2} = 1$ is: *all objects of the response set belonging to the winner class c_w are at distance $d()$ equal zero from the query*. Plots are reported in semi-logarithmic scale, since for a low confidence there is already a high accuracy. When the algorithm run, the confidences c_{wkNN_1} and c_{wkNN_2} can be combined in order to have a higher confidence of the response of the classifier.

5.2.2 PCA on LBPH feature

Evaluating the LBPH feature on a 100×100 pixels image is an easy task also for an embedded node (Table 5.1). However, the storage used for the TS and the time spent to evaluate the χ^2 distance can still be high for the feature length we got. We would like to be fast without loose prediction power at node side. In this section, the results obtained by the combination of the LBPH feature and PCA will be shown. PCA allows to reduce the feature length looking at the components in the feature space with higher variance.

Let us call n the number of samples of the training set of faces. For each sample we consider the LBPH feature F_i that is an histogram of dimension d , so $F_i \in \mathfrak{R}^d$. The mean value μ of these features is

$$\mu = \frac{\sum_i F_i}{n} \quad (5.8)$$

Now, the covariance matrix C is

$$C = \frac{1}{n} \cdot \sum_{i=1}^n (F_i - \mu) \cdot (F_i - \mu)^T \quad (5.9)$$

In order to perform PCA the eigenvectors of the covariance matrix must be evaluated. It can be shown that the maximum number of principal components that can be extracted from C can be evaluated as the minimum between the number of samples used to build the model n and the original number of dimensions of the samples d . For example, if each sample has size 10000 and the number of samples is 1200, the principal components that can be retained are at most 1200. So, in the following, the number of principal components has been fixed to 256. In this way we can compare the different results for increasing training set sizes that start from 15 samples per class.

The Euclidean norm, and no longer the chi-square distance, is now used as dissimilarity function for features comparisons. Our intuition is that the high lost in effectiveness is mainly related to the change in the distance function.

1NN

The results reported in this subsection can be compared with the ones in subsection 5.2.1. This approach can be used if 100% accuracy is not required and misclassification can be acceptable. Figure 5.9 shows how the accuracy and the classified percentage varies when we enlarge the TS. For a threshold higher than 0.3, the accuracy starts to decrease whereas the number of classified faces increases. For thresholds higher than 1.3, over than the 90% of faces are classified and the accuracy is constant. Finally, Figure 5.10 relates the percentage of classified faces and the accuracy obtained, while increasing TS. A higher number of training set samples allows to get high accuracy but, unfortunately, it never reaches 1.

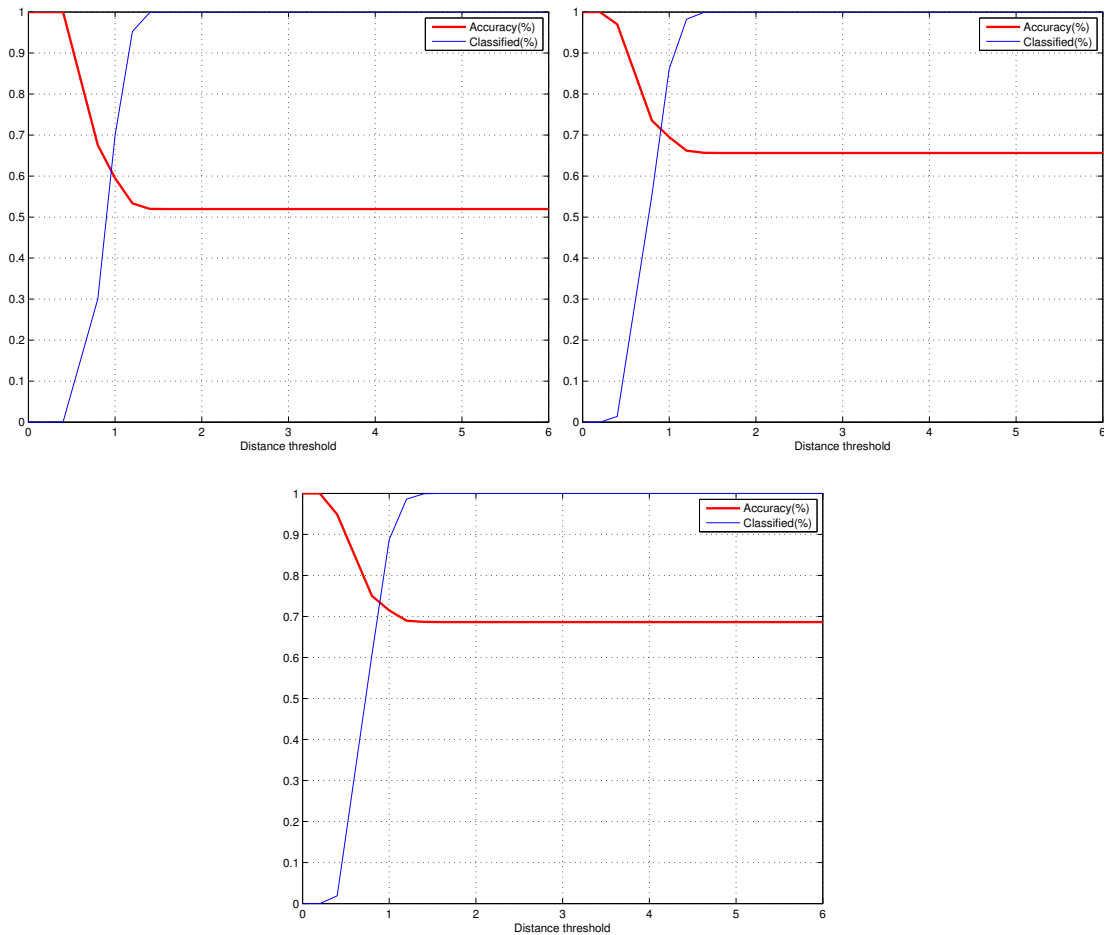


Figure 5.9

1NN combined with PCA feature reduction when varying the threshold t and the number of samples per class in the training set (15, 40, 50).

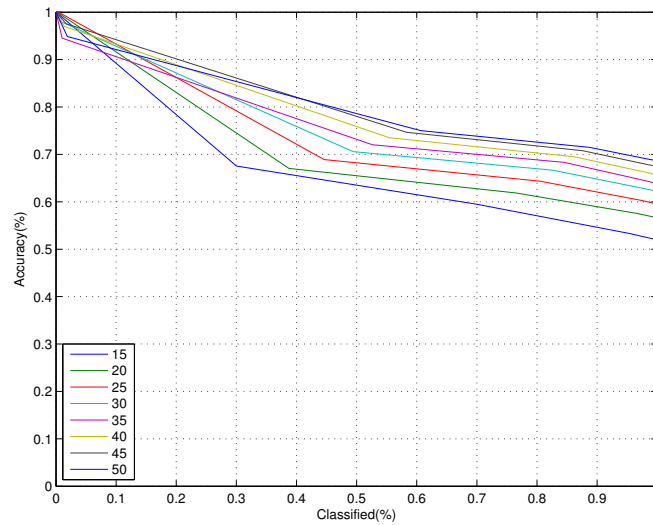


Figure 5.10

1NN combined with PCA (Accuracy(%) vs Classified(%)) when the number of training set samples per class increases (this number is reported in the bottom-left legend). The 100% of accuracy is never reached.

kNN

The same considerations reported in subsection 5.2.1 hold here. When PCA is used, the k_{opt} , while increasing TS, is equal to 2. We have seen that kNN without PCA is not able to reach 100% of accuracy so we do not expect that after PCA reduction kNN is able to get higher accuracy. Figure 5.11 shows how accuracy and the percentage of classified faces are related.

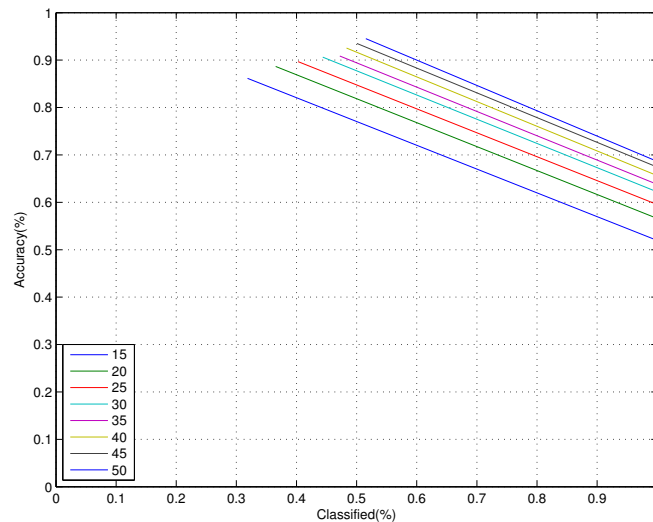


Figure 5.11

kNN algorithm when PCA is applied on LBPH feature (Accuracy(%) vs Classified(%)) and the training set size grows. The legend reports for each curve the number of used samples. The 100% of accuracy is never reached.

Weighted kNN

The weighted variant of kNN is still able to reach good results, despite the heavy space reduction. As in the case in which no feature reduction is applied, the k_{opt} value changes when the TS is enlarged with new samples. Let us consider how accuracy and the percentage of classified elements changes if $k_{opt} = 10$ is chosen (the variation of the accuracy for k equals 5, 7, 10 is very low). Figure 5.12 and Figure 5.13 report those results and they can be compared with the ones in subsection 5.2.1. Even though high accuracy can be reached, the number of classified elements when it is near 100% is very low. In the previous case (weighted kNN on LBPH in section 5.2.1), when 100% accuracy is required, the number of classified faces is near the 70%.

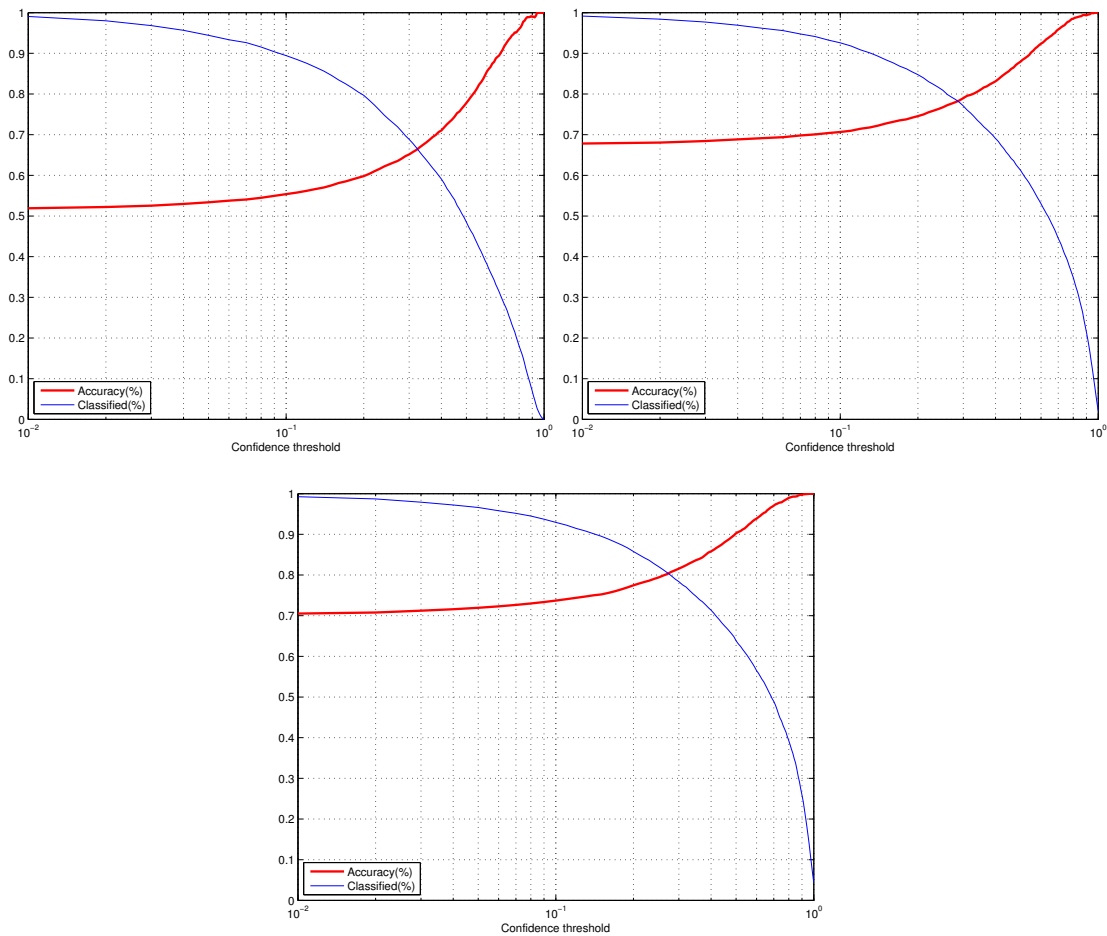


Figure 5.12

weighted kNN combined with PCA feature reduction, varying confidence for $k = 10$ and training set size

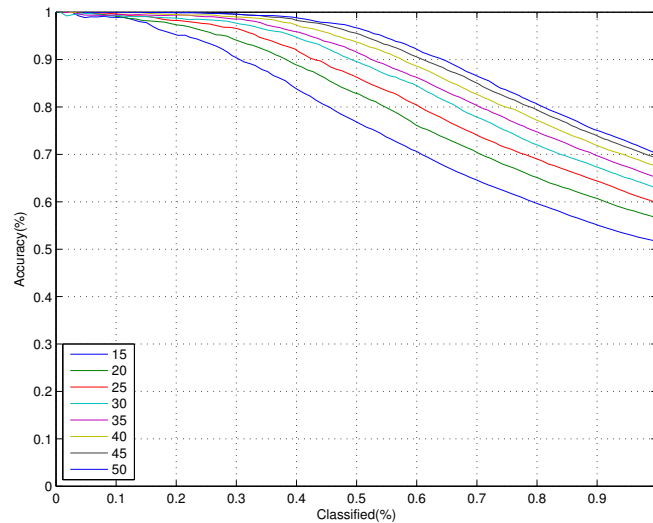


Figure 5.13
weighted kNN combined with PCA (Accuracy(%) vs Classified(%)), for $k = 10$

5.2.3 Cross-validation

In the previous sections we have tested different classifiers using the LBPH feature and a combination of PCA and LBPH on the Yale Dataset. We have tested them for different training set sizes and we have seen that the best result is given by the weighted kNN without PCA reduction, when 50 samples per class are used as training set. Now we can try to assess if this result generalize using a validation technique known as k-fold cross-validation.

Cross-validation is a model validation technique for assessing how the results of a statistical analysis will generalize to an independent data set. It is mainly used to assess how a predictive model will perform in practice. Moreover, it reduces problems such as *overfitting* of the model. One round of cross-validation involves partitioning a sample data into complementary subsets, choosing among these a subset as training set and validating the analysis on the other subset (called the *validation set* or *testing set*). In k-fold cross-validation, the set is initially divided into k equal sized subsets and at each iteration one of them is chosen as training set and the others as validation set.

Cross-validation, in the best case we have determined, can be performed in the following way:

- randomly organize 350 images per person;
- divide them in 7 complementary subsets (in this way 50 images per person can be used as training set at each round, the rest as *validation set*);
- evaluate the accuracy at each round;

- the final accuracy is an average of the ones obtained during the performed rounds.

Figure 5.14 shows the average for the accuracy and the classified percentage, for different values of confidence c_{wkNN_1} . Deviations from the mean values are shown through bars above the curves.

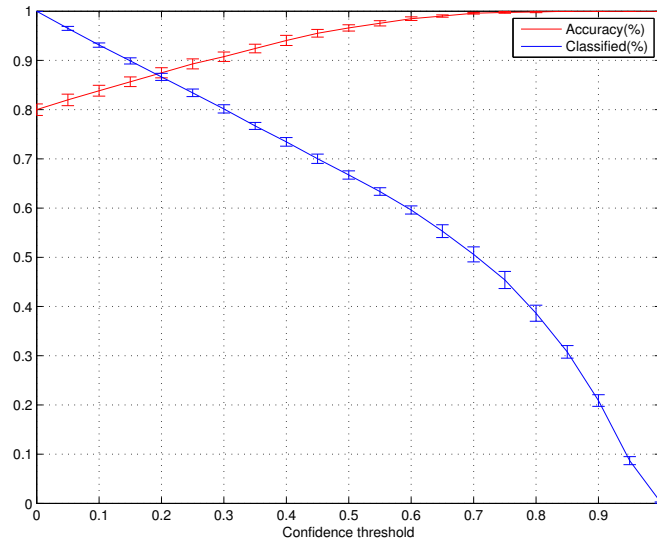


Figure 5.14

Cross validation for weighted kNN. Deviation for accuracy and the percentage of classified elements is shown through the error bars on the curves.

5.3 Simulating a network of cameras

A network of cameras has been simulated with the aim of evaluating the goodness of our recognition system and how its scalability is affected by a) the size of the cache nodes maintain b) the way nodes exchange these classifiers and c) the classification algorithm used at node side. Standard systems rely on remote servers in order to process and analyze videos or frames taken by cameras for surveillance purposes. This approach requires an underneath network with high bandwidth in order to process data in real time. In this system, instead, remote requests are performed only when a node is not able to recognize a face, allowing to reduce network traffic. We evaluate different and opposite strategies related to both caching mechanism and the way nodes exchange information, specifically:

1. nodes do not have cache and do not perform recognition, so the server is queried for each new detected face and no information is exchanged among nodes since they would not be able to store it;
2. nodes have fixed size (in term of classifiers) cache and when a node, directly or through remote request to the server, identifies someone, this information is spread to the overall network of cameras;

3. nodes have fixed size (in term of classifiers) cache and when a node (directly or through remote request to the server) identifies someone, this information is spread to the nearest neighbors of this node.

It is worth to note that the first strategy is a special case of the second and third ones, so it will be reported in the two subsections related to both of them.

As previously stated, each node has an internal cache handled according to the LRU (Least Recently Used) policy, hence if this cache is full and a new element must be added, the oldest element is removed to make room. The elements of this cache are classifiers of people made of LBPH features of their faces. Even though a portion of a classifier can be used for prediction at client side, in this simulation nodes will use the entire person's classifier as the server does. The content of this cache is the same for each node when the simulation starts. We have also assumed that each node already has a list of its neighbors so that no discover phase is needed.

A real network of cameras can have every possible kind of structure, so it is difficult to say which one is preferable or harder to simulate. In this simulation, the network is supposed to be a grid of 8×8 cameras on which 28 people are randomly placed. The system actually knows only 24 people so that 4 of them are really unknown. Cameras' points of views do not overlap each other but the whole grid area is covered by them, so that a person is seen by only a camera at time. Initially, for each person a random start point and a random end point are chosen. During the simulation, people move towards their final destinations by one step at time. Once they arrive, a new end point is chosen. This operation is performed ten times for each person. Of course, while people move, cameras try to recognize them, alert their neighbors, perform requests to the server and so on. The probability that a person remains in the same position has been chosen high in order to simulate the fact that when a person's face is captured by a camera, this will be able to detect his/her face in several sequential frames.

In the previous sections, it has been proved that good results are obtained by the weighted kNN classifier. During simulation, the server will always use this algorithm whereas 1NN, weighted kNN and a combination of 1NN and weighted kNN classifiers could run on cameras. The latter (combination of) classifier has been tested, since 1NN is more robust than weighted kNN for small cache sizes, whereas the latter performs better for higher cache sizes. When a new face must be labelled, the weighted kNN is first applied, then we check if 1NN returns the same label proposed by the weighted kNN. If it is so, this label is given to the query. When weighted kNN is used, a threshold on both confidences c_{wkNN_1} , c_{kwNN_2} (equations 2.30, 2.31) is applied, otherwise the algorithm won't be able to correctly classify faces if cache size is too small. In fact, in such a case the classifier gives high scores also to the wrong classes. Finally, note that

server runs with higher confidence to classify faces than clients, although neither on server nor on client side the training set is incremented with new recognized samples of a face, but it is needed to avoid the injection of wrong information on the network.

The following measures are extracted after these simulations:

1. Server
 - (a) number of classification requests received;
 - (b) number of classified faces and correctly classified faces;
 - (c) number of sent classifiers to clients;
2. Client (averaged values)
 - (a) classification requests sent to the server;
 - (b) alert messages and total number of classifiers sent to neighbors;
 - (c) total number of classifiers received;
 - (d) number of used classifiers (through which someone has been recognized);
 - (e) number of unused classifiers;
 - (f) number of tried recognitions and number of correct recognitions;
 - (g) request of a classifier to the server already present in cache (for debug purpose).

All these parameters are reported according to the cache size of clients. Worst results are expected from the situation in which cameras advise the entire network.

5.3.1 Alerting the whole network

In this situation, a node sends an alert message containing the id of the person it has recognized to the whole camera network. Those cameras can require this classifier later, if they do not have it. The classification algorithm that runs on smart nodes affect the above values, as we are going to see.

The average *number of requests* to the server coming by clients is supposed to become smaller when the number of elements in the caches grows. This actually happens irrespectively of the classifier that is used. Anyway, if the weighted kNN is applied on clients, this number is quite small for reduced cache sizes, since the error committed by this classifier is higher than the one committed by 1NN in such a case. That is, the new incoming face is often labelled with one of the classes available in cache and no request is made to the server, although thresholds on both confidences c_{wkNN_1} and c_{kwNN_2} are used. Due to this, sequential *alarm messages* through

which a node informs that it has seen person X , will contain the same id. Since it is unlikely that after few seconds a neighbor will require the same classifier again, a simple approach to reduce such a flooding of messages is to check if the last message sent contained the same id of the one that is going to be sent, avoiding useless communication. The *number of sent classifiers* by each node decreases when the cache size grows, since the probability that a neighbor already has that classifier is higher. For low capacity of the cache, this number depends upon the classifier: if 1NN is used a lot of classifiers are exchanged, whereas if weighted kNN is applied, the algorithm produces a lot of misclassification and a low percentage of classifier is sent (since the initial content of these caches is the same for each client). So, the *number of classifiers received* by nodes sent by neighbors and the server tends to diminish for higher cache dimensions, whereas the number of actually *used classifier* grows. However, due to the fact that a node contacts the entire camera network, when cache size is small, a lot of classifiers are exchanged even if they won't be needed. From the face recognition performance point of view, it is better to consider clients and server separately. The *average number of classifications* to be performed on clients depends only on the simulation and the random walks, whereas the *number of classifications on server* side to be performed, mainly depends on cache sizes of the node and the used classifier. The number of classifications on server side is extremely high when nodes have no cache (this matches with the first strategy). Anyway, on server side, where the weighted kNN is always applied, due to the higher confidence required for classifying faces the accuracy is very high. On client nodes, when 1NN is used, due to a lower required confidence the accuracy never reaches 1 but it is quite stable and does not depend on cache size. The weighted kNN, instead, reaches high accuracy for bigger cache sizes.

In the following the most meaningful plots are reported in order to compare the case in which 1NN, weighted kNN and their combination is applied as classifier on nodes. When the two algorithms are combined, some of the previous problems such as misclassifications for low cache dimensions are reduced.

In Figure 5.15, the average number of alert messages and classifiers sent by clients are reported. It is interesting to note that the number of classifiers sent decreases when cache size grows. When weighted kNN is applied, no classifiers are sent when cache size is equal to one and few alert messages are sent since it classifies every face as belonging to the person in cache, so sequential alert messages are suppressed or no classifier is needed by cameras (they already have it). The average number of requests performed by clients is reported in Figure 5.16. When 1NN is used, this number decreases since the algorithm is able to find a nearest neighbor (even if wrong) among the classifiers stored in cache. The w-kNN instead roughly performs the same number of requests except for cache size equals to one. This is due to the fact that when the

classifier is confused among two or more classes that achieve high scores, a remote request is performed. In Figure 5.17 the average number of classifiers used, not used and received by nodes are reported. These plots show the drawback of the "alert all" strategy: too many classifiers are received even if they are not needed at all. In fact, for small cache sizes, the percentage of used classifiers with respect the overall classifier is very low. When cache size grows the number of used and unused classifier is roughly the same. Figure 5.18 shows the average performance of the classifier on client. The error committed by 1NN is roughly the same during classification, independently from the cache size. The weighted kNN achieve great accuracy when the number of local classifiers is high but wrong classifications are performed for small cache sizes. The combination of the classifier performs better than the other two independently from the cache size. Finally, in Figure 5.19 the total number of classifications, succeeded classifications and correct classifications performed by the server are reported. The behavior is almost the same in each case: the number of incoming requests and so of the succeeded classifications and correct classifications decreases for higher sizes of the caches. As already state, when weighted kNN is applied and cache size is small, nodes do not perform requests to the server since they erroneously classify the detected face as belonging to the person in cache (plot in the middle, when cache size is equal to one). Finally, the case in which no cache is available at client side is reported too and corresponds to cache size equals to zero.

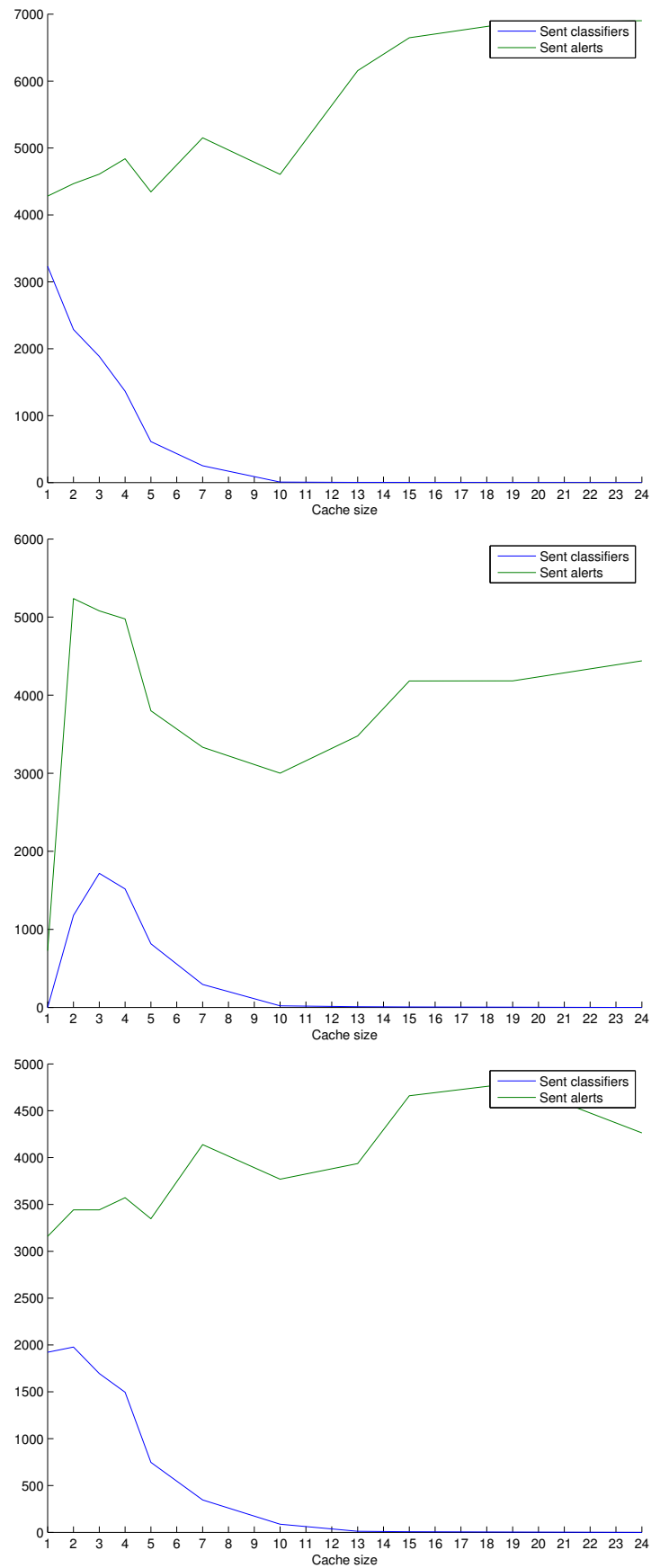
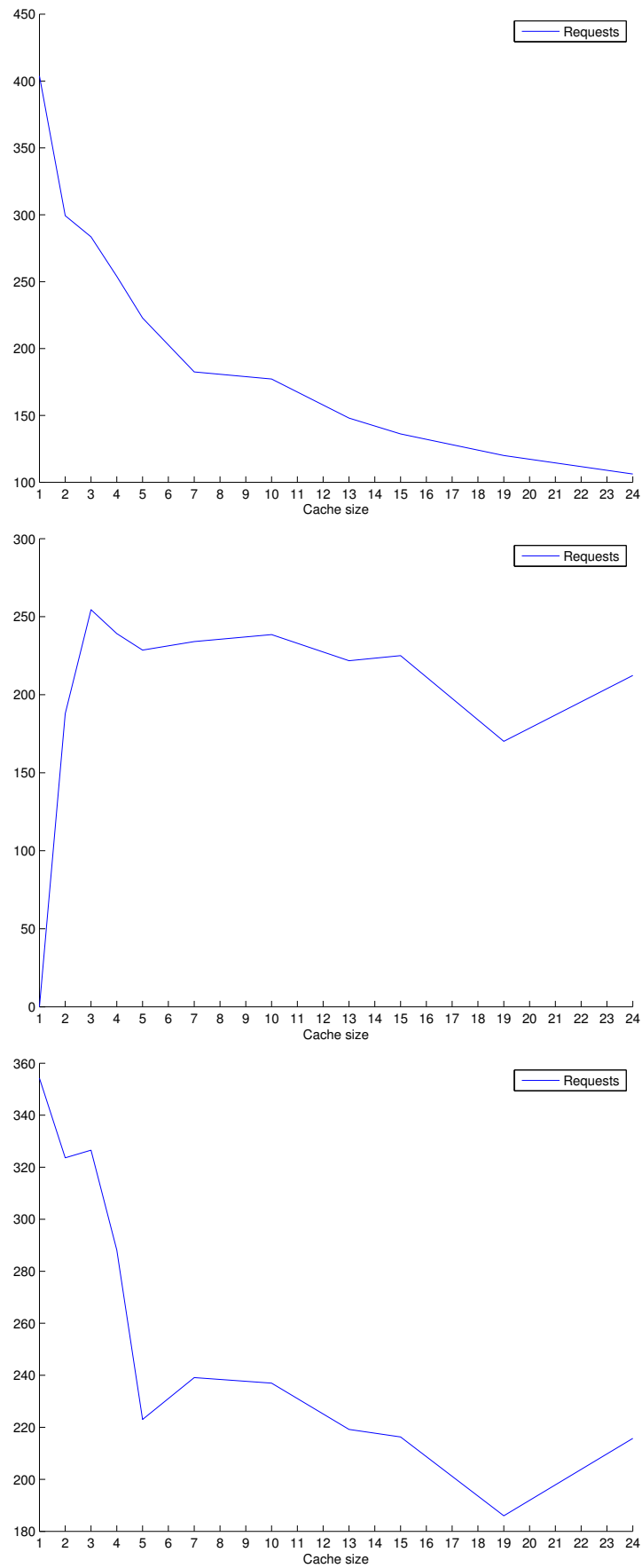


Figure 5.15

Network simulation - alert messages sent to the whole network (1NN top-left, wkNN top-right and 1NN + wkNN bottom): average number of alert messages and classifiers sent by clients.

**Figure 5.16**

Network simulation - alert messages sent to the whole network (1NN top-left, wkNN top-right and 1NN + wkNN bottom): average number of requests performed by clients.

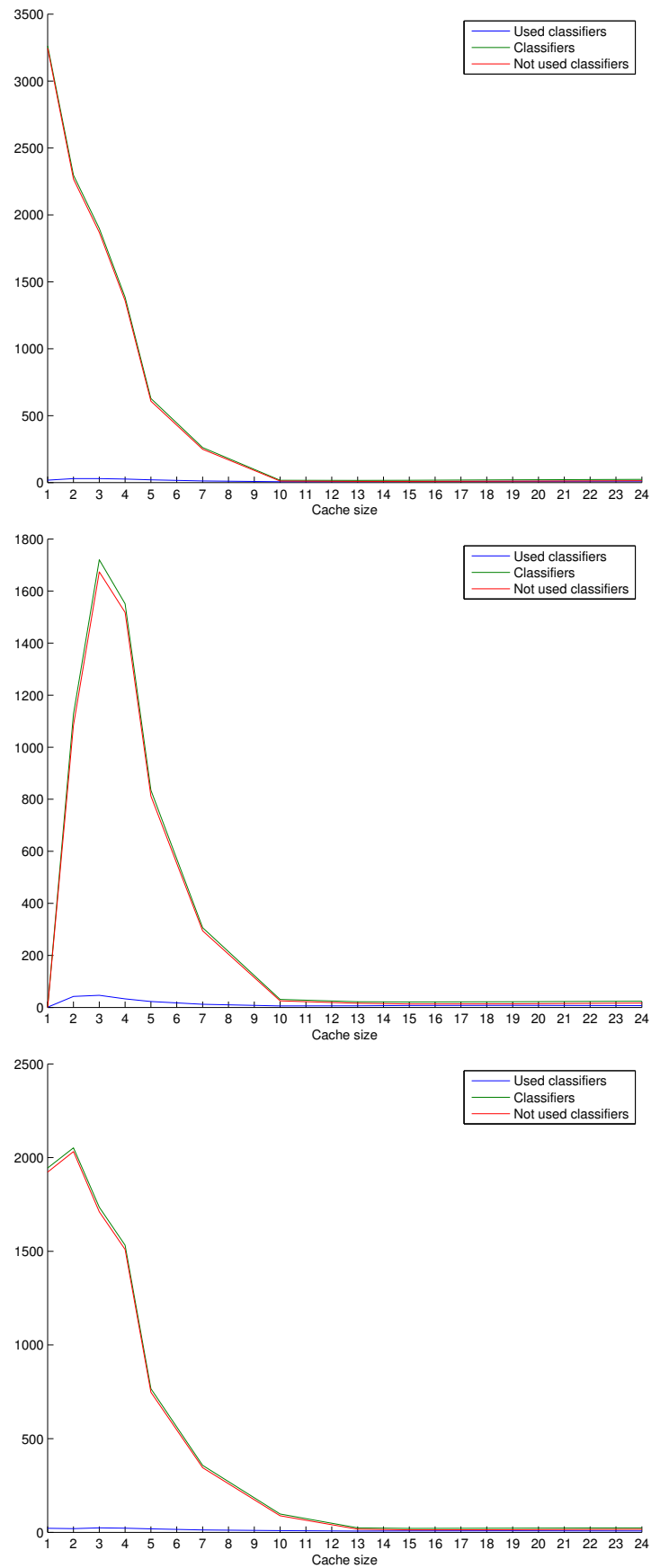
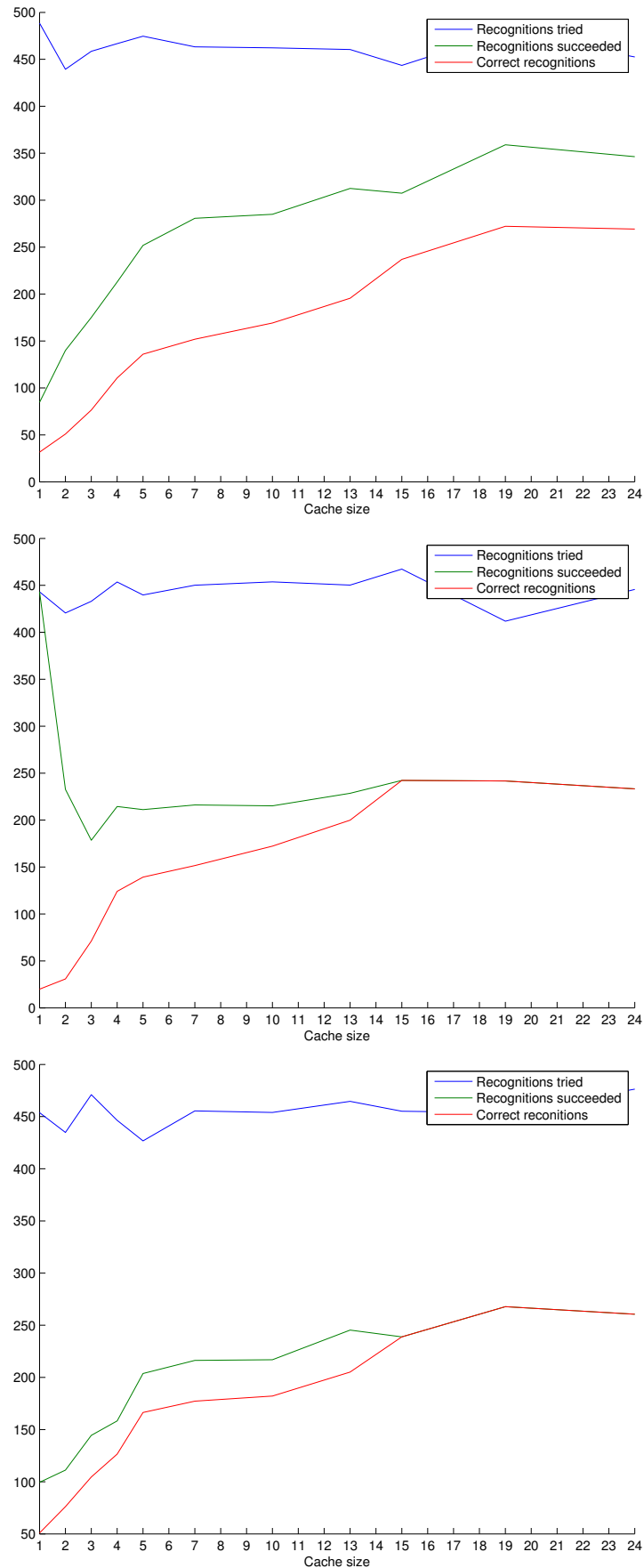


Figure 5.17

Network simulation - alert messages sent to the whole network (1NN top-left, wkNN top-right and 1NN + wkNN bottom): average number of classifiers used, not used and received by nodes.

**Figure 5.18**

Network simulation - alert messages sent to the whole network (1NN top-left, wkNN top-right and 1NN + wkNN bottom): average number of classifications, succeeded classifications and correct classifications performed by clients.

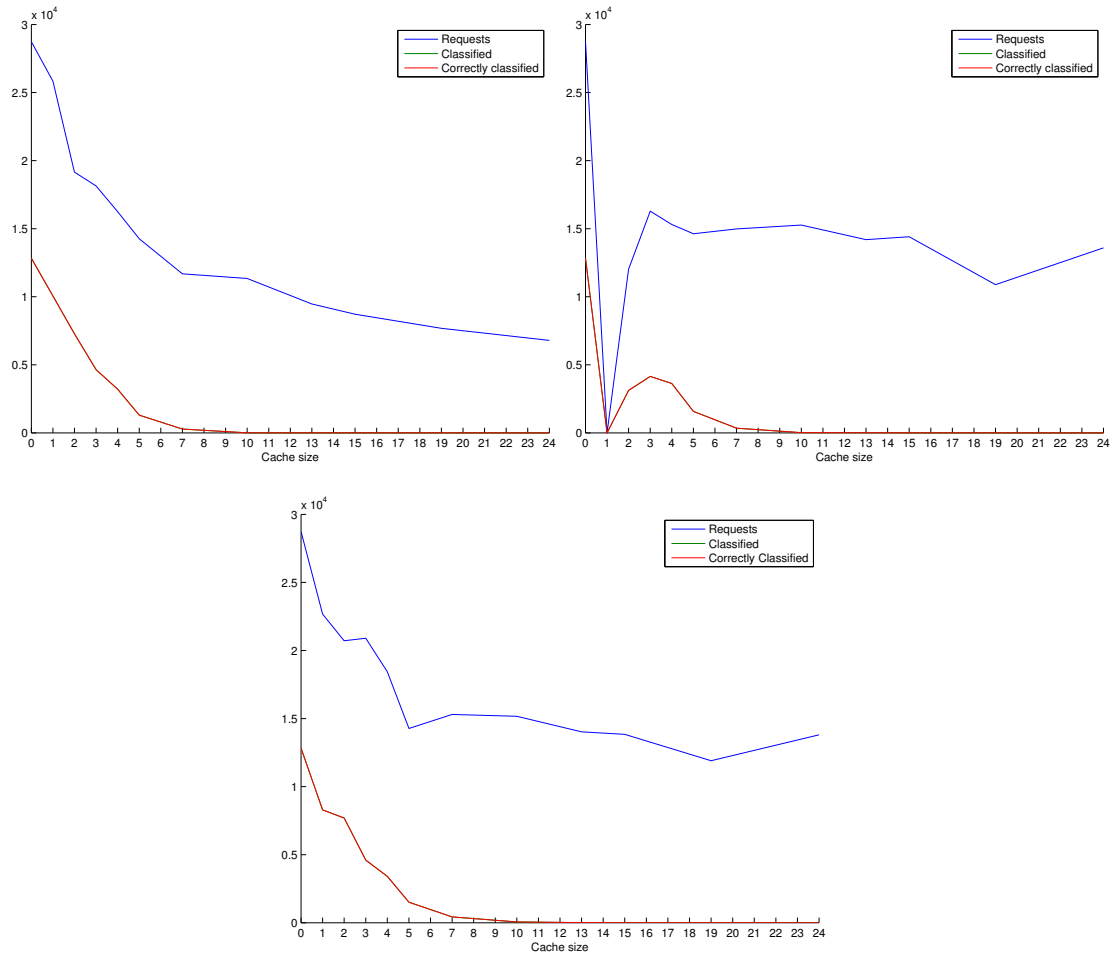


Figure 5.19

Network simulation - alert messages sent to the whole network (1NN top-left, wkNN top-right and 1NN + wkNN bottom): number of classifications, succeeded classifications and correct classifications performed by the server.

5.3.2 Alerting the nearest neighbors

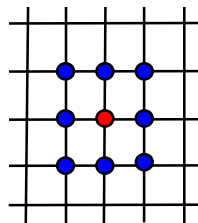


Figure 5.20

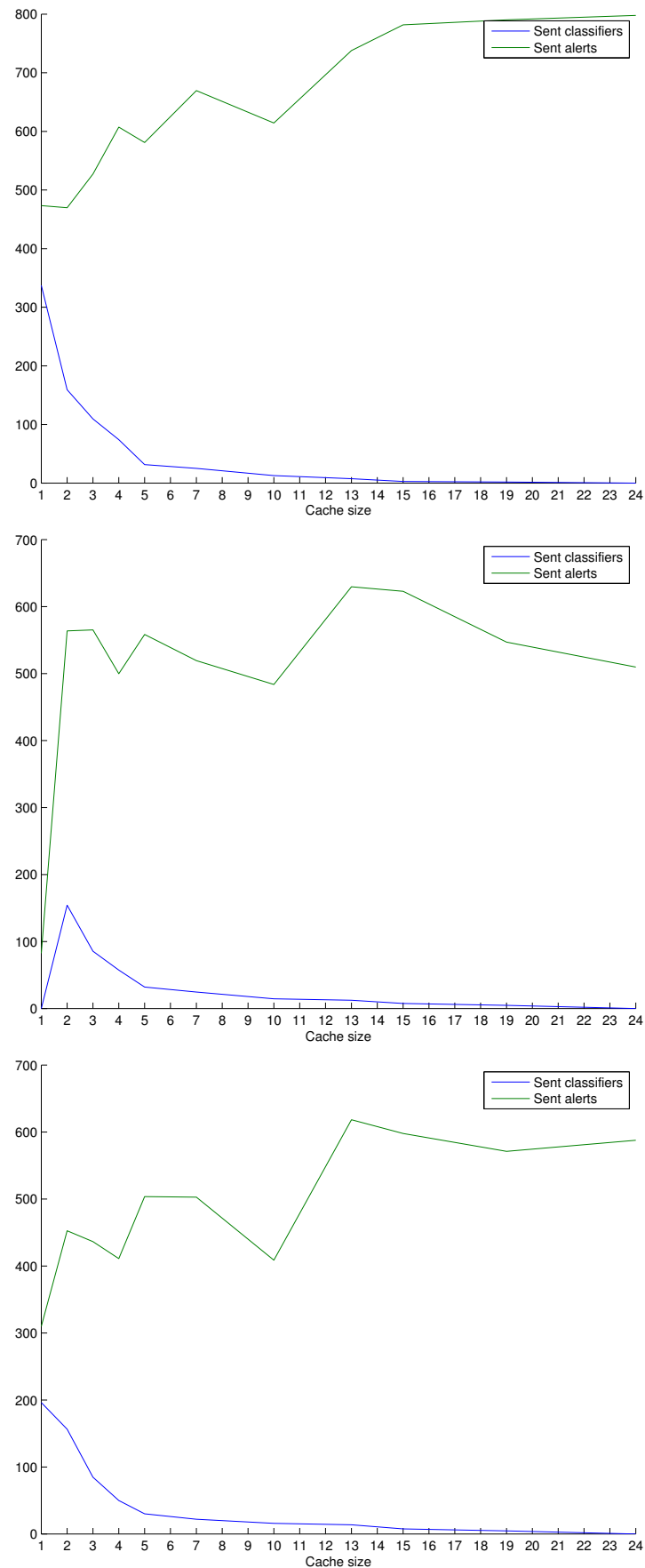
Nearest neighbors of the red camera are reported in blue

Although it is very unlikely that a camera network has this simple structure, it is even more unlikely that a classifier of a person will be needed by all cameras. In fact, the probability that

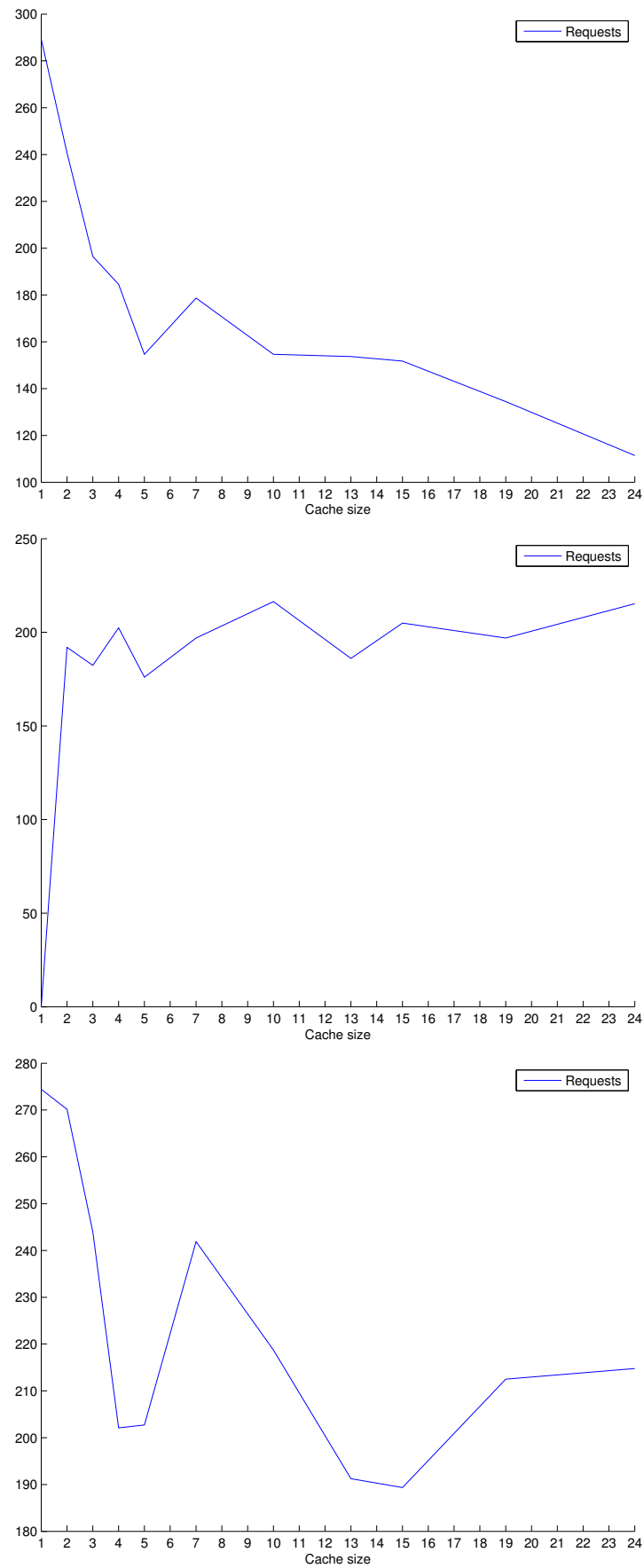
a person will be seen by a camera far away from the one that has just seen him now is very low. Thus, a more reasonable approach would be to advise the nearest cameras that person with id X will probably be detected by them after a while. Figure 5.20 shows the nearest neighbors for a camera in the simulated network.

The same reasoning reported in the previous subsection for the different parameters and their dependences applies here. Intuitively, data to be exchanged are lower with respect the previous situation. Almost all the averaged statistics are affected by this new approach, except the ones relative to the number of recognitions to be performed on client side. In fact, it depends upon the random walks only. In the following the mainly affected average statistics are reported.

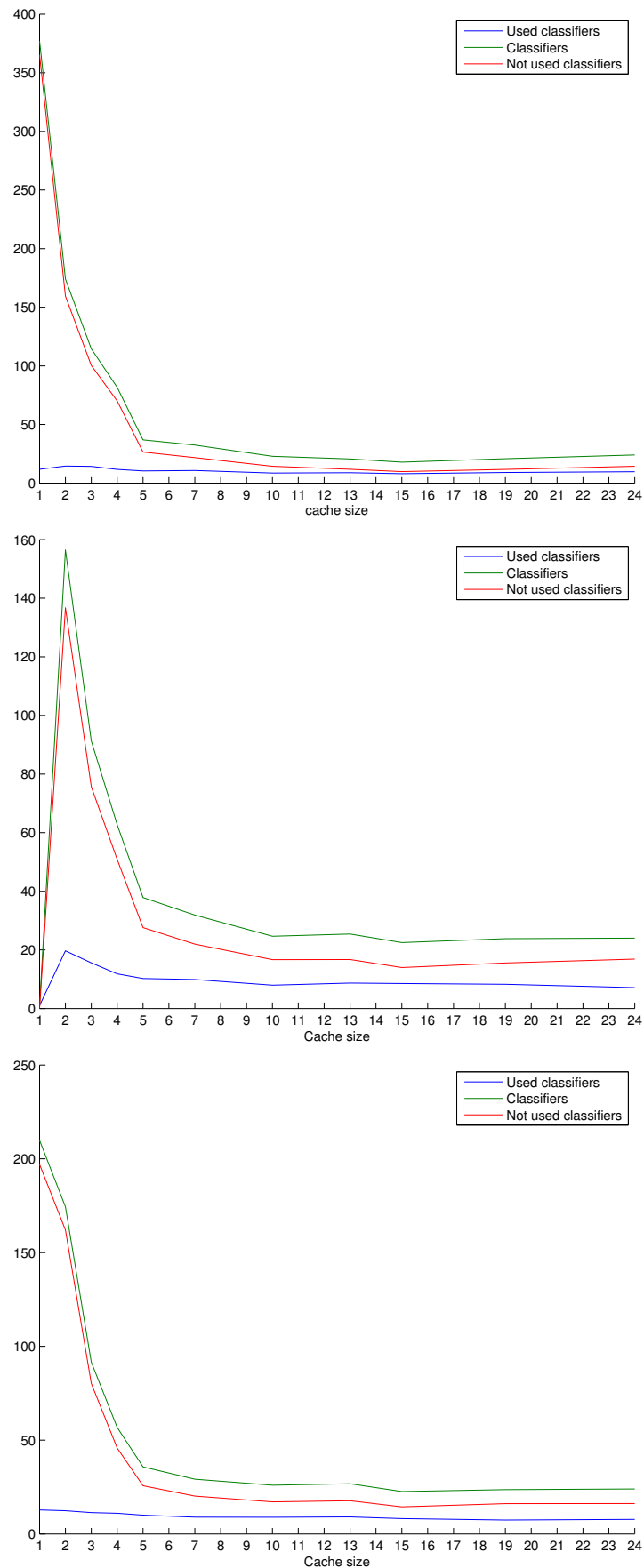
In Figure 5.21 the average number of sent classifiers and alert messages sent by nodes is reported. Compared to the previous approach, both average numbers are reduced by a factor of 10. This makes sense since clients alert only their neighbors. Again, the average number of sent classifiers decreases when cache size grows, due to the fact that the probability that a node already has that classifier grows as well. The averaged number of requests performed by clients, when the different classifiers are used, is reported in Figure 5.22. In this case, when the cache is small, lower requests are performed with respect to the previous strategy, since less useless classifiers are exchanged among nodes and the local classifier performs slightly better, see Figure 5.24. Thus, less remote requests are performed. The total number of received classifiers by nodes, together with the ones that are used or not is reported in Figure 5.23. When the alert-all strategy is used, nodes exchange too many useless classifiers that will leave their cache without being used. The same holds here, for smaller cache size, however, when this number grows, the percentage of used and not used classifiers is comparable. This aspect is important, since nodes' memory isn't wasted. A cache size of (at least) 5 classifiers must be used. Finally, in Figure 5.25 the number of classification requests done to the server, together with the succeeded classifications and the correct ones is reported. Less requests reach the server than the previous case, since classifiers on nodes perform better.

**Figure 5.21**

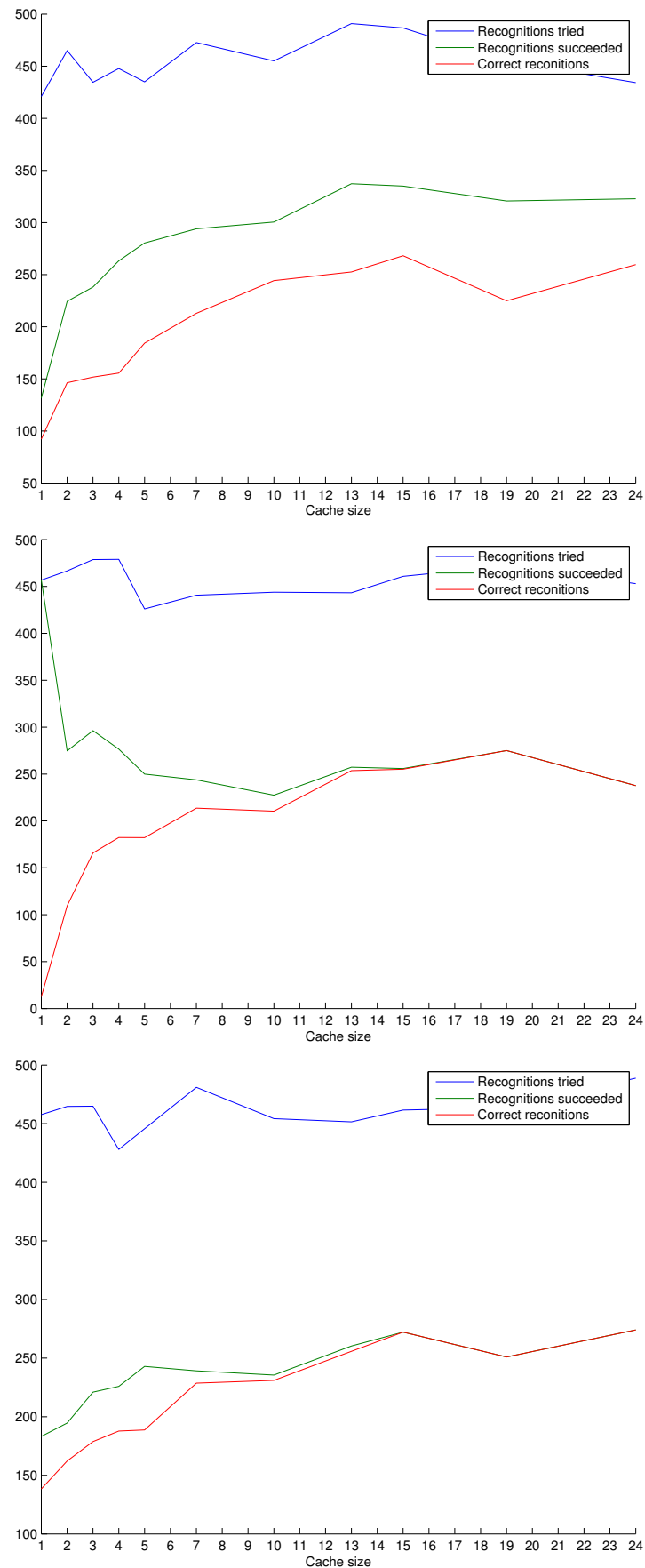
Network simulation - alert messages sent to the nearest neighbors (1NN top-left, wkNN top-right and 1NN + wkNN bottom) average number of alert messages and classifiers sent.

**Figure 5.22**

Network simulation - alert messages sent to the nearest neighbors (1NN top-left, wkNN top-right and 1NN + wkNN bottom): average number of requests performed by clients.

**Figure 5.23**

Network simulation - alert messages sent to the nearest neighbors (1NN top-left, wkNN top-right and 1NN + wkNN bottom): average number of classifiers used, not used and received by nodes.

**Figure 5.24**

Network simulation - alert messages sent to the nearest neighbors (1NN top-left, wkNN top-right and 1NN + wkNN bottom): average number of classifications, succeeded classifications and correct classifications performed by clients.

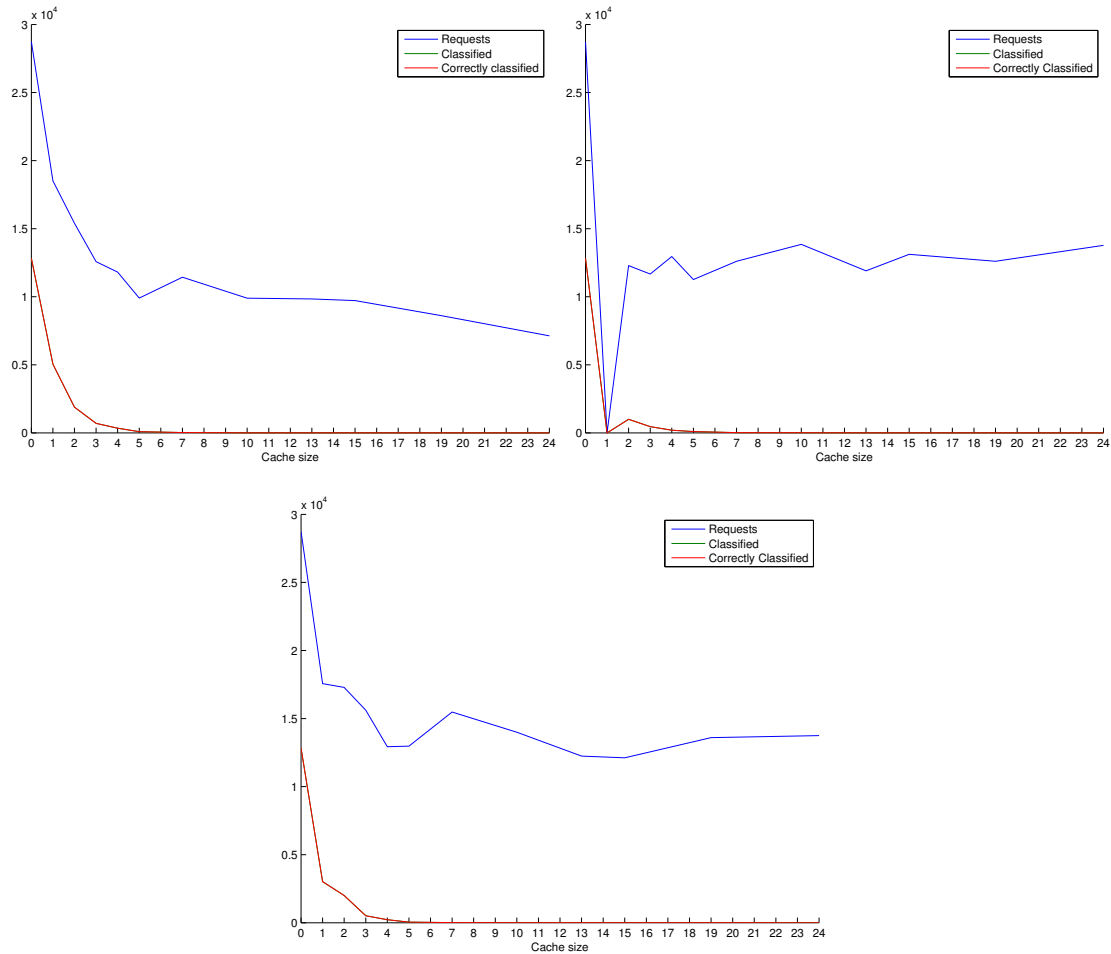


Figure 5.25

Network simulation - alert messages sent to the nearest neighbors (1NN top-left, wkNN top-right and 1NN + wkNN bottom): number of classifications, succeeded classifications and correct classifications performed by the server.

Cache mechanism for unrecognized people

Looking at Figure 5.25, the number of requests made by clients is still high and slowly decreases when the cache size increases. These requests can be further reduced considering that when the server is not able to recognize someone, it is useless to ask again a classification for faces similar to this one. A new cache can be introduced at client side to contain the last unrecognized people by the server. Again, the cache is handled according to the LRU policy. Thus, faces unrecognized by the server can be placed here and when a new one is detected by a node, it tries to directly classify the face with the information it has available. If it is not able to do so, it checks if a similar one (1NN algorithm) is in the cache of unknown people. If it is so, no request is sent to the server. Since on server side the classifiers could be potentially updated with new information coming by other cameras, it is better to discard too old elements. Thus, clients perform a request even if a similar face is in cache but it has been seen a long time ago. The number of requests on server side is greatly reduced, see Figure 5.26 in which the

cache containing the classifiers has been fixed to five. Only the cases in which 1NN and the weighted kNN are applied as classifiers on node side have been reported. An unknown cache of size equals to 2 is enough to reduce a lot the number of remote requests performed by a client and there is no advantage in maintaining a larger cache, since the number of remote requests performed is almost the same.

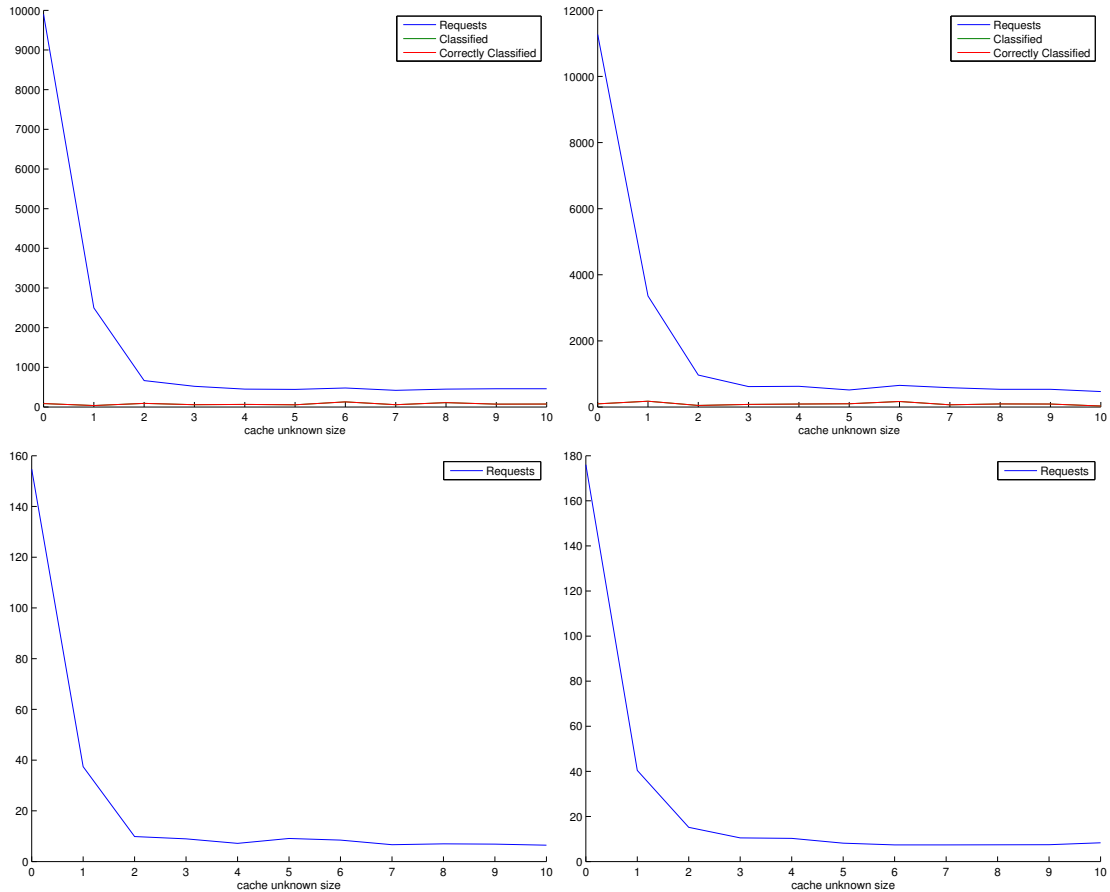


Figure 5.26

Network simulation - alert messages sent to nearest neighbors (1NN on the left, weighted kNN on the right): cache of unknown people by the server greatly affects the number of requests made by clients to the server: (a) number of requests to the server and number of classified objects by the server (b) average number of requests made by a client.

5.4 Exploiting unrecognized faces

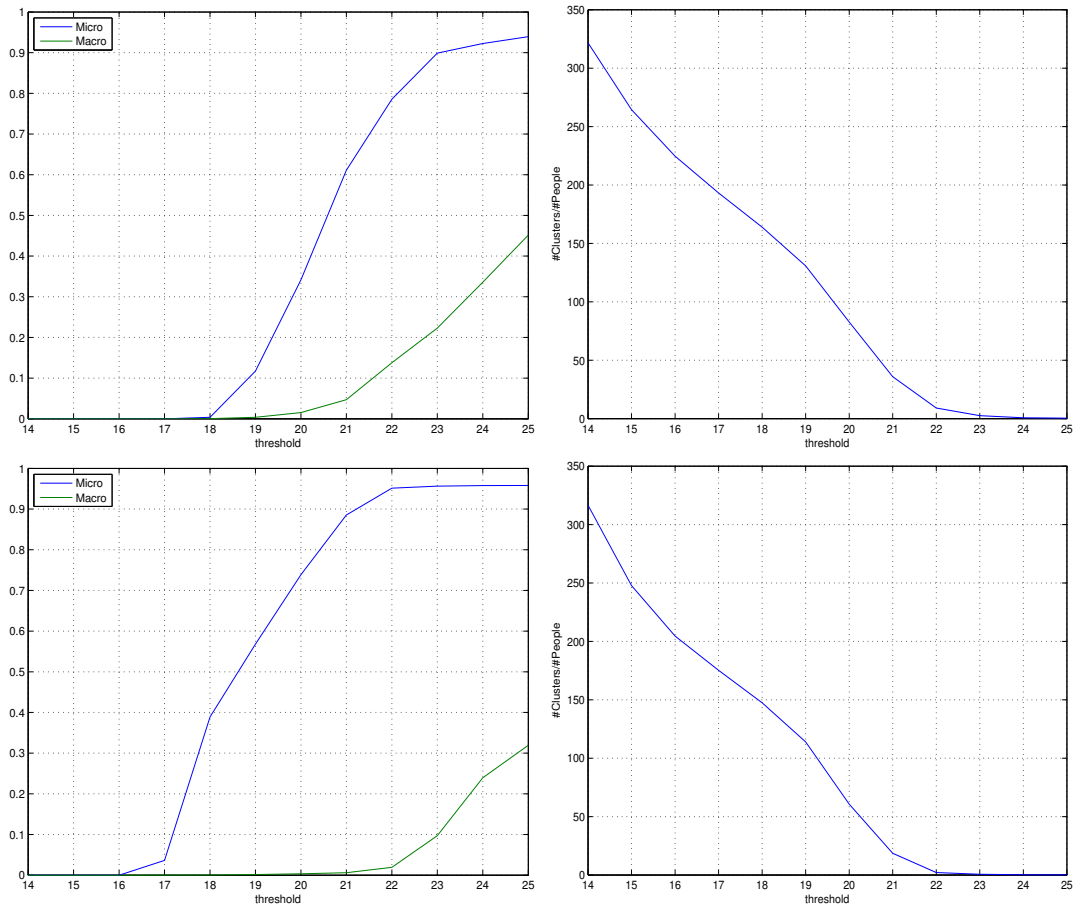
In chapter 4 two clustering algorithms that rely on the LBPH feature for collecting human faces that the system is not able to classify have been described. The first algorithm, tries to assign each new incoming unknown face to the nearest cluster. The distance among an object o_i and a cluster C_j has been defined as

$$d(o_i, C_j) = \min_x \chi^2(o_i, o_x), o_x \in C_j \quad (5.10)$$

Where $\chi^2(\cdot)$ still denotes the Chisquare distance, since we are comparing LBPH feature extracted from face images. If the minimum distance is higher than a given threshold t , than the creation of a new cluster containing the object o_i is forced. The second algorithm instead, tries to exploit a merge mechanism. In fact, a new incoming face can be the linkage of two or more existing clusters of the same person whose face has been captured under different light conditions or poses. Again, a threshold t is used to understand which clusters can be merged. If none of them is eligible, then a new one is created in order to contain o_i .

The evaluation of the algorithms is based on three metrics: the micro and macro average errors defined in chapter 4 and the ratio $\frac{\#Clusters}{\#People}$.

Unfortunately, both algorithms generate an high number of clusters when t has a low value, keeping low the macro and micro average error. For higher values of t instead, the ratio $\frac{\#Clusters}{\#People}$ rapidly decreases but the micro average error is ≈ 0.9 , whereas the macro error remains low due to the effect of the denominator (see equation 4.3). The merge mechanism is even dangerous and this can be seen in Figure 5.27. In fact, for the same threshold t , the micro average error is higher for the second clustering algorithm than the first one. Anyway, also in the first approach, when the threshold t becomes higher, faces belong to different people are unified under a single cluster. One of the reasons is that faces belonging to different people but captured under a similar light condition are no longer distinguished by the algorithms for high value of t . An example of such wrong cluster is reported in 5.28. The LBP operator is robust against monotonic gray changes in the illumination conditions but in real-life scenarios light effects are complex and the previous property of the operator is not enough. In future work different preprocessing schema to achieve illumination normalization while still preserving the necessary information for finding similarity among faces, such the approach proposed in [22] can be exploited.

**Figure 5.27**

Results for the clustering algorithms (from top-left to bottom-right): a) micro and macro average error for Algorithm 1 b) ratio $\frac{\#Clusters}{\#People}$ for Algorithm 1 c) micro and macro average error for Algorithm 2 and d) $\frac{\#Clusters}{\#People}$ for Algorithm 2

**Figure 5.28**

Example of wrong cluster in which four faces captured under the same light condition are put together.

Chapter 6

Conclusions

In this work, a face recognition system for smart camera networks has been presented. The recognition task can be distributed among cameras in order to efficiently reduce the involvement of the remote server and the usage of the network. The software enables smart nodes to detect and recognize human faces in the environment and address common problems that affect face recognition algorithms such as different head poses and illumination conditions. Eyes detection is applied in order to compensate face's rotation. Face description relies on the LBPH feature that is well suited for devices with low computing power due to its simplicity and fast evaluation, although its memory consumption is high. Three classification algorithms have been tested and their accuracy was reported. The weighted kNN was able to reach higher accuracy and to classify a good percentage of the faces. For instance, we achieve more than 90% of accuracy being able to recognize 80% of faces. The LBPH feature has been also used by clustering algorithms implemented and tested in order to help human operator in labelling unrecognized faces stored on server side. Unfortunately they generate too many clusters per person when the error is set to be low. Collaboration strategies among nodes, to reduce as much as possible network load were defined, simulated, and evaluated. If cameras alert any other cameras when they recognize a person, too much alerts and classifiers are exchanged. In fact, a lot of these classifiers are discarded by cameras without being used (no one is recognized through them), resulting in non optimal usage of resources, and more requests to the server are sent. If nodes alert only their nearest neighbors, the number of alerts and sent classifiers is reduced by a factor of 10. When the local cache contains at least 5 classifiers, the best performance were obtained. The weighted kNN classifier combined with the 1NN classifier has better performance than the others. An additional cache is maintained by clients to store faces that the server was not able to recognize. This is used to further reduce communications among nodes. A good size for this cache is 2, through which the average number of requests to the server is reduced by a factor of 8. Larger caches do not allow to further reduce the number of requests performed by

a client significantly, and needs more time to perform the similarity search. Finally, tests on the Raspberry PI platform have shown that five frames per second can be elaborated while face detection and face recognition are applied, and CPU load can be reduced of $\approx 30\%$ through motion detection.

6.1 Future work

The results obtained by this thesis can be further improved along various directions.

A smarter communication protocol could be introduced in order to a) reduce transmitted data through the usage of data compression algorithms b) enable the system to detect anomalies, and c) enable the server to automatically configure the network of cameras, for example determining and informing a new added camera of its neighborhood.

The Raspberry Pi hardware could be further exploited. For example, GPU was not used in this work. However, some of the used algorithms such as face detection can be heavily parallelized. The use of the GPU can accelerate these computer vision algorithms.

Indexing could be exploited in order to speed-up classification time. Currently, each time a new sample must be labelled, the classification algorithm sequentially scans the entire training set in order to find the correct label for the object.

Bibliography

- [1] E. Martinez-Martin and A. P. D. Pobil, *Robust Motion Detection in Real-Life Scenarios*. Springer Briefs in Computer Science, Springer, 2012.
- [2] J. C. Augusto, “Ambient intelligence: the confluence of ubiquitous/pervasive computing and,” *Artificial Intelligence*, pp. 213–234, 2007.
- [3] E. Hjelmås and B. K. Low, “Face detection: A survey,” *Computer Vision and Image Understanding*, vol. 83, pp. 236–274, Sept. 2001.
- [4] P. Viola and M. Jones, “Robust real-time object detection,” in *International Journal of Computer Vision*, 2001.
- [5] T. Ojala, M. Pietikäinen, and D. Harwood, “A comparative study of texture measures with classification based on feature distributions,” *Pattern Recognition*, vol. 29, no. 1, pp. 51–59, 1996.
- [6] L. Zhang, R. Chu, S. Xiang, S. Liao, and S. Z. Li, “Face detection based on multi-block lbp representation,” in *Proceedings of the 2007 International Conference on Advances in Biometrics, ICB’07*, (Berlin, Heidelberg), pp. 11–18, Springer-Verlag, 2007.
- [7] Y. Freund and R. E. Schapire, “A decision-theoretic generalization of on-line learning and an application to boosting,” in *Proceedings of the Second European Conference on Computational Learning Theory, EuroCOLT ’95*, (London, UK, UK), pp. 23–37, Springer-Verlag, 1995.
- [8] M. hsuan Yang, D. Roth, and N. Ahuja, “A snow-based face detector,” in *Advances in Neural Information Processing Systems 12*, pp. 855–861, MIT Press, 2000.
- [9] M. Viola, M. J. Jones, and P. Viola, “Fast multi-view face detection,” in *Proc. of Computer Vision and Pattern Recognition*, 2003.
- [10] M. Turk and A. Pentland, “Eigenfaces for recognition,” *J. Cognitive Neuroscience*, vol. 3, pp. 71–86, Jan. 1991.

-
- [11] K. Etemad and R. Chellappa, “Discriminant analysis for recognition of human face images (invited paper).,” in *AVBPA* (J. Bigun, G. Chollet, and G. Borgefors, eds.), vol. 1206 of *Lecture Notes in Computer Science*, pp. 127–142, Springer, 1997.
- [12] L. Wiskott, J.-M. Fellous, N. Kruger, and C. von der Malsburg, “Face recognition by elastic bunch graph matching.,” in *ICIP (1)*, pp. 129–132, 1997.
- [13] Y. Taigman, M. Yang, M. Ranzato, and L. Wolf, “Deepface: Closing the gap to human-level performance in face verification,” in *Proceedings of the 2014 IEEE Conference on Computer Vision and Pattern Recognition, CVPR '14*, (Washington, DC, USA), pp. 1701–1708, IEEE Computer Society, 2014.
- [14] T. Ahonen, A. Hadid, and M. Pietik inen, “Face description with local binary patterns: Application to face recognition.,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 28, no. 12, pp. 2037–2041, 2006.
- [15] P. N. Belhumeur, J. P. Hespanha, and D. J. Kriegman, “Eigenfaces vs. fisherfaces: Recognition using class specific linear projection.,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 19, no. 7, pp. 711–720, 1997.
- [16] C.-H. Chen, J. Favre, G. Kurillo, T. P. Andriacchi, R. Bajcsy, and R. Chellappa, “Camera networks for healthcare, teleimmersion, and surveillance,” *Computer*, vol. 47, pp. 26–36, may 2014.
- [17] K. Abas, C. Porto, and K. Obraczka, “Wireless smart camera networks for the surveillance of public spaces,” *Computer*, vol. 47, pp. 37–44, May 2014.
- [18] N. Gheissari, T. B. Sebastian, and R. Hartley, “Person reidentification using spatiotemporal appearance,” in *Proceedings of the 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition - Volume 2, CVPR '06*, (Washington, DC, USA), pp. 1528–1535, IEEE Computer Society, 2006.
- [19] T. Bouwmans, F. E. Baf, and B. Vachon, “Background modeling using mixture of gaussians for foreground detection: A survey,” in *Recent Patents on Computer Science*, pp. 219–237, 2008.
- [20] R. T. Collins, A. J. Lipton, T. Kanade, H. Fujiyoshi, D. Duggins, Y. Tsin, D. Tolliver, N. Enomoto, O. Hasegawa, P. Burt, and L. Wixson, “A system for video surveillance and monitoring,” 2000.
- [21] T. Kanade, R. Collins, A. Lipton, P. Burt, and L. Wixson, “Advances in cooperative multi-sensor video surveillance,” in *Darpa Image Understanding Workshop*, pp. 3–24, Morgan Kaufmann, November 1998.

-
- [22] X. Tan and B. Triggs, "Preprocessing and feature sets for robust face recognition," *IEEE conference on computer vision and pattern recognition, CVPR*, vol. 7, pp. 1–8, 2007.
- [23] D. S.Anila, "Preprocessing technique for face recognition applications under varying illumination conditions," *Global Journal of Computer Science and Technology Graphics & Vision*, vol. 12, 2012.
- [24] D. L. Baggio, S. Emami, D. M. Escriva, K. Ievgen, N. Mahmood, J. Saragih, and R. Shilkrot, *Mastering OpenCV with Practical Computer Vision Projects*. Packt Publishing, Limited, 2012. recommended: advanced OpenCV project support/examples inc. iOS and Android examples.
- [25] A. S. Georghiades, P. N. Belhumeur, and D. J. Kriegman, "From few to many: Illumination cone models for face recognition under variable lighting and pose," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 23, pp. 643–660, 2001.
- [26] D. Maturana, D. Mery, and A. Soto, "Face recognition with local binary patterns, spatial pyramid histograms and naive bayes nearest neighbor classification," in *Proceedings of the 2009 International Conference of the Chilean Computer Science Society, SCCC '09*, (Washington, DC, USA), pp. 125–132, IEEE Computer Society, 2009.
- [27] D. Nguyen, D. Halupka, P. Aarabi, and A. Sheikholeslami, "Real-time face detection and lip feature extraction using field-programmable gate arrays," *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, vol. 36, no. 4, pp. 902–912, 2006.