PARTIAL LEAST SQUARES FOR FACE HASHING

CÁSSIO ELIAS SANTOS JÚNIOR

PARTIAL LEAST SQUARES FOR FACE HASHING

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação do Instituto de Ciências Exatas da Universidade Federal de Minas Gerais como requisito parcial para a obtenção do grau de Mestre em Ciência da Computação.

Orientador: William Robson Schwartz

Belo Horizonte
Outubro de 2015

CÁSSIO ELIAS SANTOS JÚNIOR

PARTIAL LEAST SQUARES FOR FACE HASHING

Dissertation presented to the Graduate Program in Computer Science of the Universidade Federal de Minas Gerais in partial fulfillment of the requirements for the degree of Master in Computer Science.

Advisor: William Robson Schwartz

Belo Horizonte
October 2015

Ficha catalográfica elaborada pela Biblioteca do ICEx - UFMG

Santos Júnior, Cássio Elias.

S237p Partial least square for face hashing / Cássio Elias Santos Júnior. — Belo Horizonte, 2015.
xxviii, 72f. : il. ; 29cm.

Dissertação (mestrado) - Universidade Federal de Minas Gerais Departamento de Ciência da Computação.

Orientador: William Robson Schwartz.

1. Computação - Teses. 2. Visão por computador-Teses. 3 Indexação de imagens - Teses I. Orientador II. Título.

519.6*84(043)

UNIVERSIDADE FEDERAL DE MINAS GERAIS
INSTITUTO DE CIÊNCIAS EXATAS
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

FOLHA DE APROVAÇÃO

Partial least squares for face hashing

CÁSSIO ELIAS DOS SANTOS JÚNIOR

Dissertação defendida e aprovada pela banca examinadora constituída pelos Senhores:

[illegible]

Prof. William Robson Schwartz - Orientador
Departamento de Ciência da Computação - UFMG

[illegible]

Prof. Carlos Eduardo Thomaz
Departamento de Engenharia Elétrica - FEI

[illegible]

Prof. Erickson Rangel do Nascimento
Departamento de Ciência da Computação - UFMG

Belo Horizonte, 24 de agosto de 2015.

Dedicado à Jéssica e aos meus pais, Francisco e Terezinha. Dedicado à memória de Celsa e Marli.

Acknowledgments

First, I would like to thank deeply professor William Robson Schwartz for the outstanding orientation on my undergraduate and graduate studies.

I would also like to thank professors Ewa Kijak, Guillaume Gravier and Silvio J. F. Guimarães for their intellectual and financial support without which this work would never be completed.

I thank the members of the examination committee, professors Erickson R. do Nascimento and Carlos E. Thomaz, for the enrichment that they brought to this work.

I am eternally thankful for all support that my parents gave me, allowing me to focus on research and studies.

I am very grateful for my beloved Jessica I. C. Souza for all support and love that she gave me throughout my master.

I am also very grateful for all members of the SSIG team, the NPDI lab and the Linkmedia team for kindly receiving me in such amiable way and for all my friends in Brazil: Adriano F. Schiavon, Andrey Bicalho Santos, Antônio A. Nazaré Jr., Artur J. L. Correia, Bruno do N. Teixeira, Carlos A. Caetano Jr., Carlos A. P. Filho, César A. M. Ferreira, David Saldana, Elerson R. da S. Santos, Filipe D. M. de Souza, Gabriel R. Gonçalves, Henrique B. da Silva, Jéssica S. de Souza, Júlia E. E. de Oliveira, Kleber J. F. de Souza, Lilian C. B. dos Santos, Marcelo de M. Coelho, Marco T. A. Rodrigues, Ramon F. Pessoa, Raphael F. de C. Prates, Rensso V. H. M. Colque, Ricardo B. Kloss, Samir M. A. Leão, Samira S. da Silva, Sandra E. F. de Avila, Tiago O. Cunha, Victor H. C. de Melo, Virginia F. Mota, Waner Miranda; and in France: Ahmet Iscen, Amélie Royer, Anca-Roxana Simon, Himalaya Jain, Li Weng, Miaojing Shi, Petra Bosilj, Ronan Sicre, Vedran Vukotic.

I awould like to thank the Brazilian National Research Council – CNPq (Grant #477457/2013-4), the Minas Gerais Research Foundation – FAPEMIG (Grants APQ-00567-14 and CEX - APQ-03195-13) and the Coordination for the Improvement of Higher Education Personnel – CAPES (DeepEyes Project and STIC-AMSUD 001/2013).

“If everything you try works, you are not trying hard enough.”
(Gordon Moore)

Resumo

Reconhecimento de faces é um importante tópico de pesquisa devido a sua aplicação em vigilância, perícia e interação humano-computador. Nos últimos anos, uma miríade de métodos para identificação de faces foi proposta na literatura com apenas alguns destes focando em scalabilidade. Este trabalho propõe um simples porém eficiente método para identificação de faces escalável baseado em partial least squares (PLS) e em funções hash aleatórias independentes inspiradas por locality-sensitive hashing (LSH), resultando na abordagem chamada *PLS para indexação* (PLSH). Além disto, a abordagem original do PLSH é estendida utilizando seleção de características para reduzir o custo computacional para avaliar as funções hash baseadas em PLS, resultando na abordagem estado-da-arte *PLSH estendido* (ePLSH). O método proposto é avaliado nas bases de dados FERET e FRGCv1. Os resultados mostram redução significativa do número de indivíduos avaliados na identificação de faces (redução para $0,3\%$ da galeria), assim provendo speedup médio de até 233 vezes comparado com a abordagem força bruta (quando se avalia todos os indivíduos da galeria) e de até 58 vezes comparado com trabalhos anteriores na literatura.

Abstract

Face identification is an important research topic due to areas such as its application to surveillance, forensics and human-computer interaction. In the past few years, a myriad of methods for face identification has been proposed in the literature, with just a few among them focusing on scalability. In this work, we propose a simple but efficient approach for scalable face identification based on partial least squares (PLS) and random independent hash functions inspired by locality-sensitive hashing (LSH), resulting in the *PLS for hashing* (PLSH) approach. The original PLSH approach is further extended using feature selection to reduce the computational cost to evaluate the PLS-based hash functions, resulting in the state-of-the-art *extended PLSH* approach (ePLSH). The proposed approach is evaluated in the dataset FERET and in the dataset FRGCv1. The results show significant reduction in the number of subjects evaluated in the face identification (reduced to 0.3% of the gallery), providing averaged speedups up to 233 times compared to evaluating all subjects in the face gallery and 58 times compared to previous works in the literature.

List of Figures

List of Tables

List of Algorithms

List of Acronyms

CCA canonical correlation analysis

BRIEF binary robust independent elementary features

BRISK binary robust invariant scalable keypoints

CLBP circular local binary patterns

ePLSH extended PLS for face hashing

FAST features from accelerated segment test

FERET facial recognition technology

FREAK fast retina keypoint

FRGC face recognition grand challenge

HOG histogram of oriented gradients

ITQ iterative quantization

LBP local binary patterns

LDA linear discriminant analysis

LDE linear discriminant embedding

LFW labeled faces in the wild

LLE locally linear embedding

LSH locality sensitive hashing

MARR maximum achievable recognition rate

NIPALS nonlinear iterative PLS

NN	nearest neighbors
ORB	oriented FAST and rotated BRIEF
PCA	principal component analysis
PLEB	point location in equal balls
PLS	partial least squares
PLSH	PLS for face hashing
POEM	patterns of oriented edge magnitudes
PQ	product quantization
SIFT	scale invariant feature transform
SRC	sparse representation-based classification
SSH	similarity sensitive hashing
SURF	speeded-up robust features
SVM	support vector machine
VIP	variable importance on projection
VLAD	vector of locally aggregated descriptors

Contents

Chapter 1

Introduction

According to Chellappa et al. [2010], there are three tasks in face recognition depending on which scenario it will be applied: *verification*, *identification* and *watch-list*. In the verification task ($1 : 1$ matching problem), two face images are provided and the goal is to determine whether these images belong to the same subject. In the identification task ($1 : N$ matching problem), the goal is to determine the identity of a face image considering identities of subjects enrolled in a face gallery. The watch-list task ($1 : N$ matching problem), which may also be considered as an open-set recognition task [Wechsler, 2009], consists in determining the identity of a face image, similar to the identification task, but the subject may not be enrolled in the face gallery. In this case, the face recognition method may return an identity in the face gallery or a not-enrolled response for any given test sample.

In this dissertation, we focus on the face identification task. Specifically, the main goal is to provide a face identification approach scalable to galleries consisting of numerous subjects and on which common face identification approaches would probably fail on responding in low computational time. There are several applications for a scalable face identification method:

- In a surveillance scenario consisting of numerous cameras spread throughout a large area and with hundreds of people passing every minute, such as in a large city, airport or sport events, the face identification should be able to return identities from a potential large number of suspects fast enough such that authorities have a chance to respond in an emergency case.

- In human-interaction, such as in video game consoles, marketing or interaction support for users with impairments and special needs, a face identification method

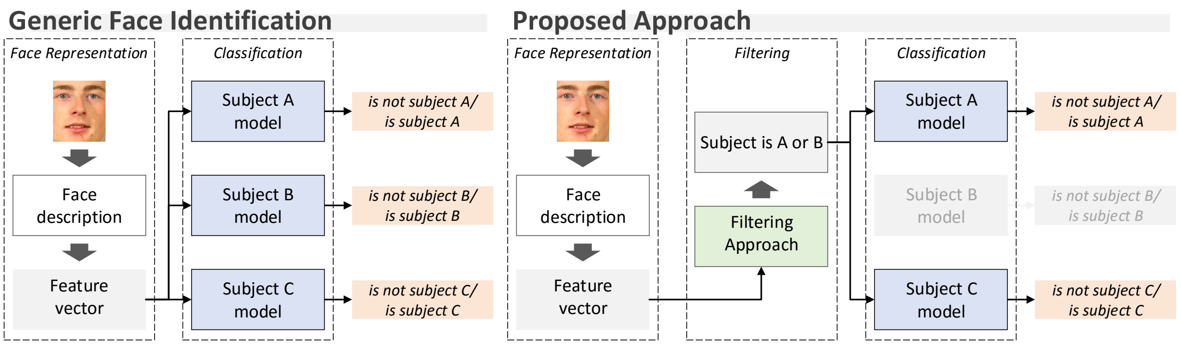


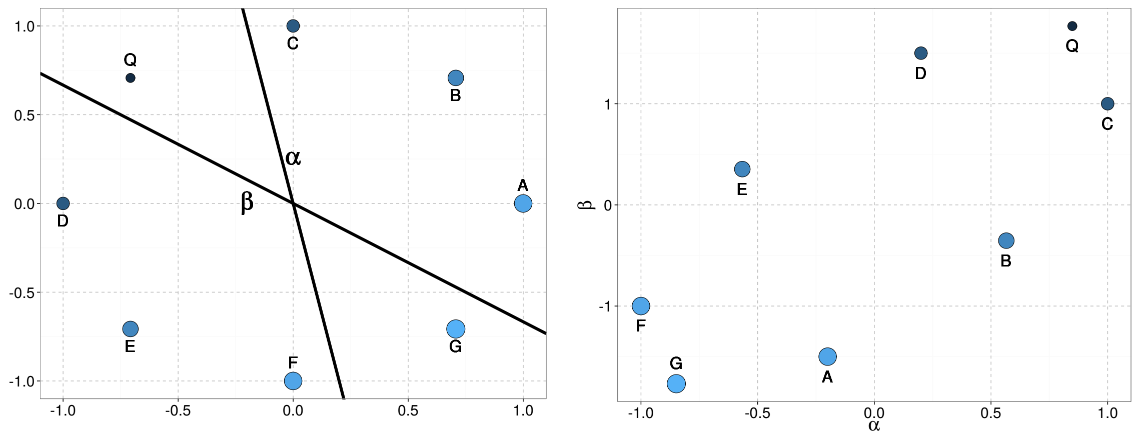
Figure 1.1: Common face identification pipeline and the proposed pipeline with the filtering approach which is used to reduce the number of evaluations in the classification step with low computational cost. The filtering approach is the main contribution in this work and it is tailored considering recent advances in large-scale image retrieval and face identification based on PLS.

could be used in an online system to store profiles and preferences for world-wide users and for fast identifying them and loading their data anyplace in the world.

- In social media, such as tagging people faces automatically in images, the face identification must be able to respond fast since slow websites tend to reduce the number of visitors [Brutlag, 2009].

The few aforementioned applications show the importance of performing face identification fastly and, in fact, several works in the literature have been developed in the past years motivated by these same types of applications (surveillance, forensics, human-computer interaction, and social media). However, most of the works focus on developing fast methods to evaluate one test face and a single subject enrolled in the gallery. These methods usually develop low computational cost feature descriptors for face images that are discriminative and with low memory footprint enough to process several images per second. Note that these methods still depend on evaluating all subjects in the face gallery. Therefore, if the number of subjects in the gallery increases significantly, these methods will not be able to respond fastly and new methods shall be developed to scale the face identification to this larger gallery.

Face identification methods usually consist of a face representation or description in the feature vector where mathematical models can be applied to determine the face identity. In this case, it is used one model to determine each identity in the face gallery, therefore, being necessary a number of models equal to the gallery size. Note that the parameters in each model are learned using samples for each subject in the face gallery and every model must be evaluated to correctly identify a test sample. In this dissertation, we propose a method to reduce the number of models evaluated in the face



(a) (b)

Figure 1.2: LSH idea illustration. (a) Random directions α and β in the original space (b) and the projected samples. The goal is to find the enrolled samples D and C from the query sample Q, which can be accomplished considering the projections α and β.

identification by eliminating identities that are somewhat clearly not the identity in the test sample. Figure 1.2a illustrates the common face identification pipeline employed in practice and the main component tackled in this dissertation.

There is an extensive literature of works regarding large-scale image retrieval that could be employed in face identification. However, most of these works focus on returning a list containing images from the dataset that are similar to the test image. Although reasonable to recover images in large datasets, such approaches are not suitable to apply directly to the face identification task. The models from subjects in the face gallery should optimally be described regarding the discriminative features related to each subject identity, which might consume less memory, specially if several samples per subject are available, and less computational time, since only discriminative features are evaluated to determine the face identity.

Note that we do not claim to solve all of the aforementioned problems, but we hope to provide in this work some details for the future development of scalable face identification and a practical -but not optimal- alternative for the methods presented in the literature. The proposed approach is inspired by the family of methods regarded as *locality-sensitive hashing* (LSH), which are the most popular large-scale image retrieval method in the literature, and the *partial least squares* (PLS), which has been explored intensively in our research group in several past works regarding face recognition [Guo et al., 2011; Schwartz et al., 2012; de Paulo Carlos et al., 2015; Santos Jr and Schwartz, 2014] and numerous other tasks. We call the proposed approach PLS *for hashing*, abbreviated to PLSH and ePLSH in its extension.

The main goal in LSH is to approximate the representation of samples in the high dimensional space using a small binary representation where the search can be implemented efficiently, employing a hash structure to approximate near-identical binary representations. The idea in LSH is to generate random hash functions to map the feature descriptor in the high dimensional representation to bits in the binary representation.

As an example of LSH, suppose that we have a two dimensional query sample Q and samples A to G enrolled in the gallery (see illustration in Figure 1.2a). The goal is to find the closest samples to Q using the l_2 distance metric (Euclidian distance), which are C and D. In LSH, the search for the nearest neighbors considering the l_2 distance metric is approximated by generating random projection vectors in which each element is drawn from a standard normal distribution. The idea is that, if two samples are close in the high dimensional space (similar vectors), they will almost surely be projected to similar values using the random projection vector. For instance, in Figure 1.2a, we select the orientation of a projection line α from a standard normal distribution and present the projected samples in the horizontal axis of Figure 1.2b. However, this is not always the case as some distant samples will also be projected to the same value as close samples. The random projection α in Figure 1.2 does not separate the sample B from Q, which are far apart in the original space. In this case, it is necessary a second projection β along with α to separate B from Q.

In the PLSH approach, the random projection in the aforementioned example is replaced by PLS regression, which provides discriminability among subjects in the face gallery and allow us to employ a combination of different feature descriptors to generate a robust description of the face image. PLSH is able to provide significant improvement over the brute-force approach (evaluating all subjects in the gallery) and compared to other approaches in the literature. Furthermore, since the evaluation of hash functions in PLSH requires a dot product between the feature and regression vectors, additional speedup can be achieved by employing feature selection methods, resulting on the extended version of PLSH (ePLSH). The following contributions are presented in this work.

- A fast approach for face identification that support a combination of several feature descriptors and high dimensional feature vectors.

- The proposed approach presents at least comparable performance with other methods in the literature and up to 58 times faster when enough samples per subject are available for train.

- Extensive discussion and experimentation regarding alternative implementations that may guide future development in scalable face identification methods.

- The proposed approach is easy to implement and to deploy in practice since only two trade-off parameters need to be estimated.

The remaining of this work is organized as follows. In Chapter 2, we review works related to face identification, fast face identification and large-scale image retrieval. In Chapter 3, we describe PLS for regression, face identification and face hashing, which are the main components for the proposed face identification pipeline. Experiments and discussions regarding the proposed approach are discussed in Chapters 4. The main parameters needed to employ the proposed approach in practice are discussed in Sections 4.1.3 and 4.2.2, for PLSH, and in 4.3.3, for ePLSH. Finally, we conclude this work with final remarks and author suggestions for future directions in Chapter 5.

Chapter 2

Literature Review

This chapter reviews works related to face identification (Section 2.1) and large-scale image retrieval (Section 2.2). The goal of this chapter is to provide a background necessary to understand the development of the proposed approach and not a comprehensive literature review of face identification and large-scale image retrieval. The reader may find more information regarding face identification in the book titled *Handbook of face recognition* [Li and Jain, 2011]. For large-scale image retrieval, we refer the reader to the work [Wang et al., 2014a], regarding locality-sensitive hashing, and [Muja and Lowe, 2014], about search trees.

2.1 Face Identification

Face identification methods consist generally of two components: classifier and face representation. The classifier is responsible for receiving the face representation and returning an identity in the gallery, more specifically, it evaluates whether a face representation from a test image refers to a subject in the face gallery.

The face representation consists of feature descriptors, which are usually employed instead of pixel values for three reasons [Shakhnarovich and Moghaddam, 2011]. First, the space spammed by pixel values of face images is high dimensional. For example, a small image with 128 pixels height and width presents more than 49 thousand intensity values. Second, aspects such as changes in illumination, pose of the face, and aging, induce a large variation of pixel values, which scatter samples from the same subject in the feature space. For classification purposes, it is desirable a compact representation of samples from the same subject (low intra-class variation), while keeping samples from different subjects apart (high inter-class distances). Finally, pixel values, especially among neighbor pixels, are highly correlated, which may impose difficulties to classifiers. For instance, a large number of levels might be necessary in axis-oriented decision trees to discriminate samples from different subjects.

Early works on face identification focused on subspace representation to tackle issues related to high dimensionality, correlation among pixels and noise [Sirovich and Kirby, 1987; Belhumeur et al., 1997]. In this case, the subspace is calculated considering compact description of samples for the same subject using principal component analysis (PCA) and maximum distance between samples from different subjects using linear discriminant analysis (LDA). Basri and Jacobs [2004] showed that lightning conditions in face images can be represented in less than 10 dimensions in a PCA-like subspace of gray-scale images. However, the aforementioned linear transformations to subspaces are not enough to capture all possible variations of faces. For this reason, image feature descriptors are considered in face identification.

Feature descriptors provide a robust manner to represent face images invariant to misalignment, illumination and pose of the face. Regarding feature descriptors considered in face identification, the most commons are local binary patterns (LBP) [Xie et al., 2012; Klare and Jain, 2013], Gabor filters [Gu et al., 2012; Oh et al., 2013] and descriptors based on gradient images [Vu et al., 2012; Liao et al., 2013]. These feature descriptors capture mainly texture and shape of the face image, which are relevant for face identification [Schwartz et al., 2012]. There are two manners to represent the face image [Kämäräinen et al., 2011]: *appearance-based* (holistic), where the whole face image is represented in a feature descriptor vector; and *feature-based* (local), where fiducial points of the face image, such as nose tip or corners, eyes and mouth, are represented instead of the whole image.

The advantage of the holistic representation is the rich and easy encoding of the overall appearance of the face image. Since every pixel value contributes somehow to the final feature descriptor, more information is available to distinguish between samples from different subjects. However, preprocessing is usually necessary to correct misalignment, illumination and pose. Feature descriptors commonly employed in holistic methods are the local binary patterns (LBP) [Xie et al., 2012], Gabor filters [Gu et al., 2012], combination of both [Salah et al., 2013], and large feature sets coupled with dimension reduction techniques [Schwartz et al., 2012].

The advantage of the local representation is its robustness to differences in pose, partial occlusion and shadowing. If some fiducial points are shadowed or occluded due to pose, for instance, other points may still be used to recognize the face image. However, the resulting feature vector is often ambiguous and imposes difficulties to identify the face image due to the reduced amount of data present in the small patch around the fiducial point. Common feature descriptors employed in local methods include LBP [Klare and Jain, 2013] and Gabor filter [Oh et al., 2013].

Fiducial points can be detected considering salient regions in the face image, which include corners and textured regions in the face. These salient regions, opposed to homogeneous regions such as cheek and forehead, tend to be stable among face images in different poses and lightning conditions. However, a method to match the detected salient regions among face images is necessary to compare feature descriptors. Liao et al. [2013] employ the popular SIFT [Lowe, 2004] to detect and match salient regions among face images. Another option is to learn common detectors for fiducial points (eye corner, nose tip, among others) such that the match of fiducial points among face images is no longer necessary since feature descriptors from a common type of fiducial point can be directly compared [Valstar and Pantic, 2012].

In the past few years, a different approach based on sparse representation-based classification (SRC) has been providing high accuracy in face identification datasets [Wright et al., 2009]. SRC consists in representing a test face image as a linear combination of a dictionary of images, which is learned using samples in the face gallery. Although the original proposal of SRC requires a fair number of controlled samples per subject for training, Deng et al. [2013] extended SRC to cope with few uncontrolled samples in the face gallery.

2.1.1 Fast Face Identification

Fast face identification is not a largely explored research topic and there are few works in the literature about it [Barkan et al., 2013; Deng et al., 2012; He et al., 2014; Yuan et al., 2005; Schwartz et al., 2012]. In [Barkan et al., 2013], compact descriptors based on local binary patterns are used to compare fastly the candidates in the face gallery. In [Deng et al., 2012] and [He et al., 2014], a fast optimization algorithm is considered for SRC to reduce the computational cost when calculating the linear combination between the test and the samples in the dictionary. Although the aforementioned methods provide significant improvement on the test-subject comparison, poor performance is observed when there are numerous subjects in the face gallery since these approaches still present linear asymptotic complexity with the gallery size.

To approach face identification in large galleries, a cascade of classifiers to discard a considerable number of candidates in early initial stages with low computation cost classifiers was proposed by Yuan et al. [2005]. To keep high accuracy, the final stages of the cascade consists in more accurate and time-consuming classifiers. In [Schwartz et al., 2012], a binary tree structure was used to reduce the number of subjects tested in the face gallery, resulting in a reduced computational complexity considering the number of subjects in the face gallery when compared to the brute-force approach.

The approach proposed in this dissertation is an extension of the work [Schwartz et al., 2012] and the main difference is the employment of hashing instead of search trees. PLS is also considered with a combination of feature descriptors as in Schwartz et al. [2012], which improves the face identification recognition rate compared to single feature descriptors. In this case, the contribution of the proposed approach lies in the distinct manner in which PLS is employed for hashing and the considerable improvement in speedup compared to the aforementioned scalable face identification approaches.

2.2 Large-Scale Image Retrieval

The goal in the image retrieval task is to return a sorted list of "relevant" images enrolled in the gallery considering their similarity to a test sample. The common associated task, referred to as k-NN, consists in calculating distances between test and gallery samples and returning the first k closest samples. The brute-force k-NN approach consists in calculating all the distances between the test and gallery samples and sorting them, which is unfeasible for large galleries. In this case, efficient sub-linear algorithms to solve the k-NN task are employed. An example is kd-tree [Gan et al., 2007], which consists in splitting recursively the space in subregions and organizing them in a hierarchical structure that provides averaged asymptotic complexity $O(log(n))$ for a given gallery size n. Although kd-tree considers only the l_2 distance metric, it is possible to extend the idea to other distance metrics [Uhlmann, 1991].

The main reason of the poor performance related to distance-based approaches in high dimensional data is the small difference among distance values, which hampers the analyses of which samples are similar or dissimilar (both will present almost the same values). Furthermore, axis-oriented splits employed in kd-trees are not sufficient to approximate nearest neighbor search in the high dimensional space [Chávez et al., 2001]. In this case, the solution is embedding feature descriptors in a low dimensional space where conventional distance metrics return relevant values.

In practice, it is hard to embed feature descriptors in a low dimensional space without losing information. Therefore, some approaches consider a relaxed restriction version of k-NN, regarded as approximated nearest neighbors or ϵ-nearest neighbors (ϵ-NN), which consists in returning samples p with minimum distance from the test sample q, where the distance is at most $(1+\epsilon)d(p,q)$, for $\epsilon > 0$. The performance of ϵ-NN methods is usually measured as a function of ϵ, which represents the approximation error of the distance metric $d(p,q)$ in the high dimensional space.

There exists a decision problem related to k-NN, regarded as r-nearest neighbors (r-NN), which is more tractable and efficiently approached than k-NN. The r-NN task can be reduced to k-NN using a binary-search approach [Har-Peled, 2001] and consists in returning all samples in the gallery that are within a fixed radius distance r from the test sample. The approximated version of r-NN, regarded as (r, c)-NN, consists in returning all samples within a distance at most rc from the test sample. In this case, the parameter c is important to delimiter regions in the feature space where it is possible to determine the worst case scenario for retrieving samples within r.

There are several approaches for large-scale image retrieval in the literature. However, a complete review and description of all types of approaches is out-of-scope in this work. Instead, we focus on past works that have been successfully applied to face identification. In this context, tree-based approaches are described in Section 2.2.1, locality-sensitive hashing in Section 2.2.2, and Hamming embedding-based approaches in Section 2.2.3.

2.2.1 Tree-Based Approaches

Large-scale image retrieval based on search trees consists in partitioning recursively the samples and organizing them in the partitions using a tree structure. In this way, the organization of samples is hierarchical, with the higher level (root node of the tree) representing the full sample set and lower levels (leafs) representing subsets of samples. The number of levels in the tree (l) depends on the number of samples (n), the number of partitions in each level (p), and whether the number of samples in each partition is the same (balanced partitions), in which case $l = \lceil log_p(n) \rceil$. Unbalanced partitions lead to $l = n$, in the worst case. Controlling l is important since the computational cost to traverse the tree from the root node to a leaf has asymptotic complexity $O(l)$.

Partitioning is the main component of the search tree and it determines the effectiveness of the tree approach to balance partitions (reduce l) and to guide the search towards most likely nearest neighbors (reduce traversals). However, it is rarely possible to split real data in balanced partitions based on common features among samples within each partition. For instance, gender, ethnicity and age are hardly balanced among subjects in a specific group (engineer students, politicians, travelers, among others). Nevertheless, balanced partitions can be forced using random split of samples at the cost of traversing the tree a few times to find the nearest neighbors [Schwartz et al., 2012].

Feature transformations, such as PCA [Sproull, 1991], may be considered before each partitioning to reduce, or eliminate, the influence of redundant, irrelevant and noisy features. Another option is to employ partitioning approaches based on robust classifiers, such as SVM [Pang et al., 2005] and PLS [Schwartz et al., 2012], which provide good performance even with noisy and redundant features. The disadvantage of these approaches is the higher computational cost to traverse each node in the tree ($O(d)$, where d is the dimensionality of the data) compared to axis oriented partitioning ($O(1)$) [Wang et al., 2014b].

The following approaches employ partitioning considering maximum variance directions in the feature space based on the work of Sproull [1991]. Silpa-Anan and Hartley [2008] employ a similar approach, but considering multiple *kd*-trees and a shared priority queue, which determines the order to evaluate samples for nearest neighbors candidates. An improvement of computational cost is achieved considering a feature subset, since a *kd*-tree usually considers few features to split samples and the exact same *kd*-tree is obtained using only these features. Wang et al. [2014b] employ hyperplanes considering elements with values 1, 0 and -1 to reduce the computational cost to evaluate nodes in the tree presenting asymptotic complexity $O(d)$ for hyperplanes in $\mathbb{R}^d$. The approach also employs numerous trees, learned considering a subset of features selected randomly with probability proportional to its variance in the training samples.

Random projections are an alternative approach compared to maximum variance directions and usually provide competitive results as demonstrated in the following approaches. Dasgupta and Freund [2008] employ the median value of the samples projected in the random direction as threshold, which generates a balanced tree. Liu et al. [2004] employ the mean value of the most distant values projected in the random subspace as threshold and allow samples to be shared among node children, which increase the number of levels in the tree but reduce the number of traversals in general. In this case, the speedup is obtained since less node evaluations are necessary, which has the same computational cost of evaluating one sample for nearest neighbor (asymptotic complexity $O(d)$).

Other methods to split and traverse nodes include the work of Nister and Stewenius [2006], which splits samples based on the k-means algorithm and traverses nodes considering the distance from the sample to the closest cluster center. Pang et al. [2005] also employ a clustering approach to split data called locally linear embedding (LLE) and maximum margin classifiers (support vector machines) to traverse nodes. Schwartz et al. [2012] employ random partitioning of samples and partial least squares (PLS) regression to traverse nodes.

2.2.2 Locality-Sensitive Hashing

Locality-sensitive hashing (LSH) refers to a family of embedding approaches (see Section 2.2.3 for a brief description of embedding) that aims at mapping similar feature descriptors to the same hash table bucket with high probability while keeping dissimilar features in different buckets. LSH approaches provide logarithm asymptotic complexity in time and linear complexity in space. Indyk and Motwani [1998] were the first to introduce the term locality-sensitive hashing in 1998. However, Broder [1997] was the first to describe a LSH approach (min-hash) in 1997 to cluster and to detect near-duplicated web pages. Since then, several approaches has been proposed in the literature to include different domains in the LSH framework. A summary of numerous LSH approaches can be found in [Wang et al., 2014a].

An approach is regarded as being LSH, with arguments r, c, p_1, p_2 ($p_1 > p_2$), if the hash function $h(x)$ employed belongs to a family of hash functions $\mathcal{H}$, which approximates the distance metric $d(p, q)$ of the feature descriptors p and q in the following manner:

$$\text{if } d(p,q) \leq r \text{ then } Pr\big(h(p) = h(q)\big) \geq p_1,$$
$$\text{if } d(p,q) \geq cr \text{ then } Pr\big(h(p) = h(q)\big) \leq p_2,$$

where Pr denotes probability and r denotes the maximum distance $d(p, q)$ that maps p and q to the same bucket with minimum probability p_1. The second case ensures that distant feature descriptors ($d(p, q) \geq cr$) has low probability (p_2) to be mapped into the same bucket for a constant $c > 1$. The LSH considers the (r, c)-NN task and, in practice, it can be employed directly instead of k-NN by setting r according to the desired precision and recall of the samples retrieved and maximum approximation error equal to ϵ, where $c = (1 + \epsilon)$.

LSH employs K hash functions $\{h_1, ..., h_K\}$ selected independently and uniformly from $\mathcal{H}$. Raising K provides higher precision in practice, but increases the size of the hash table and reduces recall. For this reason, hash functions are usually grouped in L groups $\{g_1, ..., g_L\}$ of K hash functions ($g_i = \{h_{i,1}, ..., h_{i,K}\}$), resulting in L hash tables and LK hash functions in total. Raising L is known to improve recall, but the values of L and K are bounded by the computational cost of hashing features in practice. If K is set to $log_{1/p_2}(n)$ and L is set to n^ρ, where $\rho = log(1/p_1)/log(1/p_2)$, the LSH framework generates algorithms with asymptotic space complexity $O(dn + n^{1+\rho})$, n and d denote number and dimensionality of the samples, respectively, and time complexity bounded by $O(n^\rho)$ similarity calculations and $O(n^\rho log_{1/p_2}(n))$ hash function evaluations [Datar et al., 2004].

In practice, given r, $c \to 1$ is desirable, although c too small reduces precision (p_2 increase), which is compensated by generating more hash functions (given by $log_{1/p_2}(n)$). For example, Datar et al. [2004] calculate $\rho = 1/c$ considering hash functions sampled from a p-stable distribution to approximate the l_1 or l_2 distance metrics, which is close to the lower theoretical limit $\rho \to 1/2c$ [Motwani et al., 2007]. The approximation factor can also be calculated using Monte Carlo for complex hash functions [Terasawa and Tanaka, 2007].

There are two types of hash functions in LSH [Wang et al., 2014a]: *data independent*, where hash functions are defined regardless of the data; and *data dependent*, where the parameters of the hash functions are selected according to the training data. These two types are different from supervised and unsupervised learning of hash functions, in which the difference lies on whether data label is considered. For instance, data dependent hash functions may not consider the label of the data when learning hash functions. However, all supervised hash functions are intrinsically data dependent, since the family of hash functions $\mathcal{H}$ will be selected to discriminate labels.

Data independent hash functions are employed in the works of Datar et al. [2004], based on p-stable distributions; Chum et al. [2008], based on min-hash; Joly et al. [2004] and Poullot et al. [2007], both works based on space filling curves. Data independent hash functions are usually employed in heterogeneous data like in the object recognition task. In this case, the overall distribution of the data is not modeled easily using data dependent hash functions. For instance, the distribution of a common object (more samples) may outweigh uncommon objects (few samples). In this case, unsupervised data dependent functions will be biased toward representing the sample distribution of the common object. Other advantages of the data independent hash functions are the fast learning process, which is independent from the gallery size, and the enrollment of new samples, which does not require retraining hash functions.

Data dependent hash functions select a family $\mathcal{H}$ considering aspects of the data, such as discriminability among different labels and dimensions with maximum energy. In this case, hash functions unrelated to the data are discarded, which is not the case in data independent hash functions. Considering the same number of hash functions employed in the data independent approach, the number of relevant hash functions which raise the gap between higher p_1 and lower p_2 is often higher in data dependent hash functions. Examples of works employing data dependent hash functions include metric learning [Kulis et al., 2009], k-means [Datar et al., 2004], spectral hashing [Weiss et al., 2009], restricted Boltzmann machine [Salakhutdinov et al., 2007], maximum margin [Joly and Buisson, 2011] and deep learning [Torralba et al., 2008].

There are numerous LSH approaches for different metric spaces. The most common applications include LSH approaches for l_p metric space [Datar et al., 2004] based on p-stable distributions; random projections [Andoni and Indyk, 2006], which approximate cosine distances; Jaccard coefficient [Broder, 1997]; and Hamming distances [Indyk and Motwani, 1998], where the goal is to provide approximated nearest neighbors as opposed to exact nearest neighbors as presented in Section 2.2.3.

It is important to emphasize that the proposed approach is not included in the LSH family. We do employ hash functions generated independently from each other and the proposed approach considers data labels, but there is no associated distance metric and, therefore, no approximated k-NN solution. We focus on returning correct identities in a shortlist of candidates rather than approximating nearest neighbors in a given metric space. The proposed approach also behaves similarly to LSH methods, where the increase in the number of hash functions provides improved results, but we cannot prove the approximation limits of the proposed approach in the same way as in LSH. In our experiments, we notice that the results never exceed the recognition rate of the brute-force based on PLS, which might indicate that the proposed method approximates the results from PLS-based approaches.

2.2.3 Hamming-based Approaches

The most common approach in large-scale image retrieval consists in embedding feature descriptors in a low dimensional space where locality is preserved (similar feature descriptors have similar representation in the low dimensional space) and where nearest neighbor search can be implemented efficiently. This section focuses on embedding feature descriptors in the Hamming space, which is widely used in the literature and consists in the space spanned by a set of 2^S binary strings with length S. Other options for embedding include linear discriminant embedding (LDE) [Chen et al., 2005], LDE variation [Hua et al., 2007], or embedding in Huffman trees [Chandrasekhar et al., 2009, 2012]. The difference between tree-based search (Section 2.2.1) and embedding feature descriptors in Huffman trees is the hierarchical organization of samples in the former compared to sample encoding in the latter.

Most LSH approaches are Hamming embedding since LSH aims at mapping similar feature descriptors in the same hash bucket. However, there are other types of Hamming embedding approaches and common properties among Hamming-based methods (like boosting and efficient distance calculation) that are worth to mention. Most of the works presented in this section have been discussed in Section 2.2.2, however, now we focus on the Hamming embedding aspects independent from LSH in this section.

The main advantages of Hamming space embedding are the low memory requirement (one binary string for each sample) and the efficient implementation of Hamming distance using bitwise operations (*xor* operation between two strings followed by a small code to count number of bits equal to one). Given the importance of operations in the Hamming space, specially in cryptography and error detection, CPU manufactures included an instruction to compute efficiently the number of bits equal to one in the instruction set (POPCNTSSE4.2). For the purpose of completeness and clear understanding of this work, the same notation of the Hamming distance employed by Bronstein et al. [2011] is considered. In this case, Hamming distance is defined as

$$d(X,Y)_{\mathbb{H}} = \frac{S}{2} - \frac{1}{2}\sum_{i=1}^{S} sgn(x_i y_i),$$

in the space $\mathbb{H}^S = \{-1,+1\}^S$, which is equivalent to the number of different elements in the strings $X = \{x_1, ..., x_S\}$ and $Y = \{y_1, ..., y_S\}$. The function $sgn(a)$ returns $+1$ if a is positive and -1, otherwise.

There are two tasks to retrieve nearest neighbors in the Hamming space [Norouzi et al., 2012]. One task, referred to as k-nearest neighbors (k-NN), consists in retrieving a fixed number of samples k with minimum distance to the test sample. The other task, referred to as *approximate query* or *point location in equal balls* (PLEB) [Indyk and Motwani, 1998], consists in retrieving all samples within a fixed maximum distance r to the test sample. There is no better way to retrieve samples in the k-NN task than evaluating all distances between the test and gallery samples and, although k-NN is unfeasible in practice, it is useful to access the recall among Hamming embedding methods [He et al., 2013; Gong et al., 2013; Wang et al., 2012].

The approximate query approach may be implemented using hash table where samples are indexed in multiple positions according to r. In this case, the limitations are the memory necessary to allocate the hash table (viable only for small S) and the large number of samples retrieved and indexed multiple times, which is proportional exponentially to r. The number of hash table positions visited is given by

$$v(r,S) = \sum_{i=0}^{r} \binom{S}{i}^{1}.$$

Even a reasonable value for r and S may represent a search for hash table positions higher than the gallery size. For instance, $r = 4$ and $S = 128$ result in $v = 11,017,633$. Furthermore, the memory necessary to allocate the hash table with 128 bits would be

[1] S-combination of i distinct elements (S! / k! (S - k)!).

at the order of undecillions. In this case, Norouzi et al. [2012] proposed an approach that approximates the query task by employing indexing of non-overlapping substrings in multiple hash tables to avoid the aforementioned issues.

Numerous approaches have been proposed in the literature to map feature descriptors in the Hamming space. Most of them are based on optimization such as minimizing the quantization error. The supervised optimization version consists in minimizing the distance of same-class feature vectors in the Hamming space while maximizing the distance of feature descriptors from different classes. In this case, elements in the Hamming string present different discriminative performances and a weighted distance version, such as

$$d_w(X, Y)_{\mathbb{H}} = \frac{S}{2} - \frac{1}{2}\sum_{i=1}^{S} \alpha_i sgn(x_i y_i), \tag{2.1}$$

provide better results. The weights α_i can be learned using an adaptive learning algorithm, as depicted in [Shakhnarovich, 2005].

Regarding unsupervised approaches, Jégou et al. [2012] employ quantization of feature descriptors considering a random rotation of the PCA transformation to ensure equal variance among projected features. In the unsupervised version, Gong et al. [2013] employ a similar approach but considering the minimal quantization error of zero-mean samples in a zero-centered binary hypercube. In this case, an efficient optimization algorithm can be formulated, referred to as *iterative quantization* (ITQ), which provides better results than the random rotation employed in [Jégou et al., 2012]. He et al. [2013] show that competitive results can be obtained by employing the minimization of the error in the Hamming embedding instead of squared error in the k-means algorithm used in the *product quantization* (PQ) method [Jégou et al., 2012]. Norouzi and Blei [2011] employ maximum-margin optimization between dissimilar quantized feature descriptors, which is carried following an neural network optimization algorithm according to an upper-bound function.

Regarding supervised or semi-supervised Hamming embedding, approaches based on metric learning have been the most common in the literature. Most of them can be formulated following the *similarity sensitive hashing* (SSH), described in [Shakhnarovich, 2005] and defined as $x_i = sgn(w_i f + b_i)$, where each bit x_i in the Hamming space depends on f, which is a d-dimensional feature descriptor, w_i denotes d-dimensional metric parameters and b_i is bias (zero for mean centered data).

In the supervised version, Gong et al. [2013] employ canonical correlation analysis

(CCA) rather than PCA in the ITQ method. Wang et al. [2012] propose to learn the quantization separately for each bit and incorporate correction of weak quantized bits when learning new ones. Jain et al. [2008] employ random directions weighted by Mahalanobis distance. Strecha et al. [2012] employ LDA, which maximizes the inter-class distance and minimize the intra-class variance. Bronstein et al. [2011] employ an optimization similar to LDA, but considering an exponential loss function appropriated to the bag-of-words model. We refer the reader to the work of Wang et al. [2012] for a summary table of SSH-based approaches.

Approaches other than SSH-based include the work of Liu et al. [2012], which employs kernel function optimized to consider minimum same and maximum not-same Hamming distances between classes. Another option is to employ deep learning algorithms, which have been an active research topic in recent years. In this case, the same principles adopted in Hamming embedding methods are employed in the neural network. For instance, the goal in deep learning based methods is to represent the whole image in a compact binary string. One of the challenges in considering deep learning approaches is the high computational cost induced by convolutional layers, which can be significantly reduced when related regions not related to the object in the image are discarded (background and homogeneous regions) [Mopuri and Babu, 2015]. Although simple, this approach provide better results than well-known methods such as the *vector of locally aggregated descriptors* (VLAD) [Jégou et al., 2012].

The optimization step in the neural network can also be tailored to minimize reconstruction loss from the binary string, distribute uniformly the binary string and provide strings with maximum independence among bits (reduce redundancy) [Erin Liong et al., 2015]. The method can be further extended to consider labels, in which the probability of samples with different labels receiving the same binary code is minimized while samples with same labels receiving the same binary code is maximized Lin et al. [2015]. Similar to the aforementioned works, Torralba et al. [2008] employ deep learning (four or five layers in the neural network hidden algorithm depending on the string length) to map same label samples in the same string.

In conclusion, the Hamming-embedding methods are more suitable for large datasets since tree-based approaches, for instance, still require a considerable amount of memory to store the tree structure. Furthermore, Hamming embedding methods are easy to distribute among nodes in a computer cluster, which is harder to achieve for search trees. We also noticed that, although numerous deep learning methods has been proposed in the past few years, they provide little increment in performance considering the ITQ approach [Erin Liong et al., 2015].

The best Hamming-embedding methods, in general, consider four optimization strategies. First, a balanced set of bits, which tends to distribute Hamming strings (hash codes) uniformly among samples in the dataset. Second, independency among bits, which provide shorter Hamming strings. Third, minimization of the quantization lost, which ensure that locality in the high dimensional representation of the image is preserved in the Hamming embedding space. Finally, since each bit in the Hamming string may contribute differently for the image retrieval accuracy, annotated data could be employed to weight bits according to their retrieval capability.

In the proposed approach, we consider Hamming embedding but without estimating the binary string directly. Instead, we calculate incrementally the similarity of subjects each time that a bit is estimated for the test sample. As will be explained in Section 3, this incremental approach is necessary so we can employ a weighted Hamming embedding, where each bit in the binary string is weighted by the likelihood of that bit, which provides better results than comparing binary strings belonging to pairs of samples directly.

Chapter 3

Methodology

This chapter describes the methods employed in the proposed approach, namely PLS for regression (Section 3.1) and PLS for face identification (Section 3.2). The proposed PLSH is described in Section 3.3 and in Section 3.4, we describe a PLSH extension (ePLSH), which consists in employing PLS-based feature selection to improve the performance of PLSH.

3.1 Partial Least Squares Regression

PLS is a regression method that combines ordinary least squares applied to a latent subspace of the feature vectors. Several works have employed PLS for face identification [Schwartz et al., 2012], face verification [Guo et al., 2011], and open-set face recognition [Santos Jr and Schwartz, 2014]. These works consider PLS mainly due to the robustness to combine several feature descriptors, capability to deal with thousands of dimensions, and robustness to unbalanced classes. In this work, we consider PLS due to the high accuracy presented when used to retrieve candidates in PLSH and the low computational cost to test samples since only a single dot product between the regression coefficients and the feature vector is necessary to estimate the PLS response.

PLS is calculated as follows. The p-dimensional latent subspace is estimated by decomposing the zero mean matrices $X_{n\times d}$, with n feature vectors and d dimensions, and Y_n, with response values, in

$$\begin{aligned} X_{n\times d} &= T_{n\times p}P^T_{d\times p} + E_{n\times d}, \\ Y_{n\times 1} &= U_{n\times p}Q_{p\times 1} + F_{n\times 1}, \end{aligned} \tag{3.1}$$

where $T_{n\times p}$ and $U_{n\times p}$ denote latent variables from feature vectors and response values,

Algorithm 1: NIPALS($X_{n \times d}, Y_n, p$)

```
1 for $i \leftarrow 1$ to $p$ do
2     start $u_i$ randomly or with some column of $X$
3     repeat
4         $w_i \leftarrow X^T u_i / \|X^T u_i\|$; $t_i \leftarrow X w_i$
5         $q_i \leftarrow Y^T t_i / \|Y^T t_i\|$; $u_i \leftarrow Y q_i$
6     until convergence;
7     $b_i \leftarrow u_i t_i / \|t_i\|$; $p_i \leftarrow X^T t_i / \|t_i\|$
8     $X \leftarrow X - t_i p_i^T$; $Y \leftarrow Y - b_i(t_i p_i^T)$
9     return $T, P, U, Q, W, B$
```

Figure 3.1: NIPALS algorithm. $X_{n \times d}$ and Y_n denote feature vectors and target values, respectively, with n samples and d dimensions. p denotes number of dimensions in the PLS model.

respectively. The matrix $P_{d \times p}$ and the vector Q_p represent loadings and the matrix E and the vector F are residuals from the transformation. PLS algorithms compute P and Q such that the covariance between U and T is maximum [Rosipal and Kramer, 2006]. We consider the nonlinear iterative PLS (NIPALS) algorithm [Wold, 1985] (presented in Figure 3.1) which calculates the maximum covariance between the latent variables $T = \{t_1, ..., t_p\}$ and $U = \{u_1, ..., u_p\}$ using the matrix $W_{d \times p} = \{w_1, ..., w_p\}$, such that

$$\arg\max[cov(t_i, u_i)]^2 = \underset{|w_i|=1}{\arg\max}[cov(Xw_i, Y)]^2.$$

The regression vector β between T and U is calculated using matrix W according to

$$\beta = W(P^T W)^{-1}(T^T T)^{-1} T^T Y. \tag{3.2}$$

The PLS regression response $\hat{y}$ for a probe feature vector $x_{1 \times d}$ is calculated according to $\hat{y} = \bar{y} + \beta^T(x - \bar{x})$, where $\bar{y}$ and $\bar{x}$ denote average values of Y and elements of X, respectively. The PLS model is defined as the variables necessary to estimate $\hat{y}$, which are β, $\bar{x}$ and $\bar{y}$.

Efficient implementations of the NIPALS algorithm using graphical cards exist in the literature and they can provide speedup of up to 30 times compared to the CPU version [Srinivasan et al., 2010]. For prototyping, we recommend the PLS package in the R programming language [Mevik and Wehrens, 2007], which include different PLS versions and tools to analyze the model learned.

An illustration of PLS latent variables is presented in Figure 3.2. We consider Fisher's IRIS dataset [Fisher, 1936], which contains 150 samples of three types of flowers

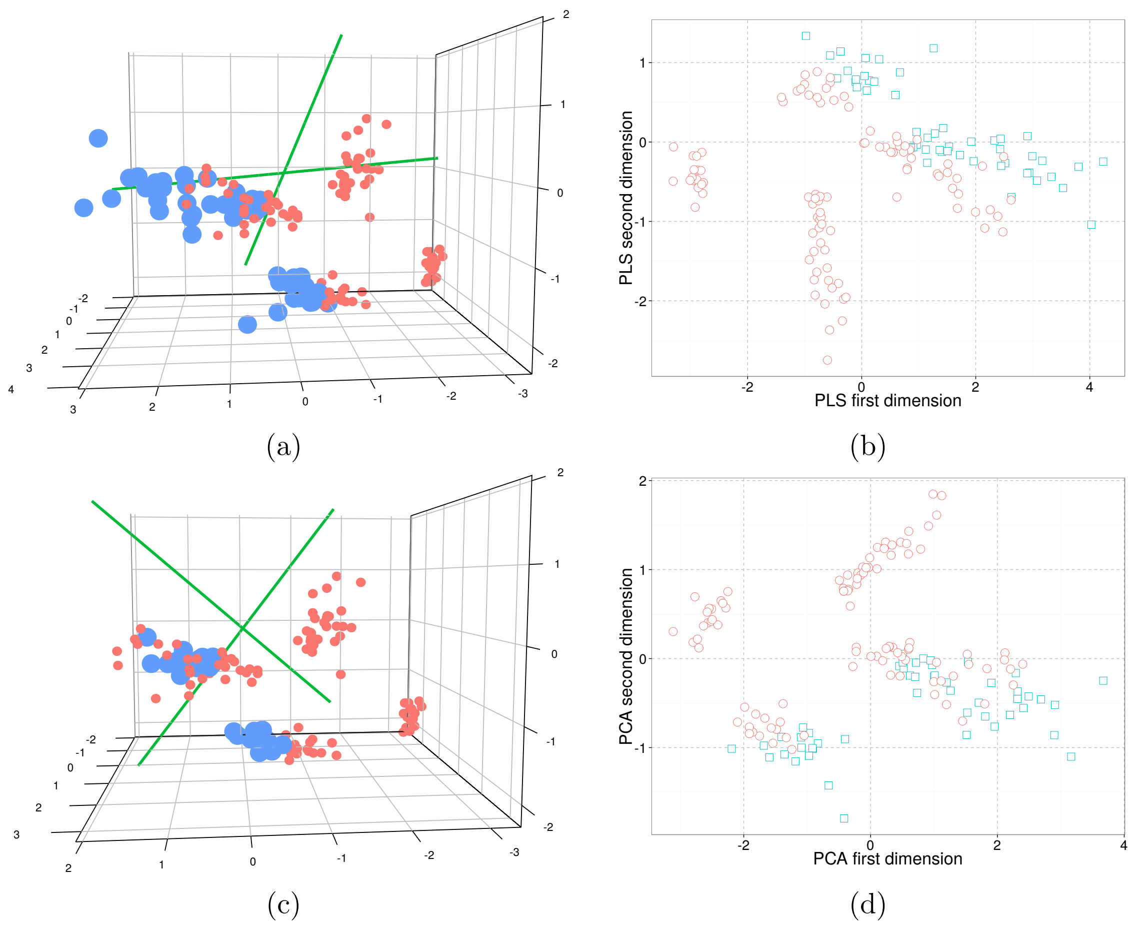


Figure 3.2: (a) PLS axis in 3-dimensional space and (b) features projected in the first two dimensions of PLS. (c) PCA axis in 3-dimensional space and (d) features projected in the first two dimensions of PCA. The goal is to separate Virginica flower samples (blue points) from other flower samples (red points). Note that, although PCA and PLS employ a similar transformation, the subspace generated by PLS discriminate better than PCA the two types of samples. Best visualized in color.

(Setosa, Versicolor and Virginica). Three predictors are considered in the illustration (petal length, sepal length and width). The goal is to separate Virginica samples (response variable is set to $+1$) from the others (response variable is set to -1). The data are presented to Algorithm 3.1 and the projection vectors W are plotted along with the data in Figures 3.2a and 3.2b, where the horizontal axis is the first PLS dimension, the one that maximizes the correlation between predictors and responses. The second PLS dimension (vertical axis in Figure 3.2b) is orthogonal to the first PLS dimension and presents the second higher correlation between predictions and responses. In this case, PLS attempts to project all Virginica samples to a single point in the latent space different from a point in which the remaining samples are projected. If such projection is not possible, one that minimizes the squared error to a single point

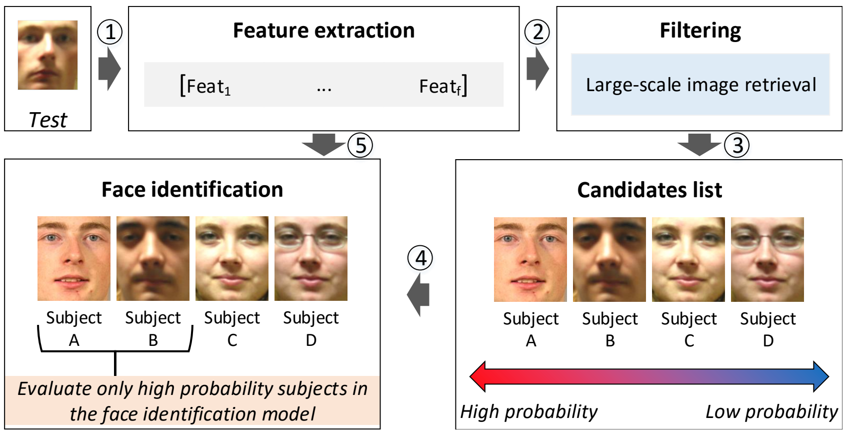


Figure 3.3: Overview of the filtering and the face identification pipeline. (1) Different feature descriptors are extracted from the test image and concatenated resulting in a large feature vector more robust to image effects than single feature descriptors. (2) The feature vector is presented to the filtering approach, which employs a large-scale image retrieval approach to (3) generate the candidate list sorted by the probability that the candidate is the subject in the test image. (4) A small percentage of the candidate list (high probability candidates) is presented to the face identification which will evaluate only the models relative to these candidates.

will be estimated. Note that if we use PCA (see Figures 3.2c and 3.2d), the subspace calculated will be different because PCA considers directions that explain most of the variance in the data. The script used to generate Figures 3.2a and 3.2b is available in Appendix A.

3.2 Face Identification Based on Partial Least Squares

The proposed approach consists in filtering subjects in the gallery using methods for large-scale image retrieval. For a given face identification approach, the evaluation of all subjects in the gallery (without filtering) is regarded as the *brute-force* approach, which is undesirable since the asymptotic time complexity is linear with the number of subjects enrolled in the gallery. The filtering approach consists in providing a shortlist to the face identification so that it evaluates only subjects presented in that shortlist.

An overview of the filtering and face identification pipeline is presented in Figure 3.3, which consists of the following steps. Different feature descriptors are extracted from a probe sample and concatenated in the first step (feature extraction). Then, the combined feature vector is presented to the filtering step, which employs large-scale image retrieval methods to generate a list of candidates sorted in decreasing order of probability that the candidate is the subject in the probe. After that, a small number of high probability candidates in the list is provided to the face identification method,

```
Algorithm 2: FaceID_{learn}(X_{n×d}, I_n)
1 $X \leftarrow (X - \mu)/\sigma$
2 $\mathcal{M} \leftarrow \varnothing$
3 for each unique $id \in I$ do
4     $Y \leftarrow \{y_i | y_i = \begin{cases} +1, & \text{if } I_i = id \\ -1, & \text{otherwise} \end{cases}$
5     $W, P, T \leftarrow \mathrm{NIPALS}(X, Y)$
6     Calculate $\beta_{\mathrm{id}}$ using Eq. 3.2
7     $\mathcal{M} \cup \{(\mathrm{id}, \beta_{\mathrm{id}})\}$
```

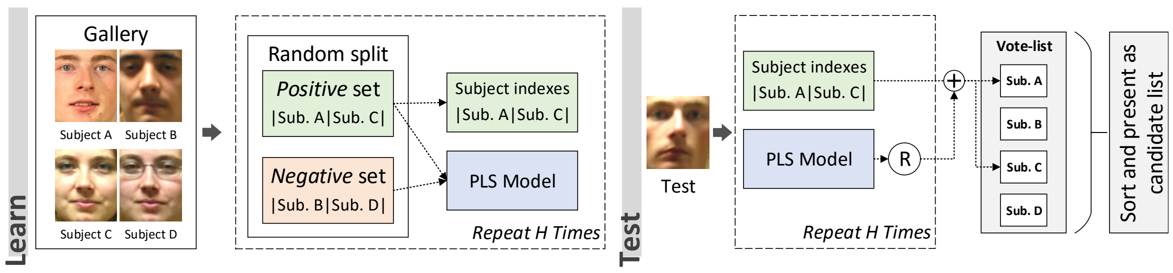
```
Algorithm 3: FaceID_{test}(x_{1×d})
1 $x \leftarrow (x - \mu)/\sigma$
2 max $\leftarrow -\infty$
3 predict $\leftarrow$ “none”
4 for each $(id, \beta_{id}) \in \mathcal{M}$ do
5     score $\leftarrow \beta_{\mathrm{id}}^T x$
6     if $score > max$ then
7         max $\leftarrow score$
8         predict $\leftarrow$ id
9 return predict
```

Figure 3.4: Original algorithm of the PLS face identification as described by Schwartz et al. [2012] with the (left) train and (right) test steps. $X_{n\times d}$ denotes n feature vectors with d dimensions, I_n denotes labels for each feature vector and $x_{1\times d}$ denotes probe sample.

which evaluates subjects following the order in the candidate list until the face identification returns a subject in the face gallery. In this case, speedup is achieved because it is not necessary to evaluate the remaining subjects in the candidate list once a gallery match is found, reducing therefore, the computational cost compared to the brute-force approach. Note that we do not calculate the probability directly but we associate it with the estimated PLS regression values as will be presented.

There are two types of errors in the filtering and face identification pipeline. The first type of error is related to the filtering approach and occurs when the subject in the test sample is enrolled in the gallery, but is not in the candidate list, resulting in a miss for any face identification approach employed afterwards. The second type of error is related to the face identification approach and occurs whenever the test sample subject is within the candidate list but a wrong identity is assigned by the face identification approach.

To evaluate the filtering and face identification pipeline, we consider the face identification method described by Schwartz et al. [2012], which consists in employing a large feature set concatenated to generate a high dimensional feature descriptor. Then, a PLS model is learned for each subject in the gallery following a *one-against-all* classification scheme: samples from the subject are learned with response equal to $+1$ and samples from other subjects with response equal to -1. Test samples are presented to each PLS model and associated to the identity related to the model that returns the maximum score. We consider the evaluation of all PLS models as the brute-force approach and, in the proposed pipeline, only PLS models that correspond to subjects in the candidate list are evaluated. The original algorithm for PLS face identification is presented in Figure 3.4.



Algorithm 4: $\text{PLSH}_{learn}(X_{n,d}, I_n, H)$

```
1  $X \leftarrow (X - \mu)/\sigma$
2  for $h = 1$ to $H$ do
3      $Y \leftarrow [-1, ..., -1]_n$; $\mathcal{P}_h \leftarrow \varnothing$
4      for each unique $id \in I$ do
5          sample $r$ from Bern. $(p = 0.5)$
6          if $r = success$ then
7              $\mathcal{P}_h \cup \{\text{id}\}$
8              $Y = \{y_i | y_i = 1,\ \text{if}\ I_i = id\}$
9      Calculate $\beta_h$ using Eq. 3.2
10     $\mathcal{M} \cup \{(\mathcal{P}_h, \beta_h)\}$
```

Algorithm 5: $\text{PLSH}_{test}(x_{1\times d})$

```
1  $x \leftarrow (x - \mu)/\sigma$
2  $V \leftarrow \{(\text{id}_1, 0), ..., (\text{id}_n, 0)\}$
3  for each $(\mathcal{P}_h, \beta_h) \in \mathcal{M}$ do
4      score $\leftarrow \beta_h^T X$
5      $V \leftarrow \{(i, s) | s = s + \text{score},\ \text{if}\ i \in \mathcal{P}_h\}$
6  $V \leftarrow \{(i, s) | s > 0\}$
7  sort $(i, s) \in V$ in decreasing order of $s$
8  return V
```

Figure 3.5: Overview of PLS for face hashing (PLSH) with (left) train and (right) test algorithms. $X_{n,d}$ denotes samples, I_n are labels, H is number of hash models, $x_{1\times d}$ denotes probe sample and $\mathcal{M}$ denotes hash model set (initially empty).

3.3 Partial Least Squares for Face Hashing (PLSH)

The PLSH method is based on two principles: (i) data dependent hash functions and (ii) hash functions generated independently among each other. Data dependent hash functions provide better performance in general (see discussion in Section 2.2.2). Hash functions generated independently are necessary to induce uniform distribution of binary codes among subjects in the gallery [Joly and Buisson, 2011]. A diagram and algorithm for PLSH is presented in Figure 3.5.

PLSH consists of two steps: *train* and *test*. In the train, we randomly split subjects in the gallery in two balanced subsets: *positive* and *negative*. The split is performed as follows. For each subject, we sample from a Bernoulli distribution with parameter $p = 0.5$ and associate the subject to the positive subset in case of "success". PLS regression models are used to determine whether a test sample belongs to the positive or to the negative subset. In this case, the PLS regression model is learned considering feature descriptors extracted from samples in the positive set with target values equal to $+1$ against samples in the negative set with target values equal to -1. This process is repeated several times[1].We define a PLSH *hash model* as a PLS model and the subjects in the positive subset.

[1]We repeat 150 times on FERET and 10-35 on FRGC datasets.

In the test, we extract the same feature descriptors employed on the train for the test sample (probe sample) and presented them to each PLSH hash model to obtain a regression value r. We define a vote-list of size equal to the number of subjects in the gallery initially with zeros, then, each position of the vote-list is increased by r according to the indexes of subjects in the positive subset of the same PLSH hash model. Note that this scheme allows us to store half of the subject indexes to increment the vote-list since it will be equivalent to increment subjects in the negative set by $|r|$ when r is negative (the differences among pairs of votes will be the same). Finally, the list of subjects is sorted in decreasing order of values and presented as candidates for the face identification.

In practice, the majority of subjects with low values in the candidate list are discarded because they rarely corresponds to the test sample. The candidate list only serves to indicate the evaluation order for the face identification method. In this case, if an identity is assigned to the probe when evaluating the first candidates in the list, there is no need to evaluate the remaining candidates.

PLSH is similar to the work of Joly and Buisson [2011], in which SVM classifiers are employed to determine each bit in the Hamming embedding. The advantage of employing PLS in this case is the robustness to unbalanced classes and support for high dimensional feature descriptors [Santos Jr and Schwartz, 2014]. We do not provide approximation bounds to PLSH as LSH methods because PLSH is based on regression scores rather than distance metrics, which are not compatible with the LSH framework.

3.3.1 Consistency

The motivations for the steps presented in the PLSH algorithm are the following. In one hand, if r is approximately equal to $+1$ in the test, the probe sample is more similar to the subjects in the positive subset and the positions in the vote-list corresponding to the subjects in the positive subset will receive more votes. On the other hand, if r is approximately equal to -1, the votes in the vote-list corresponding to subjects in the positive subset will be decremented. If r is close to zero then the vote-list will not change significantly. Assuming that be equal to $+1$ whenever the correct subject in the test sample is in the positive subset, even if other subjects in the positive subset receive the same vote, their respective votes in the vote-list will be decrement whenever they are not in the same subset as the correct subject. Note that the aforementioned statement holds for a large number of hash functions since the probability of at least two subjects being in the same subsets is negligible. A large number of hash functions also mitigate the problem of a few hash functions not returning r roughly equal to 1 even if the correct subject is in the positive subset and the increase in the number of hash functions is limited only by the computational cost to evaluate them.

3.3.2 Hamming Embedding

We do not estimate the Hamming embedding directly since there is no binary string associated to any face sample. However, PLSH is equivalent to estimating the Hamming embedding for a test sample and comparing it with the binary strings generated for each subject in the gallery. In addition, each bit of the test binary string is weighted by the absolute value of the PLS regression response. To demonstrate the aforementioned claims, consider that PLS responses can be only $+1$ or -1, such that any test sample can be represented by the sequence $X = \{+1, -1\}^H$, where H denotes the number of PLSH hash models. Consider also that each subject s in the face gallery is represented by the binary string $Y_s = \{1, 0\}^H$, where $y_i \in Y_s$ is set to 1 if the subject s was associated to the positive subset of the i-th PLSH hash model in the train step, or 0, otherwise. In this context, the weight w_s given by PLSH to each subject in the gallery is calculated as

$$w_s = \sum_{i=1}^{H} x_i y_i .$$

Note that the maximum w_s is equal to the sum of $+1$ elements in X, which occurs when $y_i = 1$, if $x_i = +1$, and $y_i = 0$, otherwise. Similarly, the minimum weight is equal to the sum of -1 elements in X, which occurs when $y_i = 1$, if $x_i = -1$, and $y_i = 0$, otherwise. If we transform X onto a binary string $\hat{X}$ such that $\hat{x}_i = 1$, if the corresponding x_i is $+1$, and $\hat{x}_i = 0$, otherwise; we can calculate the Hamming distance between $\hat{X}$ and Y_s. In fact, the exactly same Hamming distance can be calculate using w_s as

$$d(X, Y)_{\mathbb{H}} = w_{\max} - w_s, \tag{3.3}$$

where $w_{\max}$ denotes maximum possible w_s. The same analogy can be applied to the weighted Hamming distance (see Equation 2.1) if we consider x_i assuming any real number. In this case, the weight of each bit α_i is the absolute value of r and the weighted Hamming distance is equivalent to Equation 3.3.

3.3.3 Computational Requirements

The time and space complexities of the PLS algorithm are presented as follows. The amount of space necessary for the PLS algorithm depends on the number of hash models H, the dimensionality of the data D and the number of subjects in the gallery

N. Each hash model holds a PLS regression vector in R^d and the indexes of subjects in the positive subset, given by $N/2$, therefore, $H \times D$ real numbers and $(H \times N)/2$ integer indexes of space are necessary using H hash functions in the PLS algorithm. Note that it is not necessary to store the feature vectors used to train the PLS models in the test and they can be safely discarded since the PLS regression vector holds the necessary information to discriminate among the enrolled subjects.

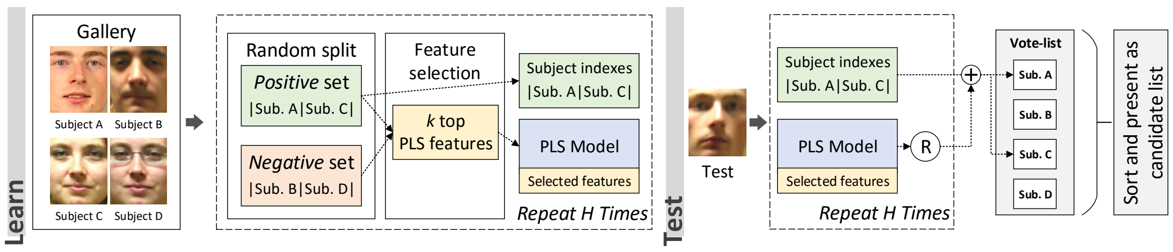
3.3.4 Alternative Implementations

In principle, some aspects of the PLSH algorithm can be changed such that PLSH can provide potential performance improvement. For instance, the parameter p of the Bernoulli distribution used to determine the subsets of subjects may be changed given that PLS hardly finds common discriminative features among subjects in a large set Santos Jr and Schwartz [2014]. However, changing p from 0.5 to other value results in a nonuniform distribution of subjects among subsets (raise hash table collisions), therefore, reducing the accuracy. As will be demonstrated in the experiments, maintaining a balanced subset of subjects to learn each hash model ($p = 0.5$) provide the best results.

Another possible implementations of PLSH that does not modify much the results is the product of votes instead of the sum, which is akin to the intersection of subsets among all hash functions. It is also possible to employ multiple partitions instead of only two by using a categorical rather than Bernoulli distribution. However, multiple partitions provide no significant difference in the results and they require twice the space requirement since the indexes of subjects that were learned with $+1$ target response in the PLS model need to be stored to allow them to receive the votes in the test. At last but not least, the computational cost to evaluate the hash functions can be reduced when considering feature selection methods, which consists in calculating the PLS regression response using few dimensions in the feature vector corresponding to the most important to discriminate between the subject subsets. As will be presented, the feature selection include a new parameter in the PLSH algorithm, the number of features selected, which can be estimated jointly with the number of hash functions to provide much better results than in PLSH without feature selection.

3.4 Feature Selection for Face Hashing (ePLSH)

The algorithms for PLSH described in Section 3.3 require a dot product between the PLS regression vector and the feature descriptor to calculate each hash function. This



Algorithm 6: ePLSH$_{learn}(X_{n\times d}, I_n, H, k)$

```
1  $X \leftarrow (X - \mu)/\sigma$
2  **for** $h = 1$ ***to*** $H$ **do**
3      $Y \leftarrow [-1, ..., -1]_n$; $\mathcal{P}_h \leftarrow \varnothing$
4      **for *each unique*** $id \in I$ **do**
5          sample $r$ from Bern. $(p = 0.5)$
6          **if** $r = success$ **then**
7              $\mathcal{P}_h \cup \{\text{id}\}$
8              $Y = \{y_i | y_i = 1,\ \text{if } I_i = id\}$
9      **$F_h \leftarrow k$ discriminative features**
10     **select features $F_h$ from $X$**
11     Calculate $\beta_h$ using Eq. 3.2
12     $\mathcal{M}_e \cup \{(\mathcal{P}_h, \beta_h, F_h)\}$
```

Algorithm 7: ePLSH$_{test}(x_{1\times d})$

```
1  $x \leftarrow (x - \mu)/\sigma$
2  $V \leftarrow \{(\text{id}_1, 0), ..., (\text{id}_n, 0)\}$
3  **for *each*** $(\mathcal{P}, \beta_h, F_h) \in \mathcal{M}_e$ **do**
4      **select features $F_h$ from $x$**
5      score $\leftarrow \beta_h^T x$
6      $V \leftarrow \{(i, s) | s = s + \text{score, if } i \in \mathcal{P}\}$
7  $V \leftarrow \{(i, s) | s > 0\}$
8  sort $(i, s) \in V$ in decreasing order of $s$
9  **return** V
```

Figure 3.6: Overview of PLS for face hashing and feature selection (ePLSH) with (left) train and (right) test algorithms (lines different from PLSH algorithm are in blue and bold). $X_{n,d}$ denotes samples, I_n are labels, H is number of hash models, k is the number of features selected, $x_{1\times d}$ denotes probe sample, $\mathcal{M}_e$ denotes hash model set (initially empty).

section describes methods to reduce the computational cost to evaluate hash functions. To discriminate PLSH with the feature selection version and to maintain consistence with the nomenclature given in our publications, PLSH with feature selection is called *extended PLSH* (ePLSH) in the rest of this work. In practice, ePLSH is equivalent to PLSH when all features are considered to evaluate hash functions. The main advantage of ePLSH is the possibility of employing thousands of additional hash functions, resulting in considerable increase of the recognition rate while keeping low computational cost to calculate the hash functions. The common feature setup considered in the PLSH and in the ePLSH approaches consists in combining four feature descriptors, which leads to a feature vector with 120,059 dimensions. However, we show in our experiments that, for the feature set considered in this work, about 500 dimensions with an increased number of hash functions provides better candidate lists than PLSH with about the same computational cost. A summary of ePLSH is presented in Figure 3.6.

The ePLSH consists of two steps: *train* and *test*. In the train, it calculates the β regression vector following the same procedure of PLSH. Then, the indexes of the k more discriminative features are stored. Considering that the range of values in the feature vector is known (zero mean and unit variance in our experiments), it is possible to calculate an approximated score using only the more discriminative features. However, if only such features are used to calculate the regression value without rebuilding the PLS model, the result would not be accurate because of the large number of remaining features, even though they present a very low contribution individually. To tackle this issue, we learn a new PLS model to replace the full feature version in PLSH, which is performed by eliminating the dimensions from the matrix X that do not correspond to the k select features and recalculate β using Equation 3.2. We define the ePLSH *hash model* as the PLS model, the subjects in the positive subset and the k selected features. Finally, the test step is carried in the same manner as in PLSH, but with the difference that only features selected in the ePLSH hash model are considered to calculate the regression score.

The difference between PLSH and ePLSH algorithms is the feature selection step of lines 9 and 10 in Algorithm 6 and line 4 in Algorithm 7. There are numerous methods for feature selection and transformation in the literature and, due to performance reasons, it is preferable to employ feature selection methods so the features can be directly indexed when calculating the dot product in the test step. If we employ feature transformation such as PCA, it will be necessary a multiplication between the PCA projection matrix and the feature vector, which is expensive in terms of computational time. In early experiments, we considered employing PCA before providing X to Algorithms 6 and 7. Although slightly difference in accuracy was observed when keeping 95% of variance explained (roughly 1% of the dimensions in the feature vector), the computational cost to project the features was not attractive because of the high dimensional feature vectors. Therefore, we consider feature selection methods.

There are numerous works about PLS-based feature selection in the literature and they are divided in three categories [Mehmood et al., 2012]: *filter*, *wrapper* and *embedded*. Filter methods are the simplest of the three and work in two steps. First, the PLS regression model is learned and, then, a relevance measure based on the learned PLS parameters is employed to select the most relevant features. Wrapper methods consist in an iterative filter approach coupled with a supervised feature selection method. Finally, embedded methods consist in nesting feature selection approaches in each iteration of the PLS algorithm. We suggest the work presented by Mehmood et al. [2012] for a comprehensive list and description of PLS feature selection methods.

In this work, we focus on PLS filter methods for feature selection for simplicity reasons. However, ePLSH is defined without lost of generality such that other PLS feature selection methods could be considered if necessary. Mehmood et al. [2012] describe three filter methods called loading weights (W), variable importance on pro-

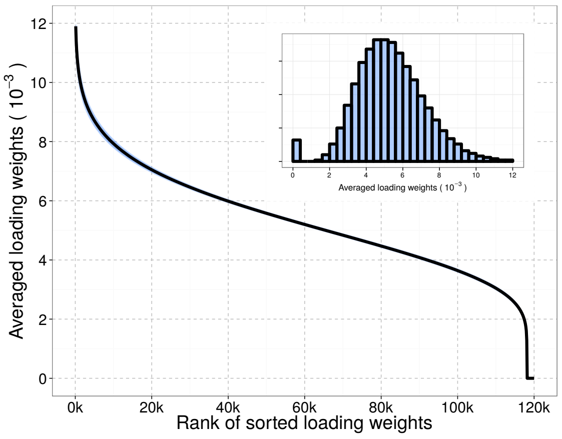


Figure 3.7: Loading weights distribution for a 120,059-dimensional feature vector (small plot) and ranked in decreasing order (big plot). The feature vector is a combination of CLBP, HOG, Gabor and SIFT feature descriptors.

jection (VIP) and regression coefficients (β). These methods are described in Sections 3.4.1, 3.4.2 and 3.4.3, respectively.

3.4.1 Loading Weights

The idea in the loading weight approach is, for each PLS component, to select the features associated with higher absolute w_i value (alternately features above a threshold [Mehmood et al., 2012]). Recall W being the output of NIPALS algorithm[2] used to calculate latent variables. In this way, the absolute coefficient $w_{f,i} \in W$, for the f-th PLS component and the i-th feature, is directly associated to the f-th latent variable. Note that one feature may be relevant to one PLS component and irrelevant for another, specially because the latent variable basis represented by W is orthonormal. Therefore, the goal is to find the set of features that are relevant to calculate at least one PLS latent variable. In this context, the loading weight method consists in selecting features $i \in [1, N]$ with highest relevance measure defined as $\max_{f=1:p}(w_{p,i})$. The distribution of loading weights is presented in Figure 3.7, where it can be seen that the distribution of weights among the dimensions is roughly normal, with mean around 5×10^{-3}, and a little skewed toward small values. As will be presented in the experiments, the 500 dimensions with higher weights (loading weights higher than about 10^{-2}) are enough to provide good results.

[2]see Section 3.1 for the PLS description

3.4.2 Variable Importance on Projection

Variable importance on projection (VIP) consists in calculating a measure that summarize the loading weights (W) of all factors for each dimension in the feature vector. VIP measure is calculated as[3]

$$v_i = \sqrt{n \sum_{f=1}^{p} \left(b_f^2 w_{f,i}^2 \right) / \sum_{f=1}^{p} b_f^2}. \tag{3.4}$$

In our experiments, the product by n can be ignored since n is constant for all features. In this case, the VIP measure will not be normalized and the common VIP threshold described in [Mehmood et al., 2012], which determines that relevant features present VIP higher than 0.8, cannot be employed directly. Recall b_i as proportional to the covariance between projected features and target values. The sum in the numerator of Equation 3.4 represents the squared sum of the loading coefficients for a specific feature weighted by the predictive capacity of each coefficient. In this way, the main difference between the loading weights and the VIP approaches is the employment of b_i in the latter. The distribution of VIP is presented in Figure 3.8a, which is similar to the loading weights distribution and resemble a normal distribution skewed toward lower values.

3.4.3 Regression Coefficients

Regression coefficients for feature selection is the simplest of the three filter methods and consists in using the regression vector directly to select the most relevant features. Recall from Section 3.1 the regression vector as $\beta = W(P^TW)^{-1}(T^TT)^{-1}T^TY$, where P and T are loading matrices from features and target values, respectively. Similar to the loading weights approaches, regression coefficients are also related to predictive capacity of the latent variables to estimate target values, however, in a more transparent manner since they are directly employed to estimate the regression values. The distribution of absolute regression coefficients, presented in Figure 3.8b, is considerable different from the other approaches distribution. The regression coefficients distribution resembles a normal distribution with zero mean, which the absolute values results in the distribution in Figure 3.8b. The main difference from the regression coefficient and the aforementioned filter approaches is the correlation of the latent variables with target values embedded in the PLS regression vector, which provides a small improvement over the loading weights and VIP results.

[3]see Section 3.1 for variable definitions.

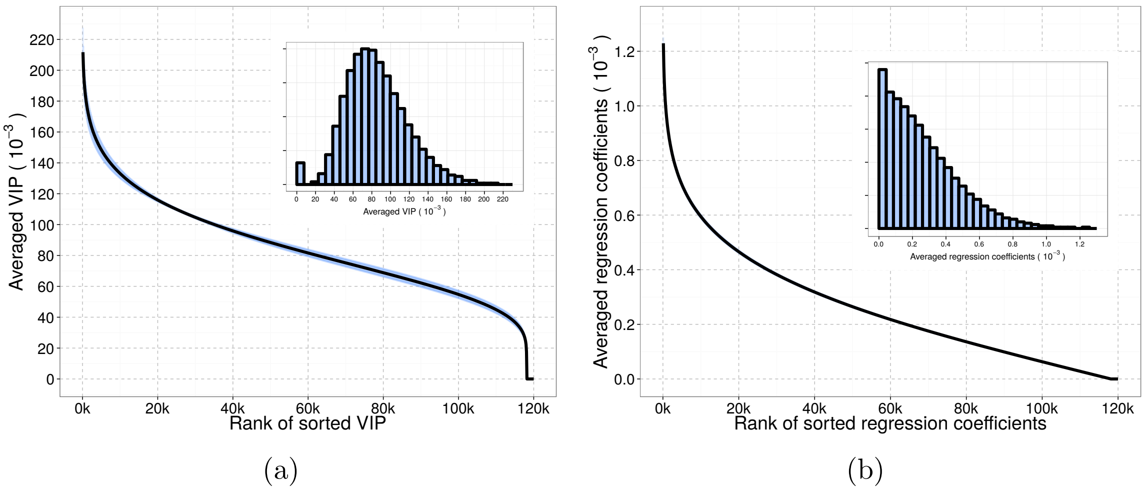


Figure 3.8: (a) VIP measure and (b) Regression coefficient distribution for a 120,059-dimensional feature vector (small plot) and ranked in decreasing order (big plot).

3.5 Early-Stop Search Heuristic

To stop the search for the correct subject in the candidate list, we employ the heuristic described by Schwartz et al. [2012]. For a short number of initial samples (15), all subjects in the candidate list are evaluated and the median value of the scores is taken as threshold for the remaining test samples. Then, subjects in the candidate list are evaluated until a score equal or higher than the threshold is obtained or the end of the list is reached.

Note that, in practice, the candidate list size is a percentage of the subjects enrolled in the gallery and most of the candidates with low weights can be discarded because they rarely corresponds to the probe sample. In this case, the worst case scenario consists in evaluating all subjects in the candidate list for every probe sample. However, the early-stop search heuristic alone is shown to reduce the number of tests in the face identification up to 63% without degrading the recognition rate so the speedup achieved is usually higher than the ratio of the gallery size divided by the number of subjects in the candidate list.

Chapter 4

Experimental Results

In this chapter, we evaluate PLSH and ePLSH in two standard face identification datasets (FERET and FRGCv1). Section 4.1 contains the common experimental setup, including datasets, face identification parameters, evaluation metric, description of the computer used in the experiments, and feature descriptors. The PLSH parameters are presented in Section 4.2. The parameters for ePLSH are discussed in Section 4.3. Evaluation on the datasets and comparisons with other methods in the literature are presented in Section 4.4 (FERET) and in Section 4.5 (FRGCv1).

4.1 Experimental Setup

All experiments regarding parameter validation in Sections 4.2 and 4.3 were performed on the FERET dataset, since it is the dataset with the largest number of subjects ($1,196$ in total). FERET consists of four test sets and we use *dup2* in Sections 4.2 and 4.3, which is considered the hardest of the dataset. The only exception is the experiment regarding the number of hash models and the gallery size in Section 4.3.4, where *fb* test set was employed since it provides more test samples ($1,195$) than the others (194, 722 and 234 in *fc*, *dup1* and *dup2*, respectively).

All experiments were conducted using an Intel Xeon X5670 CPU with 2.93 GHz and 72 GB of RAM running Ubuntu 12.04 operating system. All tests were performed using a single CPU and no more than 8 GB of RAM were necessary.

The FERET and FRGCv1 datasets are described in Sections 4.1.1 and 4.1.2, respectively. In Section 4.1.3, we describe the evaluation metric. In Section 4.1.4, we discuss the number of dimensions in PLS regression for the face identification and the face indexing approaches. Feature descriptor parameters are presented in Section 4.1.5.

4.1.1 FERET Dataset

The FERET dataset [Phillips et al., 2000] consists of 1,196 images, one per subject for training, and four test sets designed to evaluate the effects of lightning conditions, facial expression and aging on face identification methods. The test sets are:

- **fb**: consisting of 1,195 images taken with different facial expressions.
- **fc**: consisting of 194 images taken in different lightning conditions.
- **dup1**: consisting of 722 images taken between 1 minute and 1,031 days after the gallery image.
- **dup2**: is a subset of *dup1* and consists of 234 images taken 18 months after the gallery image.

In our experiments, all images were cropped in the face region using annotated coordinates of the face, scaled to 128×128 pixels and normalized using the self-quotient image (SQI) method to remove lightning effects [Wang et al., 2004].

4.1.2 FRGC Dataset

The face recognition grand challenge dataset (FRGC) [Phillips et al., 2005] consists of 275 subjects and samples that include 3D models of the face and 2D images taken with different illumination conditions and facial expressions. We follow the same protocol described by Yuan et al. [2005], which considers only 2D images and consists in randomly selecting different percentages of samples from each subject to compose the face gallery and using the remaining samples to test. The process is repeated five times and the mean and standard deviation of the rank-1 recognition rate and speedup (considering the brute-force approach) are reported. The samples were cropped in the facial region, resulting in size 138×160 pixels, and scaled to 128×128 pixels.

4.1.3 Evaluation Metric (MARR)

Recall from the face identification pipeline presented in Section 3.2 that the candidate list calculated in the filter approach (PLSH and ePLSH) is employed to reduce the number of PLS models evaluated in the face identification. In this context, the error rate of the pipeline results from errors induced by the filter approach (fail to return identity of test sample in the candidate list) and by the face identification approach (fail to identify correctly the subject in the candidate list). Therefore, to assess the

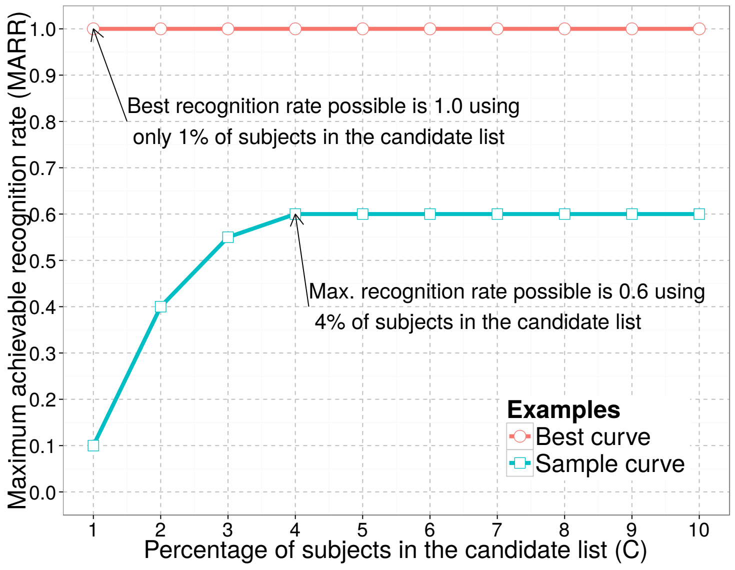


Figure 4.1: An example of the plots regarding the MARR evaluation metric for two sample curves. The MARR metric (vertical axis) considers that the candidate list is presented to an ideal face identification method, therefore, providing the upper bound of the recognition rate achievable when considering a given filtering method such as PLSH and ePLSH. The horizontal axis presents different percentages of the candidate list that are presented to the face identification approach. The best curve presents MARR equal to one for any percentage of subjects in the candidate list.

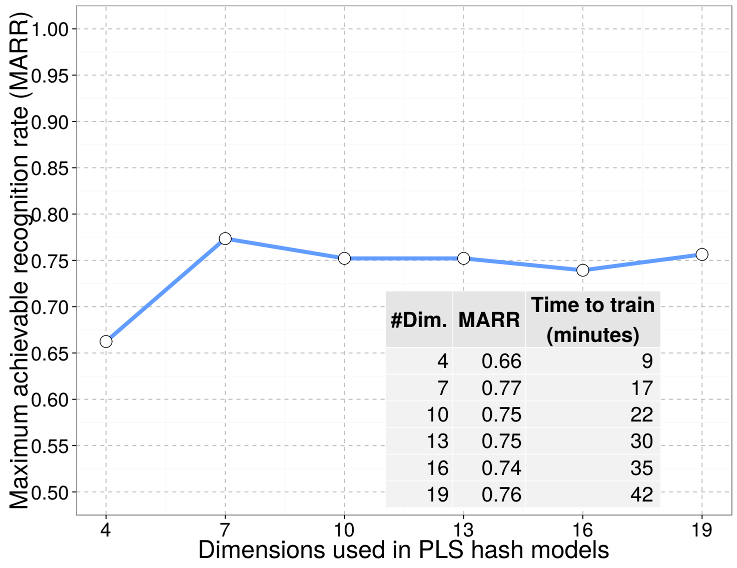
performance of the filter approach alone, we provide results considering the *maximum achievable recognition rate* (MARR), which is calculated considering that a perfect face identification method is employed for different percentages of candidates visited in the list.

Note that the MARR value is the upper bound for the recognition rate achieved by the filter and face identification pipeline. Figure 4.1 illustrates the MARR evaluation metric where better results present MARR close to one and low percentage of candidates visited (curves close to the upper left corner of the plots).

4.1.4 Number of Dimensions in the PLS models

PLS-based face identification requires only one parameter, which is the number of dimensions in the PLS latent space (p). Schwartz et al. [2012] evaluated p by varying it from 13 to 21 without noticing large variation in the results. Therefore, we set p to 20 for the face identification method in our experiments.

We conducted experiments in PLSH by varying p between 4 and 19, in steps of 3, and the results are presented in Figure 4.2 along with the time spent to train 150 hash models. Although p equal to 7 provides the best MARR and lowest time to



#Dim.	MARR	Time to train (minutes)
4	0.66	9
7	0.77	17
10	0.75	22
13	0.75	30
16	0.74	35
19	0.76	42

Figure 4.2: Number of dimensions in PLS hash models and time spent to train 150 hash models.

train, we noticed that the MARR tends to vary between 0.74 and 0.79 because of the nondeterministic nature of the PLSH algorithm. In this way, MARR for p between 7 and 19 may be seen as roughly the same and we consider p equal to 10 in the PLSH and ePLSH experiments.

4.1.5 Feature Descriptors

In this section, we describe the common feature descriptors employed in this work. In total, we evaluate 12 feature descriptors, namely BRISK, CLBP, color histogram, FREAK, Gabor filters, HOG, mean intensity values, LBP, ORB, POEM, SIFT and SURF. Employing the combination of all 12 feature descriptors is intractable considering the computational time spent to calculate the features and the memory necessary to store the hash model, which depends on the dimensionality of the data. Therefore, only the best four feature descriptors (CLBP, Gabor filters, HOG and SIFT) are considered in all experiments. The 12 feature descriptors are considered only in Section 4.2.1 to evaluate their individual performance to define the four best.

4.1.5.1 BRISK

Proposed by Leutenegger et al. [2011], binary robust invariant scalable keypoints (BRISK) has been mainly used for object detection and recognition based on points of interest. BRISK is fast to calculate and to compare since the description is in the

Hamming space. In our experiments, the neighborhood size of the BRISK parameter is set to 512 and we consider the concatenation of 169 binary descriptors sampled equally in the face image. The resulting feature descriptor has $10,816$ variables with average calculation time of 34 milliseconds. We use the implementation available in the OpenCV programming language library [Bradski, 2000].

4.1.5.2 CLBP

Local binary patterns with circular neighborhood (CLBP) mainly capture information regarding texture in the face image. CLBP is an extension of the original LBP and includes a parameter to control the radius around the central pixel by which neighbors are compared [Ahonen et al., 2006]. We set the radius parameter to 5, which is the common parameter employed in face recognition tasks. CLBP histograms are calculated in a sliding window approach with size equal to 16 pixels and stride equal to 8 pixels. We also consider accumulating all *normal codes* (codes with more than 2 transitions between bits) in the same histogram bin to reduce the dimensionality. The final descriptor is the concatenation of all histograms in the face image, resulting in $9,971$ dimensions and taking 118 milliseconds, on average, to calculate.

4.1.5.3 LBP

Local binary patterns is equivalent to CLBP (see Section 4.1.5.2) with radius parameter equal to one. However, in this case, we are evaluating only the resulting image after applying the LBP operator in the face image, i.e., without calculating LBP histograms. The final feature descriptor has $49,980$ dimensions and average time to calculate equal to 570 milliseconds per face image.

4.1.5.4 Color Histogram

Color histogram is a simple global descriptor that accumulates pixel values in a histogram. We calculate the histograms using sliding window with size equal to 16 pixels and stride equal to 8 pixels. The color space considered was the HSV and only the hue and the saturation channels were considered. The number of bins is set to 20, in the hue channel, and 32, in the saturation channel. The final descriptor is the concatenation of all histograms in the face image, resulting in $31,360$ dimensions with average calculation time of 13 milliseconds.

4.1.5.5 FREAK

Fast retina keypoint (FREAK), proposed by Alahi et al. [2012], is a SIFT-like feature descriptor that focuses on fast computation based on human eye saccadic search. As in BRISK, we set the number of bits in the descriptor equal to 512 and concatenate the descriptors calculated in 121 keypoints sampled equally in the face image. The final feature descriptor has size $7,744$ with average time to calculate equal to 670 milliseconds per face image. We use the implementation available in the OpenCV programming language library [Bradski, 2000].

4.1.5.6 Gabor filters

Gabor filters mainly capture texture information and is commonly employed in face recognition tasks [Randen and Husoy, 1999]. The main disadvantages of Gabor filters is the high computational cost to calculate and store. To compute Gabor filters, we convolve the face image with filters of size 16×16 pixels, 8 scales, equally distributed between $[0, \frac{\pi}{2}]$, and 5 orientations, equally distributed between $[0, \pi]$, which results in 40 convolved images. The images were downscaled by a factor of 4 and concatenated to assemble the final feature descriptor, resulting in $40,960$ dimensions and taking $1,475$ milliseconds to calculate per face image, on average.

4.1.5.7 HOG

Histograms of oriented gradients (HOG) was first employed for pedestrian detection and captures mainly information regarding shape in the face image [Dalal and Triggs, 2005]. Two feature setups are considered for HOG. The first setup consists in block size equal to 16×16 pixels, stride equal to 4 pixels and cell size equal to 4×4 pixels. The second setup consists in block size equal to 32×32 pixels, stride equal to 8 pixels and cell size equal to 8×8 pixels. The feature descriptor consists in concatenating the HOG descriptors from the two setups, resulting in $36,360$ dimensions and taking 81 milliseconds to calculate per face image, on average. The HOG implementation considered in this work is the one available in the OpenCV programming language library [Bradski, 2000].

4.1.5.8 Mean Intensity

Mean intensity was used by Schwartz et al. [2012] as a simple feature descriptor to capture intensity values in the face image. It consists in averaging the intensity values of a sliding window in the face image. Although simple, it provides fair accuracy and complementarity to descriptors based on shape and texture. We follow the same parameter setup provided in [Schwartz et al., 2012], with window size equal to 4 pixels and stride equal to 2 pixels. The descriptor size is $3,969$ with an average time to calculate equal to 103 milliseconds per face image.

4.1.5.9 ORB

Oriented FAST and rotated BRIEF (ORB), proposed by Rublee et al. [2011], is a SIFT-like feature descriptor that focuses on fast computation and matching, similarly to BRISK, FREAK and SURF. ORB consists in rotating the patch to a canonical orientation according to a moment equation and, then, calculating the BRIEF descriptor in the rotated patch. In our experiments, ORB is extracted in 256 keypoints evenly spaced in the face image and the number of bits in the BRIEF descriptor is set to 512. The final descriptor has $2,592$ dimensions and with an average time to calculate equal to 8 milliseconds per face image. We use the ORB implementation available in the OpenCV programming language library [Bradski, 2000].

4.1.5.10 POEM

Patterns of oriented edge magnitudes (POEM) was proposed by Vu et al. [2012] and consists in the CLBP operator applied to magnitude of gradients in different orientations. The histograms of gradient orientations are calculated in a cell with size of 7 pixels. Each histogram has 3 bins, which accumulate unsigned orientation values between 0 and π radians. The final feature descriptor has $29,913$ dimensions and present average calculation time equal to 306 milliseconds per face image.

4.1.5.11 SIFT

Scale-invariant feature transform (SIFT) [Lowe, 2004] is the most popular descriptor for object detection and recognition in the literature and several other feature descriptors are inspired by it [Rublee et al., 2011; Dalal and Triggs, 2005; Alahi et al., 2012; Leutenegger et al., 2011]. SIFT consists of two parts: keypoint detection and description. We consider SIFT descriptors calculated in 256 keypoints evenly spaced in the face image. We employed the default parameters proposed by Lowe [2004], which are 4×4 histogram cells, each with 8 bins, contrast threshold 0.04, Gaussian smoothness 1.6 and edge threshold 10. The final feature descriptor is the concatenation of all SIFT descriptors in the face image and has $32,768$ dimensions with average time to calculate equal to 30 milliseconds. We use the OpenCV programming library SIFT implementation [Bradski, 2000].

4.1.5.12 SURF

Speeded-up robust features (SURF) [Bay et al., 2008] is a SIFT-like descriptor and, similarly to SIFT, is largely employed in the literature for object detection and recognition. SURF can be seen as an approximation of SIFT descriptors, which employs Haar-like features and integral images to reduce computational time to calculate the feature descriptor. We employ the standard SURF parameters in 256 keypoints evenly spaced in the face image which, when concatenated, results in a $16,384$-dimensional feature vector with average calculation time of 29 milliseconds. We use the SURF implementation available in the OpenCV programming library [Bradski, 2000].

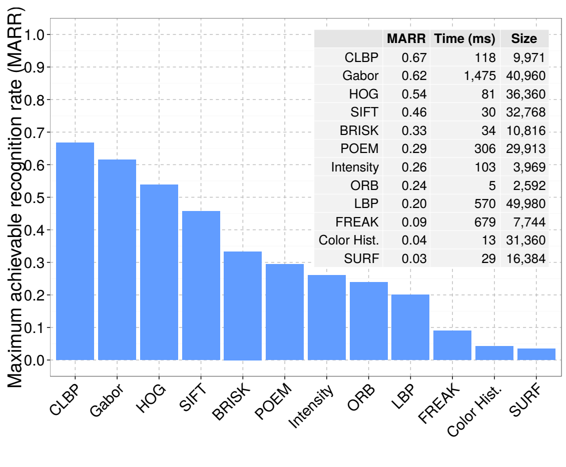
4.2 PLSH Parameter Validation

Herein we evaluate the aspects regarding PLSH model and parameter selection. In Section 4.2.1, we evaluate 12 feature descriptors and the combination of the four best. In Section 4.2.2, we evaluate different numbers of hash models. In Section 4.2.3, we discuss and evaluate whether to employ balanced partitions. In Section 4.2.4, we discuss and evaluate different number of partitions. In Section 4.2.5, we discuss and evaluate whether to employ sum or product when combining votes in the PLSH framework. In Section 4.2.6, we characterize the response of the vote-list, which is useful to discard subjects in the candidate list in linear time with the size of the list. Finally, in Section 4.2.7, we discuss stability regarding PLSH results.

4.2.1 Combination of Different Feature Descriptors

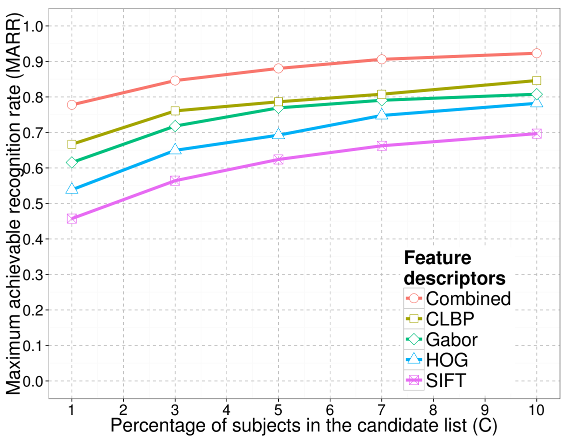
Figure 4.3a presents MARR for different feature descriptors when 1% of the candidate list is provided for identification. We consider the binary feature descriptors in the real domain such that the PLS regression can be directly employed. The number of hash models in this experiment was empirically set to 150. For each feature descriptor, a summary of the averaged time spent to calculate and the size of the feature vector is provided in the table of Figure 4.3a. The feature descriptors are sorted in decreasing order of MARR and the four best feature descriptors are CLBP, Gabor filters, HOG and SIFT. In Figure 4.3b, we evaluate these four feature descriptors with their combination and for different percentages of subjects in the candidates list.

According to Figure 4.3b, the combination of CLBP, Gabor, HOG and SIFT is responsible for an increase of about 10 percentage points (p.p.) in MARR compared to the best individual feature descriptor (CLBP). Therefore, we employ the combination



	MARR	Time (ms)	Size
CLBP	0.67	118	9,971
Gabor	0.62	1,475	40,960
HOG	0.54	81	36,360
SIFT	0.46	30	32,768
BRISK	0.33	34	10,816
POEM	0.29	306	29,913
Intensity	0.26	103	3,969
ORB	0.24	5	2,592
LBP	0.20	570	49,980
FREAK	0.09	679	7,744
Color Hist.	0.04	13	31,360
SURF	0.03	29	16,384

(a)



(b)

Figure 4.3: (a) Single features when 1% of the candidate list is provided for identification and (b) best four single features compared to their combination.

of these feature descriptors in the remaining experiments. The combined feature descriptor has $120,059$ dimensions with averaged time to calculate equal to 1.7 seconds. It is important to point out that the time spent to calculate the feature descriptors for a probe sample is constant (it does not depend on the number of subjects enrolled in the face gallery). In fact, the computational time to extract the feature descriptors can be adapted in exchange for reduced MARR. For instance, Gabor filters could be discarded to reduce the computational time to extract the features since they take 1.4 seconds per face image to calculate, on average.

Although BRISK, SURF and FREAK are based on SIFT, these descriptors provide considerable less accuracy than SIFT (at least 10 p.p. of MARR difference). We believe that this difference in accuracy might result from the manner that we are using these descriptors, which is different from the conventional bag-of-words or other representations commonly employed in the literature. BRISK, ORB and FREAK are also binary feature descriptors. Given that PLS considers the Euclidian space instead of Hamming, this might also be the reason for the reduced accuracy presented by these binary feature descriptors.

4.2.2 Number of Hash Models

Figure 4.4 presents MARR for a number of hash models equal to $10, 50, 100, 150$ and 200. According to the results, a large improvement in MARR (for any number of subjects in the candidate list) takes place when the number of hash models increases from 10 to 150 can be seen in Figure 4.4. However, the increase in MARR is negligible

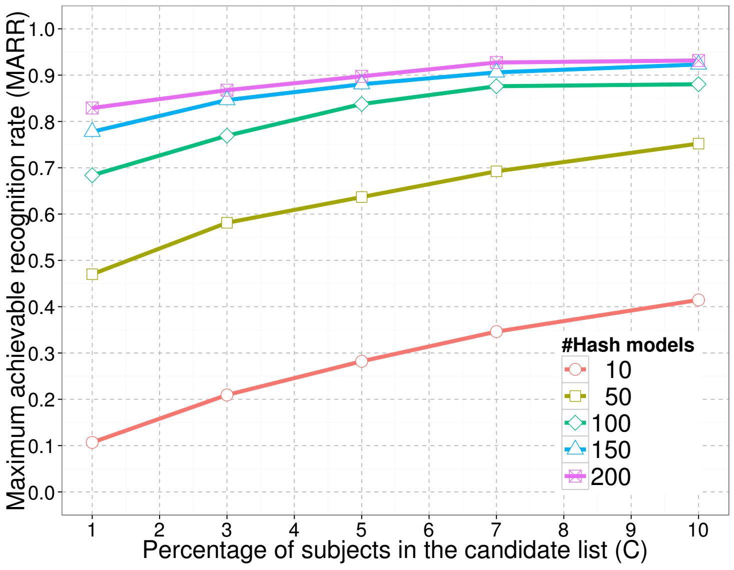


Figure 4.4: Number of hash models as a function of the MARR for different percentages of subjects in the candidate list.

when the number of hash models is raised from 150 to 200. Since the face identification and the PLSH approaches depend on a single dot product between the feature and the PLS regression vectors, the computational cost to evaluate each hash function in PLSH is about the same as the cost to evaluate each subject in the gallery. Therefore, to obtain a low computational cost for testing samples, we consider 150 hash functions in the remaining PLSH experiments. As a reference, the average time to evaluate each hash function in this experiment was 426 microseconds.

4.2.3 Balanced Partitions and Code Distribution

In this section we discuss the best value for the Bernoulli distribution parameter p necessary to distribute binary codes uniformly among subjects in the gallery. Consider that we assign a binary code for each subject, of a total of N, in a gallery such that each code has a number of bits

$$B = log_2(N). \tag{4.1}$$

We learn B hash functions to determine each bit of the code. The goal is to determine the probability of choosing a bit $b \in \{0, 1\}$ for a subject $X = [x_1, ..., x_B]$ such that each final binary code has equal probability, i.e,

$$P(code) = \prod_{i=1}^{B} P(x_i = b) = \frac{1}{N}, \tag{4.2}$$

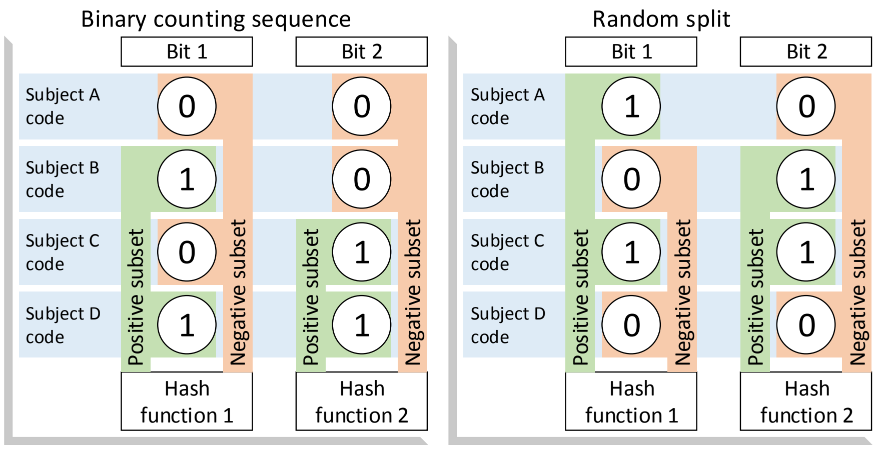


Figure 4.5: Contrast between binary counting sequence and random partition used in the PLSH bit assignment. The binary counting sequence does not generate all possible subsets and depends on the order of subjects presented.

where x_i is drawn independently and identically from a Bernoulli distribution with parameter p so that $P(x_i = 1) = p$.

Note that independence when drawing bits implies in independent hash functions. In the other hand, if we instead assign codes systematically among subjects, e.g., following a binary counting sequence (001, 010, ..., 111), the probability of assigning a specific code to a subject will depend on codes already assigned to other subjects, therefore, breaking independence among the hash functions. If we systematically assign codes to subjects, we also limit the combinatorial number of binary subsets resulting in hash functions biased toward the order of codes assigned to subjects. The advantage of independent hash functions is illustrated in Figure 4.5.

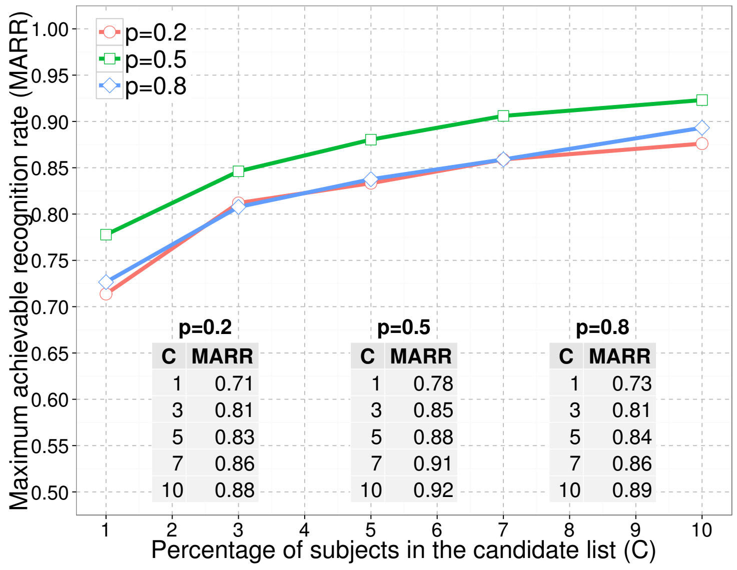
In practice, we learn more than B hash functions to reduce the number of collisions in the hash table when we independently draw bits from a probability distribution. We also expect that some hash functions will miss some bits (change one bit for another). However, considering an unbiased classifier, it is expected a zero sum of r values from the missed bits, such that the final result will be stable.

Considering the Bernoulli distribution, Equation 4.2 is rewritten as

$$P(code) = p^k(1-p)^{B-k} = \frac{1}{N}, \forall k \in \{0, ..., B\}, \tag{4.3}$$

where k is the number of bits in the code that are equal to 1. Expanding Equation 4.3,

$$P(code) = p^B = p^{B-1}(1-p) = ... = (1-p)^B = \frac{1}{N}, \tag{4.4}$$



p=0.2

C	MARR
1	0.71
3	0.81
5	0.83
7	0.86
10	0.88

p=0.5

C	MARR
1	0.78
3	0.85
5	0.88
7	0.91
10	0.92

p=0.8

C	MARR
1	0.73
3	0.81
5	0.84
7	0.86
10	0.89

Figure 4.6: Evaluation of the parameter p used for partitioning subjects in the gallery. The theoretical optimal value for p is 0.5.

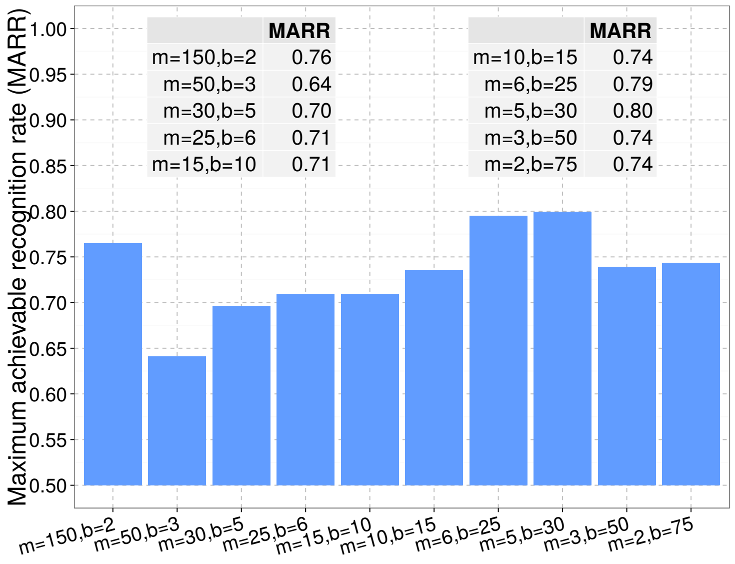
implying in $p = 1 - p = 0.5$. It can also be solved as

$$p^B = \frac{1}{N} \implies log_p(\frac{1}{N}) = log_2(N) \implies p = 0.5. \tag{4.5}$$

It is possible to demonstrate that $p = 0.5$ minimizes the expected number of collisions in the hash table. We conducted experiments with p equal to 0.2 and 0.8 and both values resulted in poor performance. Figure 4.6 presents MARR considering p equal to 0.2, 0.5 and 0.8 where it can be seen that both MARR curves for $p = 0.2$ and $p = 0.8$ are below the curve for $p = 0.5$. Based on the aforementioned discussion, we can conclude that (i) the performance of the proposed approach depends only on how well a classifier can distinguish between two random subsets of subjects and (ii) each subset must contain half of the subjects.

4.2.4 Number of Random Partitions

In principle, we may consider more than two balanced partitions in PLSH with no additional computational cost. The number of partitions (b) will change only the base of the logarithm in Equation 4.1 from 2 to b. In this case, b PLS regression vectors are necessary to determine whether a test sample belongs to a specific partition. It is also necessary to store the subjects in each partition to increment the vote-list when testing. We experimented varying the number of partitions in PLSH using a categorical rather than Bernoulli distribution with equal probability in each category. We also keep the



	MARR
m=150,b=2	0.76
m=50,b=3	0.64
m=30,b=5	0.70
m=25,b=6	0.71
m=15,b=10	0.71

	MARR
m=10,b=15	0.74
m=6,b=25	0.79
m=5,b=30	0.80
m=3,b=50	0.74
m=2,b=75	0.74

Figure 4.7: Evaluation of the partition number.

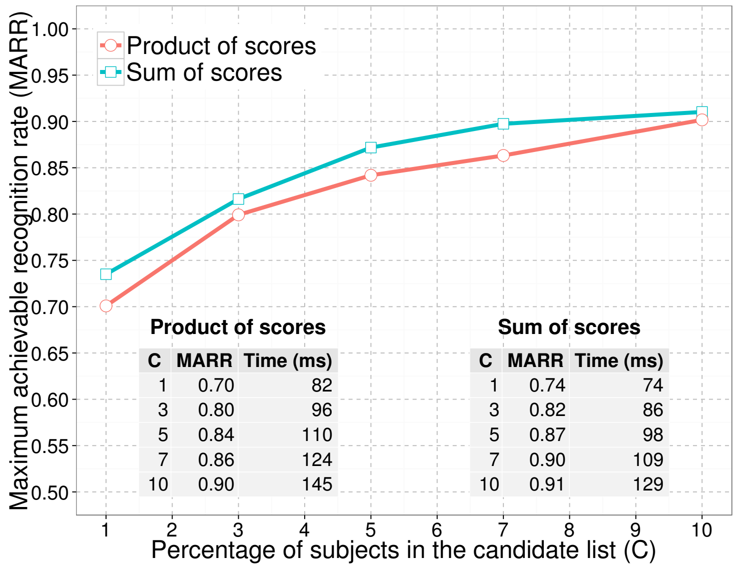
computational cost constant by fixing the number of regression vectors necessary to split samples in b partitions and varying the quantity of hash functions (m).

Figure 4.7 presents MARR for different configurations of m and b and with 1% of subjects in the candidate list. The number of projections is fixed in 150, which returns MARR equal to 0.76 using two partitions ($m = 150, b = 2$). The number of hash functions tested were $m \in \{50, 30, 25, 15, 10, 6, 5, 3, 2\}$. Although the results are not strong enough to discard the use of multiple partitions, no significant increase in MARR compared to the binary partition was found. In conclusion, to keep PLSH simple and easy to employ, we consider the binary partition to remove the need to estimate the parameter b.

4.2.5 Voting Scheme

The voting scheme allows sorting subjects in the gallery according to the association of the test sample to the positive and negative subsets. In this case, there are two intuitive approaches to associate scores to subjects in the vote-list. One approach consists in adding scores, as suggested in PLSH. The other approach consists in multiplying scores, which is akin to the intersection of subjects in the subsets that include the test sample. In this section, we discuss the implementation of the latter approach and compare it with the addition of scores.

The product of scores is carried as follows: Let $m_j = \{h(x), \mathcal{P}, \mathcal{N}\}$ be the j-th hash model with the regression function $h(x) \to \mathcal{R}$, subjects in the positive $\mathcal{P}$ and



Product of scores

C	MARR	Time (ms)
1	0.70	82
3	0.80	96
5	0.84	110
7	0.86	124
10	0.90	145

Sum of scores

C	MARR	Time (ms)
1	0.74	74
3	0.82	86
5	0.87	98
7	0.90	109
10	0.91	129

Figure 4.8: Comparison between score combination using product and sum.

negative $\mathcal{N}$ subsets. Let $s_i = \{|r_1|, ..., |r_k|\}$ be a list of scores for the subject i in the gallery, where $|r_j|$ is included in s_i if $r_j > 0$ and $i \in \mathcal{P}$ or if $r_j < 0$ and $i \in \mathcal{N}$. The combination of scores using product consists in associating

$$p = \prod_{r \in s_i} |r|, \tag{4.6}$$

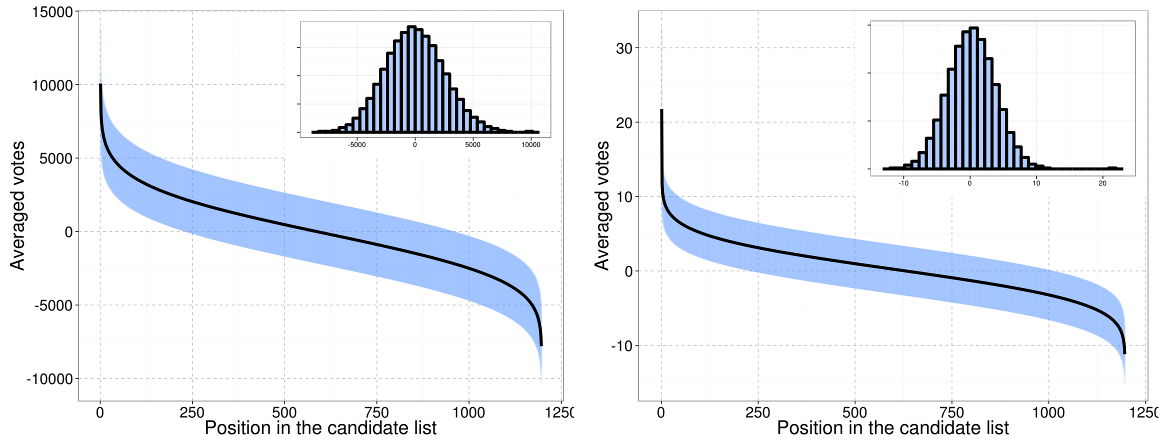
$$log(p) = \sum_{r \in s_i} log(|r| + \epsilon), \epsilon \to 0, \tag{4.7}$$

to the i-th element in the vote-list. The sum of *log* is employed in Equation 4.7 instead of the direct product of scores to avoid numeric errors (ϵ is a small positive constant). Equation 4.7 explicitly shows the difference between the sum and product of scores, which is the use of *log* in the latter.

Figure 4.8 presents the comparison between sum and product to combine scores. The average time to test samples is present in the tables of Figure 4.8 and, as expected, the product of scores is slightly slower than the sum approach due to the *log* operations. Note that there is no large difference between the two approaches in terms of MARR, therefore, we consider the sum of scores in PLSH and ePLSH.

4.2.6 Characterization of the Vote-List

Figure 4.9 presents the profile of vote values for projections drawn from the multivariate normal distribution $\mathcal{N}(0, I)$ and PLSH. As discussed in Section 2.2.2, the random projections approximate the cosine distance. The horizontal axis contains positions in



(a) (b)

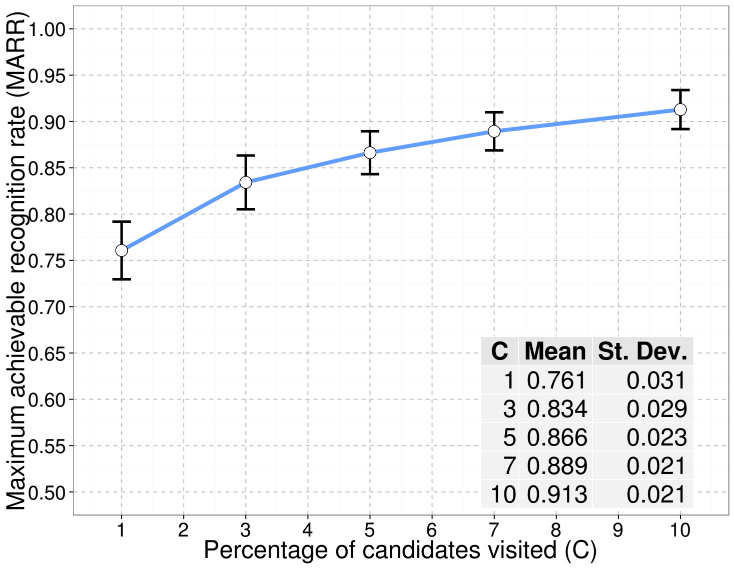
Figure 4.9: Comparison of vote values in (a) projection based on the standard multivariate normal distribution ($\mathcal{N}(0, I)$) and (b) PLSH. $\mathcal{N}(0, I)$ is employed in LSH methods to approximate l_2 distances.

the candidate list (there are 1, 196 subjects in the gallery) and the vertical axis contains the averaged vote value in the test set. Ribbons represent standard deviation and the small plots inside each figure show the histograms of the averaged vote values.

The conclusion is that the voting scheme employed in PLSH provides a bimodal distribution of vote values. One mode refers to the correct subject in the face gallery (represented by the small peak around the values 10,000 and 22 in the small plots inside Figures 4.9a and 4.9b, respectively). The other mode is relative to other subjects in the face gallery and manifests roughly as a normal distribution with zero mean. The zero mean results from the sum of r, which are learned to split samples equally using score values. In practice, what can be concluded from Figure 4.9 is that roughly half of the subjects with negative votes can be discarded in the vote-list. This heuristic is employed to reduce by half the computational cost of sorting the vote-list. Furthermore, this heuristic can be employed in any approach that consider the voting scheme described for PLSH, given that r does not present bias toward any number other than zero.

4.2.7 Stability of the Results

Figure 4.10 presents the mean MARR and standard deviation when running PLSH 10 times. Although PLSH is a nondeterministic method, it still provide fair stability, assessing that all experiments performed in this sections are easily reproducible and present statistical relevance. For instance, the best individual feature descriptor in Section 4.2.1, Gabor filter, provides MARR (at 1% of subjects in the candidate list) equal to 0.67, which is considerable lower than the averaged 0.76 MARR presented in



C	Mean	St. Dev.
1	0.761	0.031
3	0.834	0.029
5	0.866	0.023
7	0.889	0.021
10	0.913	0.021

Figure 4.10: Average MARR and standard deviation for 10 PLSH runs considering 1% of subjects in the candidate list.

Figure 4.10. The conclusion is that even with the variation in the results from the feature combination, PLSH rarely presents MARR equals to 0.67, assessing that the combination of features is better than individual features.

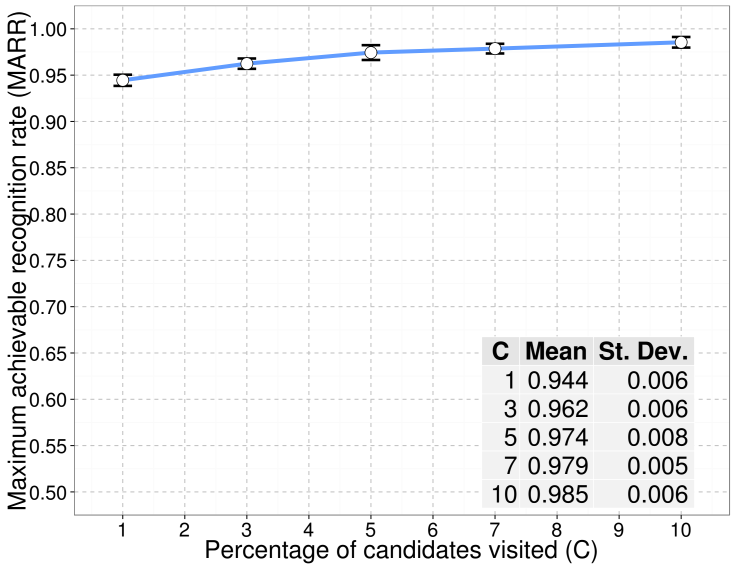
4.3 ePLSH Parameter Validation

In this section, we conduct experiments regarding stability and scalability of ePLSH in Sections 4.3.1 and 4.3.4, respectively. The feature selection methods described in Section 3.4 are evaluated in Section 4.3.2. A discussion regarding the number of features selected is presented in Section 4.3.3.

All experiments in this section consider the combination of CLBP, Gabor filters, HOG and SIFT feature descriptors. Discussion regarding the choice of other parameters and features have been presented found in PLSH Section 4.2.

4.3.1 Stability of the Results

The same experiment regarding stability of the PLSH results is performed for ePLSH in this section. The averaged MARR and standard deviation for 10 ePLSH runs are presented in Figure 4.11. We are considering regression coefficients for feature selection in this experiment and we retrain the PLS model after the feature selection step as discussed in Section 3.4. In this case, the ePLSH presents considerable more stable results than PLSH, with standard deviation around 0.006 compared to 0.03 in PLSH.



C	Mean	St. Dev.
1	0.944	0.006
3	0.962	0.006
5	0.974	0.008
7	0.979	0.005
10	0.985	0.006

Figure 4.11: Average MARR and standard deviation for 10 ePLSH runs considering 1% of subjects in the candidate list.

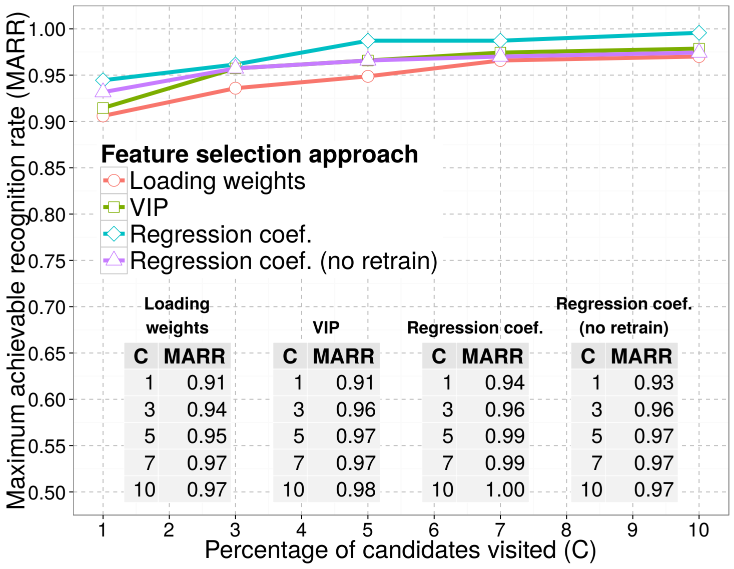
We believe that the increase in stability is a consequence of the augmented number of hash models, which reduces the variance of the sum of scores in the vote-list, resulting in a more stable distribution.

4.3.2 Feature Selection

In this section, we compare the feature selection approaches described in Section 3.4. We also compare whether we should retrain the PLS model in the regression coefficients approach to redistribute weights from the discarded feature among the selected features. According to the results presented in Figure 4.12, there is no significant difference in the feature selection approaches evaluated. Considering the experiments in Section 4.3.1, the MARR at 1% of subjects in the candidate list for the regression coefficients approach vary roughly between 0.92 and 0.96. In this case, it can be concluded that the regression coefficients approach is better than the loading weights and VIP. Furthermore, there is no significant difference between retraining or not the hash model regression coefficients after the feature selection step.

4.3.3 Number of Hash Models and Selected Features

Figure 4.13 presents MARR for 1% of subjects in the candidate list for different numbers of hash models (m) and selected features (d). The ePLSH aims at reducing the computational cost to evaluate PLSH hash functions, which can be roughly approx-



Loading weights

C	MARR
1	0.91
3	0.94
5	0.95
7	0.97
10	0.97

VIP

C	MARR
1	0.91
3	0.96
5	0.97
7	0.97
10	0.98

Regression coef.

C	MARR
1	0.94
3	0.96
5	0.99
7	0.99
10	1.00

Regression coef. (no retrain)

C	MARR
1	0.93
3	0.96
5	0.97
7	0.97
10	0.97

Figure 4.12: Evaluation of loading weights, variable importance on projection (VIP) and regression coefficients for feature selection.

imated to a number of multiplication operations equal to $m \times d$. It is important to point out that d equal to 500 provides nearly the same MARR for a sufficient large enough m. Therefore, we fix d to 500 and vary m for different datasets and number of subjects in the face gallery. In this case, we achieve minimum computational cost with almost maximum MARR using $5,000$ hash models and resulting in 2.5 million multiplications. Note that this number of multiplications refers only to the ePLSH approach such that the total computational cost of the pipeline also includes the number of multiplications in the face identification. As a comparison, the number of multiplications necessary in the brute-force approach for the $1,196$ subjects in the gallery is $1,196 \times 120,059 = 143.5$ millions, which is about 57 times more than the number of multiplications necessary to calculate all of the $5,000$ ePLSH hash functions.

The time spent to calculate each ePLSH hash function is considerable lower than PLSH hash functions. Since both approaches consists in a dot product between the feature vector and the regression vector, the number of multiplications needed to compute each hash function is equal to the dimensionality of the feature vector in PLSH ($120,059$ multiplications) and equal to the number of selected features in ePLSH (500 multiplications). In this way, ePLSH hash functions should be theoretically 240 times faster than PLSH hash functions. However, the nonlinear access to the feature vector in ePLSH hash functions may induce an additional overhead due to the weak locality of reference (accessing positions in the memory that are far from each other). We also use the same implementation of NIPALS algorithm for PLSH and ePLSH and we make

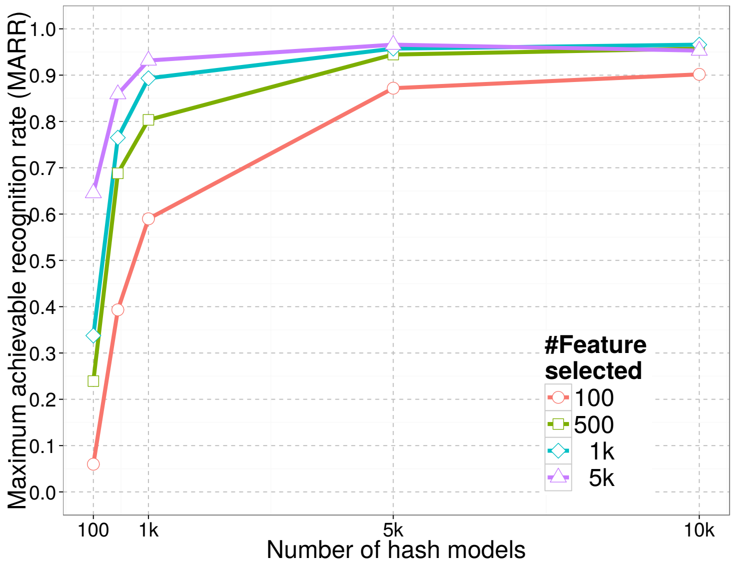


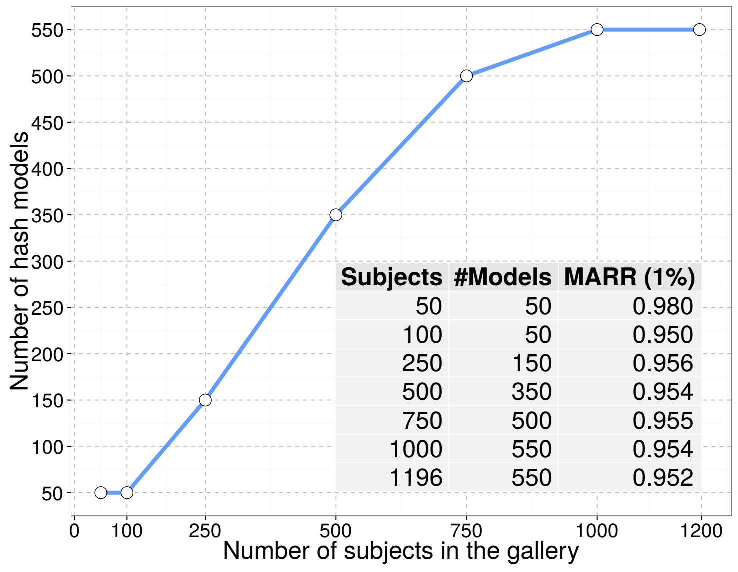
Figure 4.13: MARR with different numbers of hash models and the feature selected in ePLSH.

copies of values in the feature vector to evaluate ePLSH hash functions, which also reduce the speedup obtained in ePLSH. The best option should be directly indexing and multiplying values from the regression and the feature vectors and storing the sum without making copy of values.

The average time to calculate each PLSH hash function is 446 microseconds compared to 12 microseconds for each ePLSH hash function. However, since a considerable number of hash functions is employed in ePLSH compared to PLSH, the time to train ePLSH is significant higher than PLSH. The time spent to train all the $5,000$ hash functions in ePLSH is 14 hours compared to 22 minutes for the 150 hash functions in PLSH, which may not impose an issue because the train is performed offline and only once for a fixed face gallery. The train can also be accelerated considering other PLS algorithms such as SIMPLS [De Jong, 1993] rather than NIPALS.

4.3.4 Number of Hash Models and Gallery Size

In all experiments presented so far, we considered a fixed number of subjects in the gallery, which are in total $1,196$ for the FERET dataset. We still need to assess the ePLSH performance with an increasing number of subjects in the face gallery, which, theoretically, should require a logarithmic number of hash models to index the subjects in the face gallery (see Section 4.2.3). For the experiment in this section, we randomly select $50, 100, 250, 500, 750, 1000$ subjects in the FERET dataset to be enrolled onto the face gallery. We consider the *fb* test set in FERET because it has more test



Subjects	#Models	MARR (1%)
50	50	0.980
100	50	0.950
250	150	0.956
500	350	0.954
750	500	0.955
1000	550	0.954
1196	550	0.952

Figure 4.14: Number of hash models necessary to provide at least 0.95 MARR with different gallery sizes and 1% of subjects in the candidate list.

samples (1, 195 in total) and ePLSH because it provides more stable and better results than PLSH. We also consider only test samples of subjects enrolled in the face gallery because we are evaluating the closed set recognition. We raise the number of hash models from 50 to 550, in steps of 50, until we reach at least 0.95 MARR for 1% of subjects in the candidate list.

The results in Figure 4.14 demonstrate that at least the number of hash models necessary to maintain accuracy is logarithmic with the size of the face gallery. However, the number of subjects in the candidate list still depend on 1% of the gallery size. The problem in this case is that the face identification still needs to evaluate 1% of subjects in the face gallery, considering the worst case of the early-stop heuristic. We tried varying the percentage -or fixing to a small value- the number of subjects in the candidate list, but for both cases, the number of hash models did not stabilize for the number of enrolled subjects evaluated. We believe this happens because the number of subjects in our evaluation is not large enough to demonstrate convergence of the number of hash models. Nonetheless, Figure 4.14 indicates that we can reduce at least in two orders of magnitude the number of subjects evaluated in the face identification, which is so far, the best known result in the literature as will be presented in the next sections.

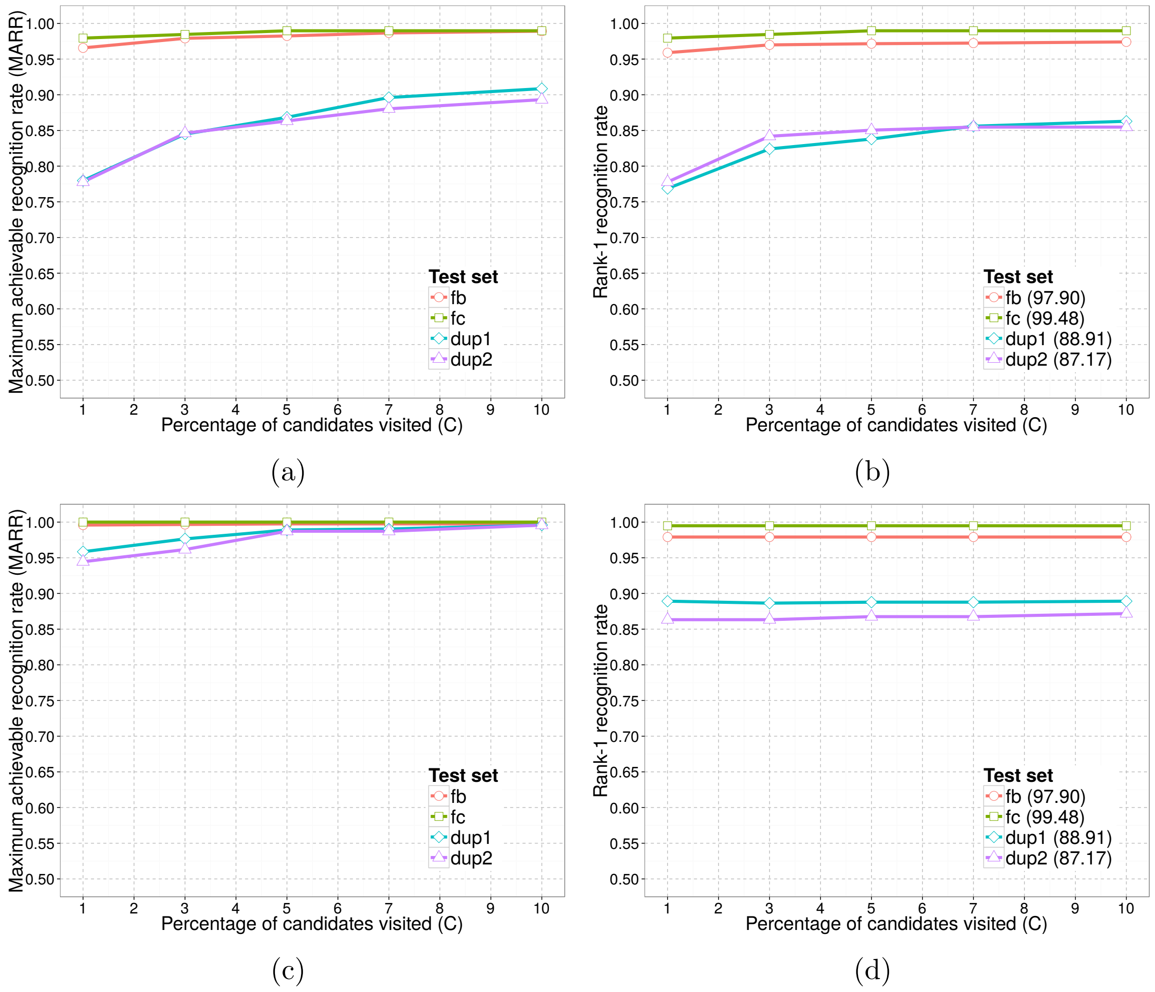


Figure 4.15: Results on the FERET dataset. (a) PLSH MARR curves, (b) PLSH rank-1 recognition rate, (c) ePLSH MARR curves and (d) ePLSH rank-1 recognition rate. Number in parenthesis indicate rank-1 recognition rate for the brute force approach.

4.4 Results on the FERET Dataset

Results regarding MARR and rank-1 recognition rate for PLSH in all test sets from the FERET dataset are presented in Figures 4.15a and 4.15b. For the test sets *fb* and *fc*, about 1% of subjects in the candidates list is enough to achieve more than 95% of the rank-1 recognition rate of the brute-force approach (presented in the legend of Figure 4.15b for each test set). However, for the test sets *dup1* and *dup2*, about 5% of subjects in the candidate list ensured at least 95% of the brute-force rank-1 recognition rate. The theoretical speedup in the worst case can be calculated considering the 150 PLSH hash function evaluations and the 5% of the gallery size, which consists of 60 PLS projections. In this case, if the early-stop search heuristic is not considered, i.e., all subjects in the candidate list are evaluated for each test sample, the number of

PLS projections would be 210 compared to the $1,196$ projections necessary in the brute-force approach, which would still results in a 5.6 times speedup.

Results from ePLSH are presented in Figures 4.15c and 4.15d. Using only 1% of subjects in the candidate list, it is possible to recover all subjects in the rank-1 recognition rate from brute-force approach for all four test sets. In this case, the rank-1 recognition rate from the ePLSH pipeline is the same as the brute-force approach, but with reduction to 1% of the subjects evaluated in the identification. Considering that the cost to evaluate all hash models in ePLSH is about the same as in PLSH, the theoretical speedup is 7.38 times compared to the brute-force approach in the worst case.

4.5 Results on the FRGC Dataset

Results from the FRGC dataset for PLSH and ePLSH are presented in Table 4.1 along with results from three other methods as presented in the literature. The three methods are the cascade of rejection classifiers (CRC) from [Yuan et al., 2005], the PLS-based search tree [Schwartz et al., 2012], and our previous published work [Santos Jr et al., 2015], which consists of PLSH with the combination of HOG, Gabor filter and LBP feature descriptors. For PLSH and ePLSH, we vary the number of hash models and the maximum percentage of subjects visited in the candidate list and we present the results with rank-1 recognition rate close to 0.95 and higher speedups. In this way, it is possible to compare directly the maximum speedup achievable when using PLSH and ePLSH compared to the other approaches, which also provide rank-1 recognition rate close to 0.95.

Results for a fixed setup that provide at least 0.95 rank-1 recognition rate are also provided, consisting of 50 hash models with 25% of subjects in the candidate list for PLSH and 200 hash models with 10% of subjects in the candidate list for ePLSH. The experiments were conducted with the following percentages of subjects in the candidate list (rounding up): $0.1, 0.5, 1, 3, 5, 7, 10, 13, 15, 20, 25, 30$. The number of hash models evaluated are: $10, 15, 20, 25, 30, 35, 40, 45, 50$; for PLSH, and $25, 50, 75, 100, 125, 150, 175, 200$, for ePLSH.

According to Table 4.1, the rank-1 recognition rate is reasonably stable, with variance in the first decimal place, which is similar to the results regarding stability presented for PLSH and ePLSH. The speedup for PLSH and ePLSH decreases considerable as the number of samples per subject available for train reduces. The reason for that is the increase in the number of hash models and the maximum number of subjects visited in the candidate list to guarantee at least 0.95 rank-1 recognition rate. Even with reduced speedups considering 35% of samples available for train, ePLSH provides significant improvement over the speedup achieved by the tree-based approach (3.6 times faster), while PLSH provide competitive speedup.

The speedup provided by PLSH and ePLSH compared to the tree-based approach is noticed with 90% of the samples available for train, where PLSH is about 5 times faster than the tree-based approach while ePLSH is about 13 times faster than PLSH. Finally, in the worse case, ePLSH provides at least 14 times speedup considering the brute-force approach in the setup with 200 hash models and 10% of subjects in the candidate list.

	% of samples for train	90%	79%	68%	57%	35%
CRC [Yuan et al., 2005]	Speedup	$1.58\times$	$1.58\times$	$1.60\times$	$2.38\times$	$3.35\times$
	Rank-1 rec. rate	80.5%	77.7%	75.7%	71.3%	58.0%
Tree-based [Schwartz et al., 2012]	Speedup	$3.68\times$	$3.64\times$	$3.73\times$	$3.72\times$	$3.80\times$
	Rank-1 rec. rate	94.3%	94.9%	94.3%	94.46%	94.46%
PLSH [Santos Jr et al., 2015] HOG,Gabor filter,LBP	Speedup	$(16.84 \pm 1.56)\times$	$(7.30 \pm 1.40)\times$	$(5.66 \pm 0.41)\times$	$(3.42 \pm 0.34)\times$	$(2.79 \pm 0.11)\times$
	Rank-1 rec. rate	$(96.5 \pm 0.7)\%$	$(96.7 \pm 1.6)\%$	$(93.4 \pm 1.3)\%$	$(93.6 \pm 0.5)\%$	$(93.3 \pm 0.7)\%$
	Hash models	10	20	25	35	35
	Max. candidates	3%	10%	13%	20%	30%
PLSH	Speedup	$(18.24 \pm 1.28)\times$	$(8.61 \pm 0.30)\times$	$(6.95 \pm 0.31)\times$	$(3.96 \pm 0.05)\times$	$(3.49 \pm 0.17)\times$
	Rank-1 rec. rate	$(95.31 \pm 0.62)\%$	$(95.31 \pm 0.70)\%$	$(93.60 \pm 1.15)\%$	$(94.67 \pm 0.34)\%$	$(94.60 \pm 0.16)\%$
	Hash models	10	20	30	50	50
	Max. candidates	3%	13%	13%	15%	25%
PLSH - 50 hash models 25% max. candidates	Speedup	$(2.95 \pm 0.03)\times$	$(4.00 \pm 0.16)\times$	$(4.13 \pm 0.30)\times$	$(3.16 \pm 0.03)\times$	$(3.49 \pm 0.17)\times$
	Rank-1 rec. rate	$(99.69 \pm 0.12)\%$	$(98.26 \pm 0.06)\%$	$(97.74 \pm 0.42)\%$	$(96.19 \pm 0.15)\%$	$(94.60 \pm 0.16)\%$
ePLSH	Speedup	$(\mathbf{233.61 \pm 37.05})\times$	$(\mathbf{98.93 \pm 8.39})\times$	$(\mathbf{45.42 \pm 3.84})\times$	$(\mathbf{22.29 \pm 1.03})\times$	$(\mathbf{14.21 \pm 1.74})\times$
	Rank-1 rec. rate	$(96.03 \pm 0.70)\%$	$(95.02 \pm 0.45)\%$	$(95.98 \pm 0.31)\%$	$(94.67 \pm 0.49)\%$	$(94.44 \pm 0.40)\%$
	Hash models	50	100	150	150	200
	Max. candidates	0.1%	0.5%	3%	5%	10%
ePLSH - 200 hash models 10% max. candidates	Speedup	$(19.74 \pm 1.35)\times$	$(16.30 \pm 1.01)\times$	$(19.12 \pm 1.89)\times$	$(12.28 \pm 0.57)\times$	$(14.21 \pm 1.74)\times$
	Rank-1 rec. rate	$(99.79 \pm 0.22)\%$	$(98.30 \pm 0.11)\%$	$(97.63 \pm 0.04)\%$	$(96.71 \pm 0.36)\%$	$(94.44 \pm 0.40)\%$

Table 4.1: Comparison between the proposed approach and other approaches in the literature. The highest speedups are shown in bold.

Chapter 5

Conclusions and Future Works

In this work, we proposed and evaluated PLSH and its extension ePLSH for face indexing. PLSH is inspired by the well-known locality-sensitive hashing for large-scale image retrieval and PLS for face identification, which provides fast and robust results for face indexing. Additional gain in speedup was achieved with the ePLSH, a method that employs PLS-based feature selection to reduce the computational cost to evaluate hash functions, enabling a large amount of additional hash functions to be employed and raising the indexing precision. We evaluated several parameters and alternative implementations of PLSH in the hope that they will be useful for future face indexing development. The experiments were conducted on two face identification standard datasets, FERET and FRGCv1, with $1,196$ and 275 subjects, respectively. Although these datasets do not provide enough number of subjects for a proper evaluation regarding scalability to large galleries, PLSH and ePLSH still provide significant improvement in speedup compared to other scalable face identification approaches in the literature.

The conclusions and considerations regarding PLSH and ePLSH are the following:

- They support for high dimensional feature vectors, allowing different complementary feature descriptors to be employed to increase the robustness of the face indexing.
- They are easy to implement and deploy in practice since the only parameters needed to be set are the number of hash models and subjects in the candidate list (considering a fixed set of parameters for the feature descriptors and face identification.).
- They do not provide good performance when the number of samples per subject is reduced.

- Incremental enrollment of subjects in the framework requires re-training of the hash models, which may be prohibitive to perform in practice, specially for ePLSH which demands considerable more hash models.

In future works, we may consider the incremental learning algorithm for PLS rather than NIPALS [Zeng and Li, 2014], which might solve the issue regarding the incremental enrollment of subjects. We also may consider learning PLSH hash models for different subsets of subjects in the gallery, which have already been extensively study by de Paulo Carlos et al. [2015] to make PLS face identification scalable to incremental enrollment of subjects in the gallery. In this way, it is possible, for instance, to distribute the processing among numerous nodes in a computer cluster, which should be necessary to scale the approach for millions of subjects. The performance drop of PLSH and ePLSH when there are few samples per subject in the face gallery might be alleviated by generating synthetic samples using face morphing methods, which has already been considered for PLS face identification to leverage the recognition rates [Schwartz et al., 2012]. Finally, the main issue regarding the experiments in this work is the low number of subjects in the datasets, which hampers the possibility to demonstrate scalability for PLSH and ePLSH. Large face identification datasets are challenging to build since they require large human effort to label and verify correctness of the identities associated to the face images. However, building large datasets for face identification might be possible by crawling face images in the Internet using the same principles employed in the LFW dataset [Huang et al., 2007] or using human work services such Amazon Mechanical Turk.

Bibliography

Ahonen, T., Hadid, A., and Pietikainen, M. (2006). Face description with local binary patterns: Application to face recognition. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 28(12):2037–2041.

Alahi, A., Ortiz, R., and Vandergheynst, P. (2012). Freak: Fast retina keypoint. In *Computer Vision and Pattern Recognition (CVPR), IEEE Conference on*, pages 510–517.

Andoni, A. and Indyk, P. (2006). Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. In *Foundations of Computer Science, IEEE Symposium on*, pages 459–468.

Barkan, O., Weill, J., Wolf, L., and Aronowitz, H. (2013). Fast high dimensional vector multiplication face recognition. In *Computer Vision (ICCV), IEEE Conference on*, pages 1960–1967.

Basri, R. and Jacobs, D. (2004). Illumination modeling for face recognition. In *Handbook of Face Recognition*, pages 89–111. Springer.

Bay, H., Ess, A., Tuytelaars, T., and Van Gool, L. (2008). Speeded-up robust features (SURF). *Computer Vision and Image Understanding, Elsevier Transactions on*, 110(3):346–359.

Belhumeur, P. N., Hespanha, J. P., and Kriegman, D. J. (1997). Eigenfaces vs. fisherfaces: Recognition using class specific linear projection. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 19(7):711–720.

Bradski, G. (2000). The OpenCV Library. *Software Tools, Dr. Dobb's Journal of. Retrieved online from* `http://opencv.org/` *on August 2015.*

Broder, A. Z. (1997). On the resemblance and containment of documents. In *Compression and Complexity of Sequences, IEEE Conference on*, pages 21–29.

Bronstein, A. M., Bronstein, M. M., Guibas, L. J., and Ovsjanikov, M. (2011). Shape Google: Geometric words and expressions for invariant shape retrieval. *Graphics (TOG), ACM Transactions on*, 30(1):1.

Brutlag, J. (2009). Speed matters for google web search. Technical report, Google. Retrieved online at `http://googleresearch.blogspot.com/2009/06/speed-matters.html` on August, 2015.

Chandrasekhar, V., Takacs, G., Chen, D., Tsai, S., Grzeszczuk, R., and Girod, B. (2009). CHOG: Compressed histogram of gradients a low bit-rate feature descriptor. In *Computer Vision and Pattern Recognition (CVPR), IEEE Conference on*, pages 2504–2511.

Chandrasekhar, V., Takacs, G., Chen, D. M., Tsai, S. S., Reznik, Y., Grzeszczuk, R., and Girod, B. (2012). Compressed histogram of gradients: A low-bitrate descriptor. *Computer Vision, Springer Journal of*, 96(3):384–399.

Chávez, E., Navarro, G., Baeza-Yates, R., and Marroquín, J. L. (2001). Searching in metric spaces. *Computing Surveys (CSUR), ACM Transactions on*, 33(3):273–321.

Chellappa, R., Sinha, P., and Phillips, P. J. (2010). Face recognition by computers and humans. *Computer, IEEE Transactions on*, 43(2):46–55.

Chen, H.-T., Chang, H.-W., and Liu, T.-L. (2005). Local discriminant embedding and its variants. In *Computer Vision and Pattern Recognition (CVPR), IEEE Conference on*, volume 2, pages 846–853.

Chum, O., Philbin, J., Zisserman, A., et al. (2008). Near duplicate image detection: min-hash and tf-idf weighting. In *British Machine Vision Conference (BMVC).*, volume 810, pages 812–815.

Dalal, N. and Triggs, B. (2005). Histograms of oriented gradients for human detection. In *Computer Vision and Pattern Recognition (CVPR), IEEE Conference on*, volume 1, pages 886--893.

Dasgupta, S. and Freund, Y. (2008). Random projection trees and low dimensional manifolds. In *Theory of Computing, ACM Symposium on*, pages 537–546.

Datar, M., Immorlica, N., Indyk, P., and Mirrokni, V. S. (2004). Locality-sensitive hashing scheme based on p-stable distributions. In *Computational Geometry. ACM Symposium on*, pages 253–262.

De Jong, S. (1993). Simpls: an alternative approach to partial least squares regression. *Chemometrics and Intelligent Laboratory Systems, Elsevier Transactions on*, 18(3):251–263.

de Paulo Carlos, G., Pedrini, H., and Schwartz, W. R. (2015). Classification schemes based on partial least squares for face identification. *Journal of Visual Communication and Image Representation*, 32:170 – 179. ISSN 1047-3203.

Deng, W., Hu, J., and Guo, J. (2012). Extended SRC: Undersampled face recognition via intraclass variant dictionary. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 34(9):1864–1870.

Deng, W., Hu, J., and Guo, J. (2013). In defense of sparsity based face recognition. In *Computer Vision and Pattern Recognition (CVPR), IEEE Conference on*, pages 399–406.

Erin Liong, V., Lu, J., Wang, G., Moulin, P., and Zhou, J. (2015). Deep hashing for compact binary codes learning. In *Computer Vision and Pattern Recognition (CVPR), IEEE Conference on*, pages 2475–2483.

Fisher, R. A. (1936). The use of multiple measurements in taxonomic problems. *Eugenics, Wiley Online Library Annals of*, 7(2):179–188.

Gan, G., Ma, C., and Wu, J. (2007). *Appendix B: The kd-Tree Data Structure*, chapter 22, pages 375–376. Siam.

Gong, Y., Lazebnik, S., Gordo, A., and Perronnin, F. (2013). Iterative quantization: A procrustean approach to learning binary codes for large-scale image retrieval. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 35(12):2916–2929.

Gu, W., Xiang, C., Venkatesh, Y., Huang, D., and Lin, H. (2012). Facial expression recognition using radial encoding of local gabor features and classifier synthesis. *Pattern Recognition, Elsevier Transactions on*, 45(1):80–91.

Guo, H., Schwartz, W. R., and Davis, L. S. (2011). Face verification using large feature sets and one shot similarity. In *Biometrics (IJCB), IEEE International Joint Conference on*, pages 1–8.

Har-Peled, S. (2001). A replacement for voronoi diagrams of near linear size. In *Foundations of Computer Science. IEEE Symposium on*, pages 94–.

He, B., Xu, D., Nian, R., van Heeswijk, M., Yu, Q., Miche, Y., and Lendasse, A. (2014). Fast face recognition via sparse coding and extreme learning machine. *Cognitive Computation, Springer Transactions on*, 6(2):264–277.

He, K., Wen, F., and Sun, J. (2013). K-means hashing: An affinity-preserving quantization method for learning binary compact codes. In *Computer Vision and Pattern Recognition (CVPR), IEEE Conference on*, pages 2938–2945.

Hua, G., Brown, M., and Winder, S. (2007). Discriminant embedding for local image descriptors. In *Computer Vision (ICCV), IEEE Conference on*, pages 1–8.

Huang, G. B., Ramesh, M., Berg, T., and Learned-Miller, E. (2007). Labeled faces in the wild: A database for studying face recognition in unconstrained environments. Technical report 07-49, University of Massachusetts, Amherst.

Indyk, P. and Motwani, R. (1998). Approximate nearest neighbors: towards removing the curse of dimensionality. In *Theory of Computing, ACM Symposium on*, pages 604–613.

Jain, P., Kulis, B., and Grauman, K. (2008). Fast image search for learned metrics. In *Computer Vision and Pattern Recognition (CVPR), IEEE Conference on*, pages 1–8.

Jégou, H., Perronnin, F., Douze, M., Sanchez, J., Perez, P., and Schmid, C. (2012). Aggregating local image descriptors into compact codes. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 34(9):1704–1716.

Joly, A. and Buisson, O. (2011). Random maximum margin hashing. In *Computer Vision and Pattern Recognition (CVPR), IEEE Conference on*, pages 873–880.

Joly, A., Frélicot, C., and Buisson, O. (2004). Feature statistical retrieval applied to content based copy identification. In *Image Processing (ICIP). IEEE Conference on*, volume 1, pages 681–684.

Kämäräinen, J.-K., Hadid, A., and Pietikäinen, M. (2011). Local representation of facial features. In *Handbook of Face Recognition*, pages 79–108. Springer.

Klare, B. F. and Jain, A. K. (2013). Heterogeneous face recognition using kernel prototype similarities. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 35(6):1410–1422.

Kulis, B., Jain, P., and Grauman, K. (2009). Fast similarity search for learned metrics. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 31(12):2143–2157.

Leutenegger, S., Chli, M., and Siegwart, R. Y. (2011). Brisk: Binary robust invariant scalable keypoints. In *Computer Vision (ICCV), IEEE Conference on*, pages 2548–2555.

Li, S. Z. and Jain, A. K. (2011). *Handbook of Face Recognition.* Springer Publishing Company, Incorporated, 2nd edition. ISBN 085729931X, 9780857299314.

Liao, S., Jain, A. K., and Li, S. Z. (2013). Partial face recognition: Alignment-free approach. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 35(5):1193–1205.

Lin, K., Yang, H.-F., Hsiao, J.-H., and Chen, C.-S. (2015). Deep learning of binary hash codes for fast image retrieval. In *Computer Vision and Pattern Recognition Workshops (CVPRW), IEEE Conference on*, pages 27–35.

Liu, T., Moore, A. W., Yang, K., and Gray, A. G. (2004). An investigation of practical approximate nearest neighbor algorithms. In *Advances in Neural Information Processing Systems, Conference on*, pages 825–832. The MIT Press.

Liu, W., Wang, J., Ji, R., Jiang, Y.-G., and Chang, S.-F. (2012). Supervised hashing with kernels. In *Computer Vision and Pattern Recognition (CVPR), IEEE Conference on*, pages 2074–2081.

Lowe, D. G. (2004). Distinctive image features from scale-invariant keypoints. *Computer Vision, Springer Journal of*, 60(2):91–110.

Mehmood, T., Liland, K. H., Snipen, L., and Sæbø, S. (2012). A review of variable selection methods in partial least squares regression. *Chemometrics and Intelligent Laboratory Systems, Elsevier Transactions on*, 118:62–69.

Mevik, B.-H. and Wehrens, R. (2007). The PLS package: principal component and partial least squares regression in R. *Statistical Software, American Statistical Association Journal of*, 18(2):1–24.

Mopuri, K. and Babu, R. (2015). Object level deep feature pooling for compact image representation. In *Computer Vision and Pattern Recognition Workshops (CVPRW), IEEE Conference on*, pages 62–70.

Motwani, R., Naor, A., and Panigrahy, R. (2007). Lower bounds on locality sensitive hashing. *Discrete Mathematics, SIAM Journal on*, 21(4):930–935.

Muja, M. and Lowe, D. G. (2014). Scalable nearest neighbor algorithms for high dimensional data. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 36.

Nister, D. and Stewenius, H. (2006). Scalable recognition with a vocabulary tree. In *Computer Vision and Pattern Recognition (CVPR), IEEE Conference on*, volume 2, pages 2161–2168.

Norouzi, M. and Blei, D. M. (2011). Minimal loss hashing for compact binary codes. In *Machine Learning (ICML), International Conference on*, pages 353–360.

Norouzi, M., Punjani, A., and Fleet, D. J. (2012). Fast search in hamming space with multi-index hashing. In *Computer Vision and Pattern Recognition (CVPR), IEEE Conference on*, pages 3108–3115.

Oh, J., Choi, S.-I., Kim, C., Cho, J., and Choi, C.-H. (2013). Selective generation of gabor features for fast face recognition on mobile devices. *Pattern Recognition, Elsevier Transactions on*, 34(13):1540–1547.

Pang, S., Kim, D., and Bang, S. Y. (2005). Face membership authentication using SVM classification tree generated by membership-based LLE data partition. *Neural Networks, IEEE Transactions on*, 16(2):436–446.

Phillips, P. J., Flynn, P. J., Scruggs, T., Bowyer, K. W., Chang, J., Hoffman, K., Marques, J., Min, J., and Worek, W. (2005). Overview of the face recognition grand challenge. In *Computer Vision and Pattern Recognition (CVPR), IEEE Conference on*, volume 1, pages 947–954.

Phillips, P. J., Moon, H., Rizvi, S. A., and Rauss, P. J. (2000). The FERET evaluation methodology for face-recognition algorithms. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 22(10):1090–1104.

Poullot, S., Buisson, O., and Crucianu, M. (2007). Z-grid-based probabilistic retrieval for scaling up content-based copy detection. In *Image and Video Retrieval. ACM Conference on*, pages 348–355.

Randen, T. and Husoy, J. H. (1999). Filtering for texture classification: A comparative study. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 21(4):291–310.

Rosipal, R. and Kramer, N. (2006). Overview and recent advances in partial least squares. In *Subspace, Latent Structure and Feature Selection: Statistical and Optimization Perspectives Workshop, SLSFS 2005 Bohinj, Slovenia, Revised Selected Papers*, volume 3940, page 34. Springer.

Rublee, E., Rabaud, V., Konolige, K., and Bradski, G. (2011). ORB: an efficient alternative to SIFT or SURF. In *Computer Vision (ICCV), IEEE Conference on*, pages 2564–2571.

Salah, S. H., Du, H., and Al-Jawad, N. (2013). Fusing local binary patterns with wavelet features for ethnicity identification. *Signal Image Process, IEEE Transactions on*, 21(5):416–422.

Salakhutdinov, R., Mnih, A., and Hinton, G. (2007). Restricted Boltzmann machines for collaborative filtering. In *Machine learning. International Conference on*, pages 791–798.

Santos Jr, C. E., Kijak, E., Gravier, G., and Schwartz, W. R. (2015). Learning to hash faces using large feature vectors. In *Content-Based Multimedia Indexing (CBMI), 13th IEEE International Workshop on*, pages 1–6.

Santos Jr, C. E. and Schwartz, W. R. (2014). Extending face identification to open-set face recognition. In *Graphics, Patterns and Images (SIBGRAPI), IEEE Conference on*, pages 188–195.

Schwartz, W. R., Guo, H., Choi, J., and Davis, L. S. (2012). Face identification using large feature sets. *Image Processing, IEEE Transactions on*, 21(4):2245–2255.

Shakhnarovich, G. (2005). *Learning task-specific similarity*. PhD thesis, Massachusetts Institute of Technology.

Shakhnarovich, G. and Moghaddam, B. (2011). Face recognition in subspaces. In *Handbook of Face Recognition*, pages 19–49. Springer.

Silpa-Anan, C. and Hartley, R. (2008). Optimised KD-trees for fast image descriptor matching. In *Computer Vision and Pattern Recognition (CVPR), IEEE Conference on*, pages 1–8.

Sirovich, L. and Kirby, M. (1987). Low-dimensional procedure for the characterization of human faces. *Optical Society of America A (JOSA), Optical Society of America (OSA) Transactions on*, 4(3):519–524.

Sproull, R. F. (1991). Refinements to nearest-neighbor searching in k-dimensional trees. *Algorithmica, Springer Transactions on*, 6(1-6):579–589.

Srinivasan, B. V., Schwartz, W. R., Duraiswami, R., and Davis, L. (2010). Partial least squares on graphical processor for efficient pattern recognition. Technical report, University of Maryland. Computer Science Department. CS-TR-4968.

Strecha, C., Bronstein, A. M., Bronstein, M. M., and Fua, P. (2012). LDAHash: Improved matching with smaller descriptors. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 34(1):66–78.

Terasawa, K. and Tanaka, Y. (2007). Spherical LSH for approximate nearest neighbor search on unit hypersphere. In *Algorithms and Data Structures, Springer Conference on*, pages 27–38. Springer.

Torralba, A., Fergus, R., and Weiss, Y. (2008). Small codes and large image databases for recognition. In *Computer Vision and Pattern Recognition (CVPR), IEEE Conference on*, pages 1–8.

Uhlmann, J. K. (1991). Satisfying general proximity/similarity queries with metric trees. *Information Processing Letters, Elsevier Transactions on*, 40(4):175–179.

Valstar, M. F. and Pantic, M. (2012). Fully automatic recognition of the temporal phases of facial actions. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 42(1):28–43.

Vu, N.-S., Dee, H. M., and Caplier, A. (2012). Face recognition using the POEM descriptor. *Pattern Recognition, Elsevier Transactions on*, 45(7):2478–2488.

Wang, H., Li, S. Z., and Wang, Y. (2004). Face recognition under varying lighting conditions using self quotient image. In *Automatic Face and Gesture Recognition, IEEE Conference on*, pages 819–824.

Wang, J., Kumar, S., and Chang, S.-F. (2012). Semi-supervised hashing for large-scale search. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 34(12):2393–2406.

Wang, J., Shen, H. T., Song, J., and Ji, J. (2014a). Hashing for similarity search: A survey. *arXiv preprint arXiv:1408.2927*.

Wang, J., Wang, N., Jia, Y., Li, J., Zeng, G., Zha, H., and Hua, X.-S. (2014b). Trinary-projection trees for approximate nearest neighbor search. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 36(2):388–403.

Wechsler, H. (2009). *Reliable face recognition methods: system design, implementation and evaluation*, volume 7. Springer-Verlag New York, Inc., Secaucus, NJ, USA. ISBN 038722372X.

Weiss, Y., Torralba, A., and Fergus, R. (2009). Spectral hashing. In Koller, D., Schuurmans, D., Bengio, Y., and Bottou, L., editors, *Advances in Neural Information Processing Systems 21, Curran Associates Conference on*, pages 1753–1760.

Wold, H. (1985). Partial least squares. In Kotz, S. and Johnson, N., editors, *Encyclopedia of Statistical Science*, pages 581–591. New York: Wiley.

Wright, J., Yang, A. Y., Ganesh, A., Sastry, S. S., and Ma, Y. (2009). Robust face recognition via sparse representation. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 31(2):210--227.

Xie, Z., Liu, G., and Fang, Z. (2012). Face recognition based on combination of human perception and local binary pattern. In *Intelligent Science and Intelligent Data Engineering, Springer Revised Selected Papers on*, pages 365–373. Springer.

Yuan, Q., Thangali, A., and Sclaroff, S. (2005). Face identification by a cascade of rejection classifiers. In *Computer Vision and Pattern Recognition Workshops (CVPRW), IEEE Conference on*, pages 152–152.

Zeng, X.-Q. and Li, G.-Z. (2014). Incremental partial least squares analysis of big streaming data. *Pattern Recognition, Elsevier Transactions on*, 47(11):3726–3735.

Appendix A

Partial Least Squares Example in R

The goal in this chapter is to provide practical information regarding PLS implementation. A sample to calculate PLS beta regression vector is presented in Figure A.1. The objective is to learn a PLS model (beta regression vector) that returns values close to +1 for the flower parameters (petal length, sepal length and width), if the flower is Virginica, and −1, otherwise. To prepare the data to learn the PLS model, we first create a matrix $X_{n \times 3}$ with the n samples in the dataset (lines 1–5). Then, a target matrix Y_{nx1} is created where each element y_i is set to +1, if the correspondent element in X is from Virginica flower or −1 otherwise (lines 6 and 7). The columns in the matrices X and Y are normalized to zero mean, according to NIPALS algorithm, and unit standard deviation to avoid that large-scale features receive more importance in the projection even if it is not very discriminant (lines 11–13).

After setting the number of factors and reserving memory for PLS matrices (lines 16–24, see Equation 3.1 for definition of each matrix), we initialize randomly (alternatively a column in X or $X^TY/||X^TY||$) a regression direction u_i (line 29) and iterate until the maximum covariance axis is found (lines 32–43). In practice, few iterations are necessary to find the maximum covariance axis. In our tests, we noticed that the convergence is immediate, at the first iteration, and the value of $W[,i]$ does not change in the next iterations. Lines 45–50 remove the $W[,i]$ contribution from X and Y so the next iteration find a new regression direction orthogonal to previously calculated directions. The β regression vector between the latent variables represented in the matrices T and U can be calculated using Equation 3.2 (line 53). A copy of the original target values Y is necessary to calculate the regression vector (line 10). In practice, pseudo-inverse is employed to avoid errors regarding singular matrices. To test, it is necessary to normalize the test sample (subtract the mean and divide by standard deviation pre-calculated in lines 11–13), and calculate the dot product between the normalized test sample and the beta regression vector.

```r
## Prepare dataset
virginica <- iris[iris$Species == 'virginica', 1:3]
others <- iris[iris$Species != 'virginica', 1:3]

X <- as.matrix(rbind(virginica, others))
Y <- rbind(matrix(data=+1, nrow=nrow(virginica), ncol=1),
                  matrix(data=-1, nrow=nrow(others), ncol=1))

## Normalize data
Ycopy <- Y
X <- X - apply(X, 2, mean)
X <- X / apply(X, 2, sd)
Y <- Y - apply(Y, 2, mean)

## Run NIPALS
factors <- 2

## reserve space for matrices
T <- matrix(nrow=nrow(X), ncol=factors)
P <- matrix(nrow=ncol(X), ncol=factors)
U <- matrix(nrow=nrow(Y), ncol=factors)
Q <- matrix(nrow=ncol(Y), ncol=factors)
W <- matrix(nrow=ncol(X), ncol=factors)
B <- matrix(nrow=1, ncol=factors)

for(i in 1:factors) {

        ## start ui randomly or with some column of X
        U[,i] <- rnorm(n=nrow(U), mean=0, sd=1)

        ## find maximum covariance axis
        for(epoc in 1:10) {

                W[,i] <- t(X) %*% U[,i]
                W[,i] <- W[,i] / sqrt(t(W[,i]) %*% W[,i])[1, 1]

                T[,i] <- X %*% W[,i]

                Q[,i] <- t(Y) %*% T[,i]
                Q[,i] <- Q[,i] / sqrt(t(Q[,i]) %*% Q[,i])[1, 1]

                U[,i] <- Y %*% Q[,i]
        }
        ## deflate matrices
        B[,i] <- t(U[,i]) %*% T[,i] / (t(T[,i]) %*% T[,i])[1, 1]

        P[,i] <- t(X) %*% T[,i] / (t(T[,i]) %*% T[,i])[1, 1]

        X <- X - T[,i] %*% t(P[,i])
        Y <- Y - B[,i] * (T[,i] %*% t(Q[,i]))
}
## Calculate regression coefficients
beta <- (W %*% solve(t(P) %*% W)) %*% solve(t(T) %*% T) %*% t(T) %*% Ycopy
```

Figure A.1: PLS sample code in R language used to discriminate Virginica from others flowers in the IRIS dataset. This code was used to generate Figure 3.2.