Cergy Paris University - École doctorale n° 405 Économie, management, mathématiques, physique et sciences informatiques (EM2PSI)

High-Order Statistics for Image Representations using Metric Learning

by

Pierre Jacob

This dissertation is submitted for the degree of
Doctor of Philosophy from Cergy Paris University

Aymeric Histace	Full Professor, ENSEA	Supervisor
David Picard	Senior Research Scientist, Ecole des Ponts ParisTech	Advisor
Edouard Klein	CNE, Pôle Judiciaire de la Gendarmerie Nationale	Advisor
Matthieu Cord	Full Professor, Sorbonne University, Valeo AI	Rapporteur
Ondrej Chum	Associate Professor, CTU Prague	Rapporteur
Diane Larlus	Senior Research Scientist, Naver Labs Europe	Examiner
Nicole Vincent	Full Professor, Université de Paris	Examiner
Hervé Jégou	Senior Research Scientist, Facebook AI	Examiner

Submitted: June, 30th, 2020

English Abstract

High-Order Statistics for Image Representations using Metric Learning

Abstract:

An important challenge in artificial intelligence is the learning of useful data representations, or features, on which recognition algorithms can be used in order to solve a specific task (object or scene recognition, video annotation, *etc.*). By learning rich representations, that is, which encode the relevant information with respect to the given task, performances of the majority of recognition algorithms can be highly improved. The major difficulty in learning such representations rests on the high dimensionality of the input data, and the lack of prior knowledge about the relevant patterns that should be encoded.

In this thesis, we particularly focus on image representation for the task of content-based image retrieval using the deep metric learning framework. These representations must be low-dimensional in order to reduce the memory storage, the computation cost, and to speed-up the search. Also, they should encode useful information in order to be able to retrieve the most relevant images from the database.

We make the following contributions to alleviate the aforementioned issues: First, we present three image representations that leverage dictionary learning in order to produce richer representation along with covariance matrices or attention mechanisms. Second, we propose two improvements during the training: a method that generates hard examples to resume the training, and a regularization that improves the robustness of the local features for a variety of representations. We empirically show that the proposed strategies significantly improve baseline methods and provide stronger results than most of the state-of-the-art methods.

Keywords:

metric learning; image retrieval; bilinear pooling; second-order pooling; high-order pooling; regularization; triplet generation; attention mechanism

French Abstract

High-Order Statistics for Image Representations using Metric Learning

Résumé:

Un défi majeur de l'intelligence artificielle est l'apprentissage de représentations, ou descripteurs, sur lesquels des algorithmes de reconnaissance peuvent être utilisés afin de résoudre une tâche spécifique (reconnaissance d'objets ou de scènes, annotation de vidéos, *etc.*). En apprenant des représentations riches, c'est-à-dire qui encodent les informations pertinentes par rapport à la tâche donnée, les performances de la majorité des algorithmes de reconnaissance peuvent être fortement améliorées. La difficulté majeure de l'apprentissage de telles représentations repose sur la grande dimensionnalité des données d'entrée et sur le manque de connaissance a priori des caractéristiques pertinentes à encoder.

Dans cette thèse, nous abordons l'apprentissage de représentations d'images pour la tâche de recherche d'images basée sur le contenu visuel en apprentissage de distance. Ces représentations doivent être de faible dimension afin de réduire l'utilisation mémoire, le coût en calculs et afin d'accélérer la recherche. Elles doivent également encoder les informations pertinentes pour la recherche d'images.

Nous apportons les contributions suivantes : Premièrement, nous présentons trois représentations d'images qui tirent parti de l'apprentissage de dictionnaires ainsi que des matrices de covariance ou des mécanismes d'attention afin de produire des représentations plus riches. Deuxièmement, nous proposons deux améliorations de l'entraînement : une méthode qui génère des exemples difficiles pour reprendre l'entraînement lorsque l'échantillonnage aléatoire des examples pertinents devient trop complexe, et une méthode de régularisation qui améliore la robustesse des descripteurs locaux pour différentes représentations. Nous montrons empiriquement que les stratégies proposées améliorent considérablement les méthodes de base et donnent des résultats plus élevés que la plupart des méthodes actuelles.

Mot-clefs:

Apprentissage de métrique; recherche d'images; agrégation bilinéaire; agrégation du second-ordre; agrégation d'ordres élevés; génération de triplets; mécanismes d'attention

Publications

Journals:

- **Pierre Jacob**, David Picard, Aymeric Histace, Edouard Klein. *"DIABLO: Dictionary-based attention block for deep metric learning"*. Pattern Recognition Letters, volume 135, pages 99 – 105, July 2020.

International conferences:

- **Pierre Jacob**, David Picard, Aymeric Histace, Edouard Klein. *"Improving Deep Metric Learning with Virtual Classes and Examples Mining"*. In submission, Nov. 2019.
- **Pierre Jacob**, David Picard, Aymeric Histace, Edouard Klein. *"Metric Learning With HORDE: High-Order Regularizer for Deep Embeddings"*. Proceedings of the IEEE International Conference on Computer Vision (ICCV), Oct. 2019.
- **Pierre Jacob**, David Picard, Aymeric Histace, Edouard Klein. *"Efficient Codebook and Factorization for Second Order Representation Learning"*. Proceedings of the IEEE International Conference on Image Processing (ICIP), Sept. 2019.
- **Pierre Jacob**, David Picard, Aymeric Histace, Edouard Klein. *"Leveraging Implicit Spatial Information in Global Features for Image Retrieval"*. Proceedings of the IEEE International Conference on Image Processing (ICIP), Oct. 2018.

Not included in the Thesis:

- Romain Leenhardt, Pauline Vasseur, Cynthia Li, Jean Christophe Saurin, Gabriel Rahmi, Franck Cholet, Aymeric Becq, Philippe Marteau, Aymeric Histace, Xavier Dray, Sylvie Sacher-Huvelin, Farida Mesli, Chloé Leandri, Isabelle Nion-Larmurier, Stéphane Lecleire, Romain Gerard, Clotilde Duburque, Geoffroy Vanbiervliet, Xavier Amiot, Jean Philippe Le Mouel, Michel Delvaux, **Pierre Jacob**, Camille Simon-Shane, Olivier Romain. *"A neural network algorithm for detection of GI angiectasia during small-bowel capsule endoscopy"*. Proceedings of Gastrointestinal endoscopy, Jan. 2019.
- Gaetan Raynaud, Camille Simon Chane, **Pierre Jacob**, Aymeric Histace. *"Active Contour Segmentation based on Histograms and Dictionary Learning for Videocapsule Image Analysis"*. Proceedings of 14th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications, Feb. 2019.
- Marc Souchaud, **Pierre Jacob**, Camille Simon-Chane, Aymeric Histace, Olivier Romain, Maurice Tchuenté, Denis Sereno. *"Mobile Phones Hematophagous Diptera Surveillance in the field using Deep Learning and Wing Interference Patterns"*. Proceedings of the IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC), Oct. 2018.

Contents

English Abstract i

French Abstract iii

Publications v

List of Figures xi

List of Tables xiii

List of Symbols xv

Preface xvii

I Background and Motivations 1

1 Image Representation Learning 3
- 1.1 Introduction ... 4
- 1.2 Global representations ... 4
- 1.3 Local features ... 5
- 1.4 Aggregation methods ... 6
- 1.5 Discussion ... 10

2 Metric Learning 13
- 2.1 Introduction ... 14
- 2.2 Formulation ... 14
- 2.3 Loss Functions ... 17
  - 2.3.1 Contrastive and triplet losses improvements ... 17
  - 2.3.2 New designs ... 19
- 2.4 Sampling Strategies ... 20
  - 2.4.1 Approximate example search ... 20
  - 2.4.2 Hard example generation ... 21
- 2.5 Ensemble Methods ... 22
- 2.6 Datasets and Evaluation Protocol ... 23
  - 2.6.1 Datasets ... 23
  - 2.6.2 Metrics ... 24
- 2.7 Discussion ... 26

3 Second-Order Pooling 29
- 3.1 Introduction ... 30
- 3.2 Formulation ... 30
- 3.3 Normalization ... 33
- 3.4 Factorization ... 38
- 3.5 Beyond Second-Order Pooling ... 43
- 3.6 Discussion ... 45

II Contributions 47

4 Dictionary-based tensor aggregation 49
- 4.1 Prologue ... 50
- 4.2 Improved Spatial Tensor Aggregation ... 51
  - 4.2.1 Introduction ... 51
  - 4.2.2 Spatial tensor aggregation ... 52

4.2.3 Local feature pre-processing ... 52
4.2.4 Normalization ... 54
4.2.5 Factorization ... 54
4.2.6 Experiments ... 55
4.2.7 Limitations ... 56
4.3 Joint Codebook Factorization ... 57
4.3.1 Introduction ... 57
4.3.2 VLAT representation ... 57
4.3.3 Differential dictionary learning ... 58
4.3.4 Modified representation ... 58
4.3.5 Factorization ... 59
4.3.6 Low-rank approximation ... 59
4.3.7 Ablation studies ... 60
4.3.8 Comparison to the state-of-the-art ... 61
4.3.9 Discussion ... 61
4.4 Dictionary learning as attention map ... 62
4.4.1 Introduction ... 62
4.4.2 Method overview ... 62
4.4.3 Attention strategies ... 63
4.4.4 Feature-wise selection ... 64
4.4.5 Dimension-wise selection ... 65
4.4.6 Implementation details ... 66
4.4.7 Ablation studies ... 66
4.4.8 Comparison to the state-of-the-art ... 69
4.5 Global Framework ... 70
4.6 Conclusion ... 71

5 HORDE: High-Order Regularizer for Deep Embedding 73
5.1 Prologue ... 74
5.2 Introduction ... 75
5.3 Proposed High-Order Regularizer ... 75
5.3.1 High-order computation ... 76
5.3.2 Theoretical analysis ... 78
5.4 Experiments ... 80
5.4.1 Comparison to state-of-the-art ... 80
5.4.2 Ablation studies ... 82
5.5 Discussion ... 84
5.5.1 Magnet loss and DVML ... 84
5.5.2 Empirical estimators and the image distribution hypothesis ... 84
5.5.3 Image representation ... 85
5.5.4 Random Maclaurin factorization ... 85
5.6 Conclusion ... 86

6 MIRAGE: Improving DML with Virtual Classes and Examples Mining 87
6.1 Prologue ... 88
6.2 Introduction ... 88
6.3 Method overview ... 89
6.3.1 *MIRAGE* overview ... 89
6.3.2 Deep metric learning ... 90
6.3.3 Training class example generation ... 90
6.3.4 Virtual class example generation ... 91
6.3.5 *MIRAGE* architecture ... 91
6.4 Experiments ... 91
6.4.1 Prototype visualization ... 91
6.4.2 Example generation ablation ... 92
6.4.3 Comparison to state-of-the-art ... 93
6.5 Discussion ... 96

6.6 Conclusion 97

III Conclusion and Perspectives 99

7 Global Conclusion 101

8 Perspectives 103
8.1 Image Representation 104
8.1.1 Extension of Joint Codebook Factorization 104
8.1.2 Factorization of Covariance Matrices 104
8.1.3 High-Order Moments for Image Representation 105
8.1.4 Invariance and Prior in Image Representation 105
8.2 Extension of *HORDE* 105
8.3 Rethinking Metric Learning 106

Bibliography 109

List of Figures

1.1 Example of images for the fine-grained classification task. 10

3.1 Matrix backpropagation orevall scheme. 35

4.1 Illustration of the implicit neighborhood matching. 53
4.2 Sample of queries improved by performing implicit spatial consistency check. 56
4.3 Illustration of post and pre-attention strategies. 63
4.4 Illustration of the merging block M for feature-wise and dimension-wise selection strategies. 64

5.1 2D visualizations of representations and deep features 76
5.2 Global overview of our *HORDE* architecture. 76
5.3 Qualitative results on CUB for *HORDE*. 83

6.1 Hard negative generation. 89
6.2 Overview of *MIRAGE* architecture. 92
6.3 *MIRAGE*: Visualization of the training and virtual prototypes on MNIST and Cub-200-2011 datasets. 93

List of Tables

4.1 Comparison with the state-of-the-art in mAP. 55
4.2 Comparison of codebook strategy in terms of parameters and performances 61
4.3 Comparison with the state-of-the-art on Stanford Online Products dataset. 61
4.4 Comparison with the state-of-the-art on CUB-200-2011 and Cars-196 datasets. 62
4.5 Comparison with the state-of-the-art on Cub-200-2011 and Cars-196 datasets 67
4.6 Comparison with the state-of-the-art on Stanford Online Products and In-Shop Clothes Retrieval 67
4.7 Impact of the pre-attention or post-attention. 68
4.8 Impact of the dictionary size 69

5.1 Comparison with the state-of-the-art on Cub-200-2011 and Cars-196 datasets. 81
5.2 Comparison with the state-of-the-art on Stanford Online Products and In-Shop Clothes Retrieval. 81
5.3 Impact of the high order moments as regularizers. 82
5.4 Impact of the high order moments when all parameters are trained. 82
5.5 Impact of the cascaded architecture when all parameters are trained using Algorithm 1. 83

6.1 *MIRAGE*: Impact of the number of training class examples and virtual classes 92
6.2 *MIRAGE*: Loss function ablation on the Cars-196 dataset. 94
6.3 *MIRAGE*: Comparison to the state-of-the-art on the Cub-200-2011 dataset. 94
6.4 *MIRAGE*: Comparison to the state-of-the-art on the Cars-196 dataset. 95
6.5 *MIRAGE*: Comparison to the state-of-the-art on the Stanford Online Products dataset. 95
6.6 *MIRAGE*: Comparison to the state-of-the-art on the In-Shop Clothes Retrieval dataset. 95

List of Symbols

The next list describes several symbols that will be later used within the body of the document.

Numbers and arrays

a	Scalar (integer or real)
$\boldsymbol{a}$	Vector
$\boldsymbol{A}$	Matrix
$\boldsymbol{I}_n$	Identity matrix with n rows and n columns
$\boldsymbol{I}$	Identity matrix with dimensionality implied by context
$\mathbf{1}_n$	Vector of size n or matrix of size $n \times n$ depending on the context, full of 1
$\mathbf{1}$	Vector or matrix with dimensionality implied by context, full of 1
$\mathbf{0}_n$	Vector of size n or matrix of size $n \times n$ depending on the context, full of 0
$\mathbf{0}$	Vector or matrix with dimensionality implied by context, full of 0
$\boldsymbol{\mathcal{A}}$	Tensor

Functions

$f : \mathcal{X} \to \mathcal{Y}$	Function f with domain $\mathcal{X}$ and range $\mathcal{Y}$
$f(\boldsymbol{x} \,;\, \boldsymbol{\theta})$	Function f over $\boldsymbol{x}$ parametrized by $\boldsymbol{\theta}$ parameters (might be omitted to lighten notations)
$\log$	natural logarithm
$[\cdot]_+$	simplified notation for the ReLU function
$\sqrt{\boldsymbol{A}}$	matrix square root if $\boldsymbol{A}$ is a positive (semi-)definite matrix
$\log \boldsymbol{A}$	matrix natural logarithm if $\boldsymbol{A}$ is a positive definite matrix

Indexing

a_i	i-th element of vector $\boldsymbol{a}$
$\boldsymbol{a}_j$	j-th column of matrix $\boldsymbol{A}$
$a_{i,j}$	Element (i, j) of matrix $\boldsymbol{A}$
$\boldsymbol{\mathcal{A}}_{i_1,\ldots,i_N}$	Element $(i_1, \ldots, i_N)$ of tensor $\boldsymbol{\mathcal{A}}$

Linear and Tensor Algebra Operations

$\boldsymbol{a}^\top \boldsymbol{A}^\top$	Transpose operator for vector $\boldsymbol{a}$ and matrix $\boldsymbol{A}$
$\odot$	Element-wise (Hadamard) product for vectors, matrices and tensors
$\otimes$	Outer (Kronecker) product for vectors
$\langle \cdot \,;\, \cdot \rangle$	Inner (dot) product for vectors, matrices and tensors
$\|\cdot\|_p$	ℓ_p-norm for vectors
$\|\cdot\|_{\boldsymbol{\mathcal{F}}}$	Frobenius norm for matrices and tensors

Probability and Information Theory

$a \sim P$ Random variable a with distribution P

$\mathbb{E}_{a \sim P}$ Expectation over a drawn from distribution P

$\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ Gaussian distribution with mean $\boldsymbol{\mu}$ and covariance $\boldsymbol{\Sigma}$

Sets

$\mathbb{A}$ Set

$a \in \mathbb{A}$ element a in the set $\mathbb{A}$

$|\mathbb{A}|$ Number of elements in $\mathbb{A}$

$\mathbb{A} \cup \mathbb{B}$ Union of sets $\mathbb{A}$ and $\mathbb{B}$

$\{a, b\}$ Set composed of the elements a and b

$\mathbb{A} \setminus \{a\}$ Set $\mathbb{A}$ without the element a

$\mathbb{R}$ Set of real numbers

$\mathbb{R}^*$ Set $\mathbb{R} \setminus \{0\}$

$\mathbb{N}$ Set of natural numbers $\{0, 1, \ldots\}$

$[N]$ Set of all natural numbers from 0 to N, both included

$[N]^*$ Set $[N] \setminus \{0\}$

$\mathbb{R}^n$ Set of n-dimensional vectors with real values

$\mathbb{R}^{n \times m}$ Set of matrices with n rows and m columns and real values

$\mathbb{R}^{n_1 \times \cdots \times n_N}$ Set of N-th order real values tensors with dimensions $n_1, \ldots, n_N$ respectively

Preface

An important challenge in artificial intelligence is the learning of useful data representation, or features, on which recognition algorithms are applied in order to solve a specific task. Such tasks may be object or scene recognition, video annotation, content-based retrieval, image segmentation, weather prediction, automatic translation, *etc.*, and might involve time series, images, text, videos, and so on. By learning rich representations, that is, which encode the relevant information with respect to the given task, performances of the majority of recognition algorithms can be highly improved. The major difficulty in learning such representations rests on the high dimensionality of the input data, and the lack of prior knowledge about the relevant patterns that should be encoded. For instance, color images are very large dimensional objects with typically millions of dimensions (*i.e.*, the number of pixels) with highly correlated areas (*e.g.*, ears are close to eyes and eyes are close to the nose in a face) and redundant information (each color channel encodes the same information about the shape). However, we do not know this information (*e.g.*, which pixels belong to the ears or the eyes, and even worse, if there is a face in the input image). As such, high-level decision-making algorithms in artificial intelligence require relevant representations in order to work well in practice. Furthermore, these representations need to be low-dimensional so as to make the decision tractable. For example, in content-based image retrieval, one need to store a representation for each image in the database. Thus, high-dimensional representations will require a large amount of memory to store all representations and lots of computation to retrieve the relevant images, which leads to a slow system that might not be useful in practice, *e.g.*, for a search engine which indexes billion of images.

In this thesis, we particularly focus on image representation for the task of content-based image retrieval. As raised by the previous example, the representations should be low-dimensional in order to reduce the memory storage, the computation and to speed-up the search. Also, they should encode useful information in order to be able to retrieve the most relevant images from the database. Thus, the representations should encode the notion of similarity between images: two images that are visually close (*e.g.*, same scene, same object, *etc.*) should have their representations close to each other while visually dissimilar images should have their representations far from each other. In order to do so, we rely on the deep metric learning framework. In deep metric learning, this notion of similarity is encoded through a distance, *e.g.*, the Euclidean distance. The image representations and this distance are learned together in order to solve the task of content-based image retrieval. As a consequence, the deep metric learning framework tries to answer the following questions:

- How to efficiently build a compact but informative image representation?
- How to learn a distance that encode the visual similarity?

This thesis addresses these two points by first learning better image representations in order to explicitly encode relevant information and then by improving the training procedure. We propose to build richer representations by considering high-order statistics of local features. Indeed, the use of covariance matrices as image representations has been shown to outperform standard approaches on a large variety of computer vision tasks, and especially for fine-grained tasks. Covariance matrices better encode finer details within the images by considering the co-occurrence of patterns in the images. However, using second-order (and higher-order) statistics raises many problems. The first one is the dimensionality of the representation: typical covariance matrices have more than hundred of thousand dimensions which makes them intractable for tasks where the image representations must be stored. The second one is that covariance matrices lie on a Riemannian manifold. This means that the Euclidean distance cannot correctly represent the data, which limit their usage in metric learning. As a complement, the training procedure of metric learning leads to sampling issues because the loss functions rely on tuples of similar or dissimilar images, and finding the informative ones (*i.e.*, the ones that generates gradients) becomes harder and harder as the training progresses. As such, we propose to train a deep metric learning network and a generator together in such a way that the latter synthesizes hard examples that generates gradients for the the former. Also, we show that the local features that are trained with the image representation and the metric tend to not be

robust because the training only optimizes the image representations. As such, we propose to tackle this phenomenon by considering regularization.

In this thesis, we make the following contributions to alleviate the aforementioned issues: First, we present three images representations that leverages dictionary learning in order to produce richer representation along covariance matrices or attention mechanisms. Second, we propose two improvements during the training: a method that generates hard examples to resume the training, and a regularization that improves the robustness of the local features for a variety of representations. We empirically show that the proposed strategies significantly improve baseline methods and provide stronger results than most of the state-of-the-art methods.

This thesis is organized as follows: In a first part, we present relevant background and motivations by considering earlier image representations. Next, we detail the metric learning framework and the use of covariance matrices as global representation in dedicated chapters as long as giving some intuitions about our proposed methods in light of the literature. Then we present our contributions: In chapter 4, we present the three image representations based on dictionary learning, second-order statistics, and attention mechanisms, that are *ISTA* (ICIP 2018), *JCF* (ICIP 2019) and *DIABLO* (Pattern Recognition Letters 2020). The two next chapters are dedicated to respectively the high-order regularization method named *HORDE* (ICCV 2019) and the generation-based methods for metric learning named *MIRAGE*.

Part I

Background and Motivations

1

Image Representation Learning

In this chapter, we present all materials that are related to similarity measures between images, from feature matching approaches to feature aggregation and image embedding. A particular focus is given on dictionary-based aggregation for large-scale image search applications.

Contents

1.1 Introduction **4**
1.2 Global representations **4**
1.3 Local features **5**
1.4 Aggregation methods **6**
1.5 Discussion **10**

1.1 Introduction

Performances of the majority of machine learning algorithms heavily depends on the choice of a data representation, or features, on which these methods are applied. Thus, designing features that better represent the information contained in the data is primordial to build algorithms with higher performances. For that reason, feature engineering has been the nerve of research in computer vision in the early 2000s, from the use of the Histogram of Oriented Gradients (HOG) for pedestrian detection [120, 39] to Oriented FAST and rotated BRIEF (ORB) [147] through Scale-Invariant Feature Transform (SIFT) [117]. These features have faced a trade-off between preserving as much information about the input as possible and having nice properties such as smoothness, natural clustering, sparsity, and so on. This approach is very important so as to get better generalization properties: These features can then be used for other tasks (classification, regression, segmentation, *etc.*) or applications (pedestrian detection, object localization, scene classification, *etc.*).

These shallow features are nowadays replaced by deep features in convolutional neural networks. Indeed, they lead to higher performances for a large variety of computer vision tasks, even when they are used off-the-shelf [153] in replacement of the previous shallow features. This approach is usually referred to *transfer learning* and has led to great advances in the computer vision community. Also, they can be trained end-to-end for a specific task, which increases their discriminative power as well as their performances.

In this chapter, we focus on representation learning for content-based image retrieval (CBIR). CBIR is the task of searching in a collection of images the most relevant ones with respect to a query image by only relying on the content of the images. This computer vision task has a wide range of real applications, such as the visual search of products [127, 115], the face verification and clustering [152], the re-identification applications for persons [206], vehicles [112] or animals [151] and so on. Also, this task has an even wider range of applications if we consider other or multiple modalities such as cross-modal retrieval (text-image [176], video [155], *etc.*), text retrieval [7] and so on.

In particular, we first introduce the earlier approaches based on color histogram [162] or texture-based representations [118] as global representations to measure similarity between images. Then, we present the strategies based on local features such as SIFT [117] and feature matching approaches [117, 128]. From these local features, we detail global representations based on aggregating local features such as Bag-of-Features (BoF) [36], VLAD [91] and Fisher Vectors [134, 150]. We review some of their extensions in light of the deep learning framework by considering aggregation-based method on deep local features [153] and end-to-end trainable formulations such as *e.g.*, Radial Basis Function (RBF) for differentiable BoF [131], soft-assignment in VLAD to build NetVLAD [1] or learnable parameters in the Fisher Vectors [165]. Finally, we discuss about fast indexing approaches using vector quantization [88, 5] and approaches that go beyond instance-level retrieval [63].

1.2 Global representations

In this section, we give an overview of the earlier approaches to build global representations for image retrieval. Color histogram [162] has been among the first methods adopted in computer vision to analyze images. For each image, a global representation is computed from the concatenation of empirical distribution of pixel values from a given 3D color space such as the standard RGB or CIE-Lab. By using a set of n bins per dimension, a global representation of size $3n$ is used to compute similarity between two images. The similarity measure can be performed in a multiple way: Swain and Ballard [162] explore different similarity measure such as histogram intersection or the ℓ_1-distance. Rubner et al. [148] extend this approach by considering normalized histogram as probability density functions. By doing so, they take advantage of divergence between distributions in order to compute the image similarity. That is, they consider the Kullback-Leibler divergence, the Jeffrey divergence but also the Earth Mover distance as similarity measure and evaluate the benefits of such approaches compared to the histogram intersection or the Minkowski distances. Adaptive color quantization [35] instead of color histograms can be used to build smaller global representations.

This approach better exploits the data by learning the quantization (*e.g.*, using K-means) in order to build smaller histograms, thus smaller representations, while having better performances.

Additionally to color representations, texture-based approaches have been explored through multi-resolution methods using, *e.g.*, wavelets or Gabor filters [118]. Gabor filters form a complete and self-similar basis but which is non-orthogonal. This basis is usually referred as Gabor wavelets, in the sense that each filter is localized in space and frequency. Each filter can be computed from a mother Gabor filter by appropriate dilatation and rotation. By doing so, an image can be represented by filtering itself with each Gabor wavelet and by computing statistics over the filtered image. Typically, the texture representation is obtained from the concatenation of the mean and the standard deviation of each Gabor wavelet. A texture representation can be computed using Gabor filters: By using many Gabor wavelets, the texture representation is computed by concatenating the mean and the standard deviation of the image filtered by each Gabor wavelet. Thus, image similarity is simply computed by taking ℓ_1 or ℓ_2 distance between their respective representations.

1.3 Local features

First developed for reconstructing 3D structure from multiple images, stereo correspondence or motion tracking, local features have been widely used for a large variety of computer vision tasks from pedestrian detection [39] to image retrieval [36, 91]. These local features are built in order to be invariant to scaling and rotation, illumination change or 3D camera viewpoint. Also, they should be well localized in both spatial and frequency domains in the same way as wavelets. By doing so, such features could be very precise and robust for most of computer vision tasks. Earlier work in local features comes from 1986 with the McConnell's patent [120] which first described the well-known Histogram of Oriented Gradients (HOG) which have become popular with the work of Dalal and Triggs on pedestrian detection [39]. The main idea of HOG is that object appearance and shape can be described using the distribution of the gradients. By first dividing the image into small non overlapping squares, the histogram of gradients is computed for each square. The histogram is built by binning the angle of the gradient and by reporting the norm of the gradient. The local feature is got by concatenating the histogram for each pixel within the square. Also, normalization can be performed in order to make this local feature invariant to illumination change.

Few years later, Lowe proposed the SIFT [116, 117] which starts from a keypoint detector before extracting the local feature in order to avoid the dense computation of HOG. By using difference-of-Gaussian, the first stage identifies potential keypoint candidate that are invariant to affine transformations. This is followed by rejecting candidates that are localized in area with low contrast (sensitive to noise) or localized along an edge by using the Hessian matrix as it is done in the Harris detector [71]. Then, an orientation histogram is computed from the gradient orientations of sample points within the keypoint's neighborhood to introduce the rotation invariance property. By doing so, the keypoint orientation is set to the highest peak of the histogram, but also to any others peaks that are greater than 80% of the highest peak: T simply leads to add multiple keypoints at the same spatial location but for different orientations. Finally, for all of these keypoints we extract a SIFT feature. Usually, a window of size 16×16 is formed around the keypoint in order to compute the SIFT feature. This window is split into 16 smaller windows of size 4×4 where gradient magnitudes and orientations are computed. Then, for each of these smaller windows, the gradient magnitudes are aggregated into a 8 bins histogram of the gradient orientations weighted by a Gaussian centered on the keypoint: gradients computed far from the keypoint have less impact than ones that are close to the keypoint. Finally, the SIFT feature for the given keypoint is the $4 \times 4 \times 8$ flattened feature. Many extensions of these local features have been proposed such as the SURF [11, 10] which speeds-up the SIFT with approximate keypoint computation and faster feature computation, BRIEF [22, 21] and ORB [147] which are binary features, DAISY [168] which are also designed to be fast, and so on.

In light of recent work in deep learning, the use of deep local features extracted from convolutional neural networks has been shown to outperform the aforementioned local features (now referred as shallow local features) in many computer vision tasks [153]. Indeed, by first pre-training a deep network on a pretext task such as classification on ImageNet [45], one can drop the top layers and use the feature map computed from the last stage of convolution layers. The learned features have

been shown to transfer to a large variety of tasks with results that are higher than using shallow features extracted. On the top of that, the possibility of training deep local features in an end-to-end manner to be discriminative for the new task has led to great improvements.

Once these local features are extracted, they can be matched between two images in order to find the corresponding pairs so as to take a decision of whether the two images are similar or dissimilar. For example, a well-known approach in image correspondence is to use Random Sample Consensus (RANSAC) [53] to estimate the homography (*i.e.*, the viewpoint change of the scene) between the two images by finding the set of local feature pairs that are matched between the two images and that are transformed in the same way. Approximate nearest-neighbors [117, 128] can also be performed in order to accelerate the matching of local features using Best-Bins-First [12, 117] or meaningful nearest-neighbors [128].

Feature matching approaches are inefficient in the case of image retrieval because we need to store all features that have been extracted from each image in the collection. When finding the most similar images with respect to a query image, the correspondence has to be performed between the query image and each image from the collection. Thus, this approach is not efficient neither in memory usage due to the storage of all local features nor in computation because the matching is quadratic in the number of extracted features and linear in the size of the image collection. Therefore, approximate nearest neighbor search strategies have to be designed, such as Locality-Sensitive Hashing [40], product quantization [89], and so on.

As a conclusion, local features have been shown to outperform most of global representations from section 1.2 for a wide variety of computer vision tasks. By extracting vectors that describe the local content on an image, these local features have been applied for 3D reconstruction, stereo correspondence but also for pedestrian detection and image retrieval. Feature maps extracted from deep convolutional neural network have been shown to transfer to a large variety of tasks with even greater performances than "shallow" local features. They can be either pre-trained on a pretext task such as classification on ImageNet [153], with weak-supervision [125] or directly on the task, using *e.g.* self-supervision [46]. Still, matching local features using *e.g.* RANSAC is costly in the case of image retrieval: local features have to be store for all images and the matching has to be performed between the query image and all images from the collection. Furthermore, the matching is quadratic in the number of local features and does not scale to large datasets without proper approximate search such as Locality-Sensitive Hashing [40] or product quantization [89].

1.4 Aggregation methods

One of the great advantage of global representations is that they are easier to store and need less computation to perform the retrieval than matching based approaches. A compromise between the discriminative power of the local features and the performance of global representations in to consider aggregation methods. In this section, we detail the following aggregation methods: Bag-of-Features [36], VLAD [91], Fisher Vectors [134, 150], VLAT [139, 140] and STA [138]. In Bag-of-Features [36], the representation is computed as the histogram of the number of occurrence of given patterns in the image from a set of local features. In order to build the representation, the first step is to define a dictionary of patterns. To do so, a set of cluster centers is learned on local features extracted from a set of training images, *e.g.*, using K-means. The dictionary of patterns is then composed of the cluster centers that have been learned. Then, for a given image, we extract local features and we assign each feature to its nearest dictionary entry. The representation is computed by building the histogram of the number of features that belong to a given dictionary entry which leads to a N-dimensional representation, where N is the size of the dictionary. In practice, large dictionaries have to be used in order to get good performances. For instance, image retrieval on Oxford5k [136] with 1 million distractors (*i.e.*, 1 million images from other topics than buildings from Oxford) require a dictionary size of 1 million in order to reach greatest performances.

VLAD [91] extends the bag-of-Features approach by considering the local features into the representation. To build the representation, a dictionary is learned similarly to the Bag-of-Features approach. However, instead of counting the number of local feature per dictionary entry, the VLAD

representation aggregates the difference between the local feature and the dictionary entry in order to characterize the distribution of the vector with respect to the center. Compared to the Bag-of-Features, the VLAD representation leads to a representation of size Nc where N is the dictionary size and c is the dimensionality of the local features. As a consequence, smaller dictionary are required in order to keep small representation. However, aggregating the difference between local features and the dictionary entry contains more information: smaller VLAD representations lead to higher performances than Bag-of-Features in practice. Moreover, these representation are easier to compress by considering dimensionality reduction methods (*e.g.*, Principal Component Analysis) and hashing strategies (*e.g.*, Locality-Sensitive Hashing or product quantization). Indeed, the authors show that even with only 16bits the compressed VLAD representation still leads to higher performances than Bag-of-Features on the Holidays dataset [88]. Finally, one can note that the Bag-of-Features computes 0^{th}-order statistics (*i.e.*, the probability) for each dictionary entry while the VLAD representation computes 1^{st}-order statistics (*i.e.*, the mean of the local features) for each dictionary entry.

VLAT [139, 140] further extends the Bag-of-Features and VLAD by considering second-order statistics. Instead of aggregating the difference between the features and the center, VLAT aggregates the difference between the second-order moment estimated with the local features from a single image that belongs to the cluster and the average second-order moment estimated on a training set. By doing so, VLAT leads to a representation of size Nc^2 which is too large to be tractable in practice. To cope with the dimensionality of the representation, the authors have used the Principal Components Analysis to reduce the dimensionality of local features. In practice, the use of second-order statistics leads to better results than 0-th order (Bag-of-Features) and 1st order (VLAD).

At the same time, Perronnin et al. [134, 150] have proposed the Fisher Vectors. The idea behind Fisher Vectors comes from the Fisher Kernel formulation [83]: the generative process of local features is modeled by a parametric probability density function $f_{\boldsymbol{\lambda}}$ with M parameters noted as $\boldsymbol{\lambda} = \{\lambda_1, \ldots, \lambda_M\}$. Thus, the score function $G_{\boldsymbol{\lambda}}(\boldsymbol{X})$ given by the gradient of the log-likelihood of the data on the model can be used to describe the contribution of each parameter to the generative process. This gradient can be represented by a vector in $\mathbb{R}^M$ and is empirically estimated for each image using the local features. Jaakkola and Haussler [83] have shown that this vector lies on a Riemannian manifold and have proposed to use the Fisher Kernel in order to compute a similarity measure as follows:

$$K(\boldsymbol{X}, \boldsymbol{Y}) = G_{\boldsymbol{\lambda}}(\boldsymbol{X})^\top \boldsymbol{F}_{\boldsymbol{\lambda}}^{-1} G_{\boldsymbol{\lambda}}(\boldsymbol{X}) \tag{1.1}$$

where $\boldsymbol{X}$ and $\boldsymbol{Y}$ are two sets of local features extracted from two images and $\boldsymbol{F}_{\boldsymbol{\lambda}}$ is the Fisher information matrix computed as follows:

$$\boldsymbol{F}_{\boldsymbol{\lambda}} = \mathbb{E}_{\boldsymbol{X} \sim f}[G_{\boldsymbol{\lambda}}(\boldsymbol{X}) G_{\boldsymbol{\lambda}}(\boldsymbol{X})^\top] \tag{1.2}$$

Because the Fisher information matrix is symmetric definite positive, its inverse can be factorized using the Cholesky decomposition and the above kernel can be re-written as a simple dot product. In the case of Fisher Vectors, the probability density function $f_{\boldsymbol{\lambda}}$ is chosen to be a Gaussian Mixture Model:

$$f_{\boldsymbol{\lambda}}(\boldsymbol{x}) = \sum_{k=1}^{N} \alpha_k f_{\boldsymbol{\lambda}_k}(\boldsymbol{x}) \tag{1.3}$$

$$= \sum_{k=1}^{N} \alpha_k \frac{1}{(2\pi)^{\frac{c}{2}} \sqrt{\det(\boldsymbol{\Sigma})}} \exp\left(-\frac{1}{2}(\boldsymbol{x} - \boldsymbol{\mu}_k)^\top \boldsymbol{\Sigma}_k^{-1} (\boldsymbol{x} - \boldsymbol{\mu}_k)\right) \tag{1.4}$$

where $\alpha_k \in \mathbb{R}^+$ with $\sum_k \alpha_k = 1, \boldsymbol{\mu}_k \in \mathbb{R}^c$ and $\boldsymbol{\Sigma}_k \in \mathbb{R}^{c \times c}$ are respectively the weight, the mean and the covariance matrix of the k-th Gaussian and N is the number of Gaussian. In this case, the parameters $\boldsymbol{\lambda}$ are $\alpha_1, \ldots, \alpha_N, \boldsymbol{\mu}_1, \ldots \boldsymbol{\mu}_N, \boldsymbol{\Sigma}_1, \ldots, \boldsymbol{\Sigma}_N$. In order to ease the computation of the gradient, the covariance matrices are supposed to be diagonal which leads to Fisher Vectors with $2N(c+1)$ dimensions. Because the covariance matrix is diagonal, the Cholesky decomposition leads to a diagonal matrix: taking into account the Fisher information matrix into the representation can be computed with element-wise product. Finally, each element of $G_{\boldsymbol{\lambda}}(\boldsymbol{X})$ is computed using the

empirical estimator. For the following equation, we denote $\mu_{k,i}$ the i-th element of the mean $\boldsymbol{\mu}_k$ and $\sigma^2_{k,i}$ the i-th diagonal element of $\boldsymbol{\Sigma}_k$ where k corresponds to the k-th dictionary entry. The elements $G_{\alpha_k}(\boldsymbol{X}), G_{\boldsymbol{\mu}_{k,i}}(\boldsymbol{X})$ and $G_{\sigma_{k,i}}(\boldsymbol{X})$ from $G_{\boldsymbol{\lambda}}(\boldsymbol{X})$ are computed as follows:

$$G_{\alpha_k}(\boldsymbol{X}) = \frac{1}{\sqrt{\alpha_k}} \sum_{\boldsymbol{x} \in \boldsymbol{X}} \gamma_k(\boldsymbol{x}) - \alpha_k \tag{1.5}$$

$$G_{\boldsymbol{\mu}_{k,i}}(\boldsymbol{X}) = \frac{1}{\sqrt{\alpha_k}} \sum_{\boldsymbol{x} \in \boldsymbol{X}} \gamma_k(\boldsymbol{x}) \frac{x_i - \mu_{k,i}}{\sigma_{k,i}} \tag{1.6}$$

$$G_{\sigma_{k,i}}(\boldsymbol{X}) = \frac{1}{\sqrt{\alpha_k}} \sum_{\boldsymbol{x} \in \boldsymbol{X}} \frac{\gamma_k(\boldsymbol{x})}{\sqrt{2}} \left(\left(\frac{x_i - \mu_{k,i}}{\sigma_{k,i}} \right)^2 - 1 \right) \tag{1.7}$$

where $\boldsymbol{x} \in \boldsymbol{X}$ is a local feature, x_i is its i-th element and $\gamma_k(\boldsymbol{x})$ is the weight assigned to $\boldsymbol{x}$ with respect to the k-th dictionary entry computed as follows:

$$\gamma_k(\boldsymbol{x}) = \frac{\alpha_k f_{\boldsymbol{\lambda}_k}(\boldsymbol{x})}{\sum_{l=1}^{N} \alpha_l f_{\boldsymbol{\lambda}_l}(\boldsymbol{x})} \tag{1.8}$$

Compared to the previous methods, the Fisher Vectors extend these representations by taking into account the 0-th order statistics (G_{α_k}) like Bag-of-Features, the first-order statistics ($G_{\boldsymbol{\mu}_k}$) like VLAD and the second-order statistics ($G_{\boldsymbol{\Sigma}_k}$) like VLAT. The main difference is that the Fisher Vectors consider diagonal second-order moment whereas VLAT considers the full matrices. Also, Fisher Vectors leads to higher performances than Bag-of-Features, VLAD and VLAT representations.

Spatial Tensor Aggregation (STA) [138] is a recently proposed aggregation method which has been designed to explicitly encode spatial information in the representation. To do so, they start from a keypoint matching method and linearize it using a tensor framework to get the image representation. For two images $\boldsymbol{\mathcal{I}}$ and $\boldsymbol{\mathcal{J}}$, we note $\boldsymbol{x}$ and $\boldsymbol{y}$ the set of local features extracted from images $\boldsymbol{\mathcal{I}}$ and $\boldsymbol{\mathcal{J}}$ respectively. We note $\Omega(\boldsymbol{x})$ the set of local features that are in the neighborhood of $\boldsymbol{x}$. Also, we note $k(\cdot, \cdot)$ a similarity measure. The keypoint matching method that encodes the local spatial information in STA, named K is defined as follow (Equations 3 and 7 from [138]):

$$K(\boldsymbol{\mathcal{I}}, \boldsymbol{\mathcal{J}}) = \sum_{\boldsymbol{x} \in \boldsymbol{\mathcal{I}}} \sum_{\boldsymbol{y} \in \boldsymbol{\mathcal{J}}} \sum_{\boldsymbol{x}_u \in \Omega(\boldsymbol{x})} \sum_{\boldsymbol{y}_v \in \Omega(\boldsymbol{y})} k(\boldsymbol{x}, \boldsymbol{y}) k(\boldsymbol{x}_u, \boldsymbol{y}_v) \tag{1.9}$$

Equation 1.9 supposes that there is only a small deformation on the content between two images, such that if a second descriptor is in the spatial neighborhood of the first one for the image $\boldsymbol{\mathcal{I}}$, the same features with the same pattern should also appears in image $\boldsymbol{\mathcal{J}}$. Thus, Equation 1.9 encodes this information for each matching pair of features $(\boldsymbol{x}, \boldsymbol{y})$ from images $\boldsymbol{\mathcal{I}}$ and $\boldsymbol{\mathcal{J}}$ by counting the number of matches between the features from their respective neighborhood. Intuitively, this matching kernel counts the number of matches between $\Omega(\boldsymbol{x})$ and $\Omega(\boldsymbol{y})$ if and only if both the central features $\boldsymbol{x}$ and $\boldsymbol{y}$ matches. This is comparable to geometric consistency in image search [88] where similarities between irrelevant features are not taken into account.

The kernel matching K is then linearized by using the dot product as the kernel similarity k and the Kronecker product $\otimes$ (Equation 8 from [138]):

$$K(\boldsymbol{\mathcal{I}}, \boldsymbol{\mathcal{J}}) = \left\langle \sum_{\boldsymbol{x} \in \boldsymbol{\mathcal{I}}} \sum_{\boldsymbol{x}_u \in \Omega(\boldsymbol{x})} \boldsymbol{x} \otimes \boldsymbol{x}_u \; ; \; \sum_{\boldsymbol{y} \in \boldsymbol{\mathcal{J}}} \sum_{\boldsymbol{y}_v \in \Omega(\boldsymbol{y})} \boldsymbol{y} \otimes \boldsymbol{y}_v \right\rangle \tag{1.10}$$

However, small non-zero similarities are added for all local features because of the dot product, which adds a non-negligible quantity of noise (quadratic in the number of local features) into the representation. To avoid this phenomenon, a dictionary-based approach similar to the one in VLAD is exploited. Thus, Equation 1.9 is modified to consider an assignment function $\boldsymbol{h}$ such that $\boldsymbol{h}(\boldsymbol{x})$ is the one-hot encoding of the local feature $\boldsymbol{x}$ with respect to the n-th dictionary entry:

$$K(\boldsymbol{\mathcal{I}}, \boldsymbol{\mathcal{J}}) = \sum_{\boldsymbol{x} \in \boldsymbol{\mathcal{I}}} \sum_{\boldsymbol{y} \in \boldsymbol{\mathcal{J}}} \sum_{\boldsymbol{x}_u \in \Omega(\boldsymbol{x})} \sum_{\boldsymbol{y}_v \in \Omega(\boldsymbol{y})} \langle \boldsymbol{h}(\boldsymbol{x}) \; ; \; \boldsymbol{h}(\boldsymbol{y}) \rangle \langle \boldsymbol{h}(\boldsymbol{x}_u) \; ; \; \boldsymbol{h}(\boldsymbol{y}_v) \rangle \, k(\boldsymbol{x}, \boldsymbol{y}) k(\boldsymbol{x}_u, \boldsymbol{y}_v) \tag{1.11}$$

Using the linearization of Equation 1.10, we get the following STA representation:

$$K(\mathcal{I},\mathcal{J}) = \left\langle \sum_{\boldsymbol{x}\in\mathcal{I}} \sum_{\boldsymbol{x}_u\in\Omega(\boldsymbol{x})} \boldsymbol{h}(\boldsymbol{x}) \otimes \boldsymbol{h}(\boldsymbol{x}_u) \otimes \boldsymbol{x} \otimes \boldsymbol{x}_u \;;\; \sum_{\boldsymbol{y}\in\mathcal{J}} \sum_{\boldsymbol{y}_v\in\Omega(\boldsymbol{y})} \boldsymbol{h}(\boldsymbol{y}) \otimes \boldsymbol{h}(\boldsymbol{y}_v) \otimes \boldsymbol{y} \otimes \boldsymbol{y}_v \right\rangle \quad (1.12)$$

Finally, for a given image $\mathcal{I}$, the STA representation $\mathcal{T}(\mathcal{I})$ is computed as follow:

$$\mathcal{T}(\mathcal{I}) = \sum_{\boldsymbol{x}\in\mathcal{I}} \sum_{\boldsymbol{x}_u\in\Omega(\boldsymbol{x})} \boldsymbol{h}(\boldsymbol{x}) \otimes \boldsymbol{h}(\boldsymbol{x}_u) \otimes \boldsymbol{x} \otimes \boldsymbol{x}_u \quad (1.13)$$

Interestingly, this formulation is tightly related to VLAD and VLAT by extending the first-order statistic to the second-order statistic compared to VLAD and by considering correlation between nearby local features compared to VLAT.

Most of these representations are not differentiable and cannot be used to train in an end-to-end manner a deep network in order to fine-tune both the global representation and the local feature for a given task. Thus, recent work have extended these representations to be differentiable. In the case of Bag-of-Features, the hard assignment to one of the dictionary entry can be replaced by a soft assignment computed using *e.g.* radial basis functions [131]. By computing the similarity with the radial basis functions and by soft assigning each local feature to a dictionary entry using the softmax operator, both the Bag-of-Features and the network parameters can be trained together for a specific task such as classification or retrieval. Similarly, NetVLAD [1] replace the hard assignment with the same approach but also the output projection that is usually computed using a Principal Components Analysis by a fully-connected layer. The architecture can be trained end-to-end for a specific task such as retrieval [1] or long-term visual localization [58]. Thanks to the formulation of Fisher Vectors, making the parameters trainable in the computation is enough to train both the representation and the local features together [165]. However, these representations are usually disregarded in most of recent deep network architectures. The main reason is that they are not required to reach high performances: simpler and more compact representations such as global average or maximum pooling are enough in practice because the local features are more discriminative than SIFT or HOG. Furthermore, large dictionaries are harder to train end-to-end without proper regularization. Indeed, few dictionary entries are used in practice and most of the local features are assigned to two or three entries.

Despite the phenomenon, specific image representations are favored in some tasks such as fine-grained image classification. Fine-grained image classification is the task of classifying objects into classes that only experts can perform, *e.g.* classifying an image as a red-faced cormorant or a pelagic cormorant (see Figure 1.1). For such tasks, second-order based representations such as bilinear pooling [110] are usually preferred as they lead to higher performances by capturing more fine-grained information. For the sake of clarity, we review these approaches in chapter 3 as they are tightly related to our contributions in tensor-based aggregation from the chapter 4. Furthermore, we present the framework of metric learning which is usually used to train both the image representation and the deep network in an end-to-end manner in chapter 2.

In conclusion, image representation based on aggregating local features are a good compromise in terms of computation and performances compared to feature matching in the case of image retrieval task. Most renowned approaches like Bag-of-Features or VLAD are unsupervised representations that exploit dictionary learning. However, to compete with the end-to-end training methods in deep learning, these representations have been modified to be differentiable in order to optimize the local deep features and the representation together for a given task. In most of the recent deep learning architectures, these representations have been disregarded to consider simpler but still efficient representations such as global average pooling, global maximum pooling or their extensions to region-based pooling [52, 169, 61, 62]. This is due to two main advantages: first, the end-to-end training which leads to very discriminative local features for any representation. Second, indirect training of dictionaries is harder and tends to favor only few entries. Still, other representations for particular tasks (*e.g.*, bilinear pooling for fine-grained image classification) are considered because they lead to higher performances in practice thanks to the higher-order statistics.

(a) Red-faced Cormorant (b) Pelagic Cormorant

Figure 1.1: Example of images for the fine-grained classification task. Both birds are cormorant but from a different breed. They have a very similar plumage and the main different is the red mark on the left bird's face.

1.5 Discussion

In this section, we first sum-up the entire chapter before discussing the presented points in light of our contributions. In a first part, we have presented the very first approaches for global image representation. By considering color histogram or texture-based approaches, an image can be represented by a simple vector from which a similarity measure based on ℓ_1 or ℓ_2 norm can be computed. However, these methods are not naturally invariant to *e.g.*, illumination change, scaling, rotation, 3D camera viewpoint, and so on. In a second part, methods that compute local features inspired by 3D reconstruction or stereo matching have been considered to compare images. By designing strongly invariant local features, *e.g.*, rotation and scaling invariance, 3D camera viewpoint invariance, *etc.*, such features are very discriminative for most computer vision tasks. In the case of image retrieval, one can consider matching local features between two images and counting them in order to measure the similarity between the two images. That is, the more local features are matched between the two images, the more the images are similar. However, such approaches are very costly in the case of image retrieval in term of computation, complexity and memory usage. Indeed, local features have to be computed and stored for all images within the collection and, at query time, the matching has to be done between the query image and each image from the collection. Thus, the matching which has a computational complexity in the square of the number of features is very costly and the storage of all local features leads to a large memory footprint. In a third part, we have detailed aggregation-based methods such as Bag-of-Features or VLAD. Global representations constructed from the local features have been shown to be a better compromise to feature matching for the image retrieval task while performing better than the earlier global representations. We have presented the most common techniques to build global representations using *e.g.*, Bag-of-Features, VLAD or Fisher Vectors but also two other approaches that aggregates high-order statistics with or without spatial local information (VLAT and STA). Finally, binary codes for image representations with off-the-shelf approaches [5, 88] or end-to-end trainable [23] is a popular way for fast indexing in image retrieval. Note that they are not discussed in this thesis because they are complementary work that can be plugged on top of any global representation.

We now discuss our contributions in tensor-based aggregation in light of these aggregation-based methods. In a first part, we extend the STA and the VLAT representations. Our first contribution, named Improved STA, modify the STA representation by considering a normalization step and a two-step factorization scheme. The idea is to exploit the local spatial information and the second-order statistics but at a reasonable cost in term of computation and memory storage. While we empirically show the benefit of normalization, its necessity is further discussed first in chapter 3

and then in chapter 4. Furthermore, we show that the two-step factorization greatly decreases the dimensionality of the representation with only small loss in performance. Also, we can explore larger dictionaries to further increase performances. Our second contribution, named *JCF*, is inspired by the work on NetVLAD and considers differentiable factorization which makes representations like VLAT or STA differentiable and trainable end-to-end. Moreover, the proposed factorization leads to very small representations with good empirical performances. Finally, we discuss in chapter 4 the links between our third contribution *DIABLO*, the proposed factorization and representations like NetVLAD, a differentiable version of STA.

In a second part, we propose two approaches to improve the training of deep networks through supervised learning. Our first contribution, named *HORDE*, considers the distribution of deep local features and tries to minimize a divergence between distributions in the case of similar images and to maximize it otherwise. By doing so, we show that local features are more robust to outliers and to sampling issues to build global representations on top of them. This approach could benefit from learning a dictionary for such representations. Indeed, matching distributions or making them disjoint encourages the distribution to be well localized, separated and compact. Thus, fewer dictionary entries should be required in order to encode the information contained in the distribution of local features. Our second contribution, *MIRAGE*, considers prototypes of virtual classes in order to generate hard examples. *MIRAGE* increases the margin between dissimilar features by construction, which naturally leads to well-defined dictionary entries. While being independent from the global representation, we argue that *MIRAGE* could benefit to other representations such as NetVLAD, *etc.*

In the next chapters, we focus on the deep metric framework which is used to train deep networks for the image retrieval task. We present the problem formulation and the recent contributions concerning loss functions, example mining and ensemble methods. Also, we present another global representation named bilinear pooling (or second-order pooling for uni-modal tasks) which can be seen as the special case of VLAT without a dictionary. We present the research direction and the recent contributions concerning the geometry of the manifold, the dimensionality reduction steps and related representations such as the Gaussian pooling.

2

Metric Learning

In this chapter, we formulate the metric learning problem and we present all the required materials to train a deep metric learning model and the recent contributions in the field.

Contents

2.1 Introduction **14**
2.2 Formulation **14**
2.3 Loss Functions **17**
2.3.1 Contrastive and triplet losses improvements 17
2.3.2 New designs 19
2.4 Sampling Strategies **20**
2.4.1 Approximate example search 20
2.4.2 Hard example generation 21
2.5 Ensemble Methods **22**
2.6 Datasets and Evaluation Protocol **23**
2.6.1 Datasets 23
2.6.2 Metrics 24
2.7 Discussion **26**

2.1 Introduction

One way to solve the CBIR task is to exploit the deep metric learning (DML) framework. The seminal work of Xing et al. [192] has demonstrated that metric learning as a convex optimization problem benefits a large set of applications, especially data clustering. Then, important work such as NCA [59], LMNN [187, 188] or ITML [41] have followed by improving the formulation, finding better algorithms to ease the optimization, *etc.* In most of metric learning tasks, the representations are usually fixed and only the metric is learned. In certain case, the problem is not linear which means that most of the aforementioned methods do not work well in practice. Thus, non-linear metric learning has been explored, either by using the kernel trick [28] or by using different strategies such as gradient boosting or different distance such as the χ^2-distance [96]. In the case of images, both the image representations and the metric are learned such that two visually-related images have image representations that are close to each other with respect to the learned metric and two unrelated images have their representations far apart. Metric learning on images has been widely studied by the computer vision community, also due to its large field of applications. Indeed, learning a structured space which corresponds to the semantic of the images is highly beneficial for others tasks such as image classification [187, 188], low-shot classification (few, one and zero shot) [157, 70, 184, 29], low-shot detection [8, 95], GANs [191], Adversarial defense [119], and so on.

In this chapter, we give an overview of the deep metric learning framework for content-based image retrieval tasks. In a first time, we formulate the metric learning problem and we present the standard pipeline to train a deep metric learning network, including the standard loss functions and the batch construction. In a second time, we discuss the intuitions behind the design of recent loss functions, and how they address some issues during the training. In a third time, we present the recent contributions in example mining. In a last time, we explore the recent contributions in ensemble methods with a dedicated formulation to the deep metric learning framework. Finally, we present the datasets used in this thesis and the standard metrics to evaluate the models.

2.2 Formulation

Deep metric learning aims at learning the image representations and a corresponding metric together such that the semantic in the visual content corresponds with the similarity measure between the image representations. We consider in the rest of this chapter that image representations are computed by following chapter 1 and that they are defined on the vector space with real values $\mathbb{R}^d$ in order to ease the notations. Thus, for all vectors $\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}$ in $\mathbb{R}^d$, $d(\cdot, \cdot)$ is a metric if it has the following properties:

$$\text{Definiteness:} \qquad d(\boldsymbol{x}, \boldsymbol{y}) = 0 \iff \boldsymbol{x} = \boldsymbol{y} \tag{2.1}$$

$$\text{Triangle Inequality:} \qquad d(\boldsymbol{x}, \boldsymbol{z}) \leq d(\boldsymbol{x}, \boldsymbol{y}) + d(\boldsymbol{y}, \boldsymbol{z}) \tag{2.2}$$

$$\text{Symmetry:} \qquad d(\boldsymbol{x}, \boldsymbol{y}) = d(\boldsymbol{y}, \boldsymbol{x}) \tag{2.3}$$

In practice, the definiteness property is usually dropped. Indeed, for two image representations $\boldsymbol{x}$ and $\boldsymbol{y}$ which belong to the same class, having $d(\boldsymbol{x}, \boldsymbol{y}) = 0$ means that background information is totally dropped by the image representations. A metric without the definiteness property is usually called a *pseudo-metric*. In this thesis, we use interchangeably the terms *metric* and *pseudo-metric* to simplify the explanation.

In the case of vector spaces, the Euclidean distance on vectors is the most natural metric between our image representations. However, the Euclidean distance is an isotropic distance, which means that all directions are similarly taken into account in the computation. This may bias the distance towards few directions, for example if the representations are not normalized. A widely accepted solution in deep metric learning is to replace the Euclidean distance by a learnable Mahalanobis distance [192]. The Mahalanobis distance considers a symmetric positive definite matrix $\boldsymbol{M} \in \mathbb{R}^{d \times d}$ such that the Mahalanobis distance $d_{\boldsymbol{M}}(\cdot, \cdot)$ is computed as follows:

$$d^2_{\boldsymbol{M}}(\boldsymbol{x}, \boldsymbol{y}) = (\boldsymbol{x} - \boldsymbol{y})^\top \boldsymbol{M} (\boldsymbol{x} - \boldsymbol{y}) \tag{2.4}$$

In the case of the standard Mahalanobis distance, the matrix $\boldsymbol{M}$ is the covariance matrix which can be approximated on the training set. In the case of deep metric learning, we make the matrix weights trainable to automatically learn the optimal distance. To ensure the properties of Equation 2.1, the matrix $\boldsymbol{M}$, which is a symmetric positive definite matrix, can be written as $\boldsymbol{M} = \boldsymbol{W}\boldsymbol{W}^\top$ with $\boldsymbol{W} \in \mathbb{R}^{d \times d}$. Thus, Equation 2.4 can be re-written as follows:

$$d^2_{\boldsymbol{M}}(\boldsymbol{x}, \boldsymbol{y}) = (\boldsymbol{x} - \boldsymbol{y})^\top \boldsymbol{M}(\boldsymbol{x} - \boldsymbol{y}) \tag{2.5}$$
$$= (\boldsymbol{x} - \boldsymbol{y})^\top \boldsymbol{W}\boldsymbol{W}^\top(\boldsymbol{x} - \boldsymbol{y}) \tag{2.6}$$
$$= \|\boldsymbol{W}^\top \boldsymbol{x} - \boldsymbol{W}^\top \boldsymbol{y})\|_2^2 \tag{2.7}$$

Using this factorization, we can replace the computation of the metric at each step for all image representations by a linear projection and an Euclidean distance. Also, the use of a Mahalanobis distance is the standard approach in deep metric learning due to its simplicity and effectiveness in practice. Interestingly, if we consider a *pseudo-metric*, this means that the matrix $\boldsymbol{W}$ is low-rank. Thus, we can consider a matrix $\boldsymbol{W} \in \mathbb{R}^{d \times d'}$ with $d' < d$. This has multiple advantages: First, the linear projection of $\boldsymbol{W}$ acts as a dimensionality reduction. Second, this formulation by construction integrates the Mahalanobis distance which is better than the Euclidean distance alone and we can learn the best projection to maximize the distance between images that are dissimilar and to minimize it between similar images. The vector space after the linear projection is typically called *Embedding* in the mathematical sense in topology. Indeed, a function ϕ (*i.e.*, the CNN) transforms an image into a vector where the structure is preserved: two visually-similar images have close representations and dissimilar ones have representations far away. As a side note, we discuss in Part III the choice of the metric in the general context of representation learning and its consequences on the performance of the network. In the rest of this thesis, the index $\boldsymbol{M}$ in the distance $d_{\boldsymbol{M}}(\cdot, \cdot)$ is usually dropped if there is no ambiguity.

To summarize, the deep metric learning framework relies on the following steps: (1) extraction of a set of deep local features from an image, (2) computation of the global image representation, (3) linear projection into the embedding space and (4) computation of the Euclidean distance between them. Once the representations are computed, we need to train them to structure the embedding space. Because deep metric learning relies on the notion of similarity or dissimilarity, we cannot uniformly sample examples from the training set as it is done in classification. Also, we need to build dedicated loss functions that handle this notion of similarity, *i.e.*, that penalize the distance between examples if it is too large for similar images or too small for dissimilar ones. Consequently, the rest of this section is dedicated to present two well-known loss functions (*i.e.*, the *contrastive loss* and the *triplet loss*) and their related sampling strategies. After this introduction, these two points are further discussed in section 2.3 and section 2.4 by considering the recent propositions in loss functions and sampling strategies respectively.

Because of the deep metric learning framework, we need to design dedicated loss functions to handle the notion of similarity/dissimilarity of images. To do so, pair-wise loss functions, such as the *contrastive loss*, have been proposed to optimize the deep metric learning architectures. The contrastive loss [66] is designed to increase the distance of dissimilar images to at least a margin β and to decrease the distance between two similar images:

$$\mathcal{L}_c(\boldsymbol{x}, \boldsymbol{y}) = \delta_{x,y} d^2(\boldsymbol{x}, \boldsymbol{y}) + (1 - \delta_{x,y}) \max\{0; \beta - d^2(\boldsymbol{x}, \boldsymbol{y})\} \tag{2.8}$$

where $\boldsymbol{x}$ and $\boldsymbol{y}$ are two image representations and $\delta_{x,y}$ is the indicator function which is 1 if the images are similar and 0 otherwise. In practice, we rarely use Equation 2.8 to compute the contrastive loss. Indeed, the part of the equation which is related to the dissimilar images is very easy to minimize by increasing the norm of the representations. By doing so, this leads to collapse the embedding space into a unique direction where similar images are collapsed into a single point and where the dissimilar ones are far away thanks to the norm of the representations. To avoid this phenomenon, we can add a regularization term that minimize the norm of the representations. This tends to reduce the collapse but it also adds another hyper-parameter that needs to be cross-validated. Another solution, which is usually preferred in practice, is to ℓ_2-normalize the embedding space. By projecting all representations on the unit hyper-sphere, the representations are structurally constrained: this

completely removes the aforementioned phenomenon. Interestingly, the Euclidean distance in this case is directly proportional to the *cosine similarity*:

$$d^2(\boldsymbol{x},\boldsymbol{y}) = \left\| \frac{\boldsymbol{W}^\top \boldsymbol{x}}{\|\boldsymbol{W}^\top \boldsymbol{x}\|_2} - \frac{\boldsymbol{W}^\top \boldsymbol{y}}{\|\boldsymbol{W}^\top \boldsymbol{y}\|_2} \right\|_2^2 \tag{2.9}$$

$$= \left\| \frac{\boldsymbol{W}^\top \boldsymbol{x}}{\|\boldsymbol{W}^\top \boldsymbol{x}\|_2} \right\|_2^2 + \left\| \frac{\boldsymbol{W}^\top \boldsymbol{y}}{\|\boldsymbol{W}^\top \boldsymbol{y}\|_2} \right\|_2^2 - 2 \left\langle \frac{\boldsymbol{W}^\top \boldsymbol{x}}{\|\boldsymbol{W}^\top \boldsymbol{x}\|_2} \; ; \; \frac{\boldsymbol{W}^\top \boldsymbol{y}}{\|\boldsymbol{W}^\top \boldsymbol{y}\|_2} \right\rangle \tag{2.10}$$

$$= 2 \left(1 - \left\langle \frac{\boldsymbol{W}^\top \boldsymbol{x}}{\|\boldsymbol{W}^\top \boldsymbol{x}\|_2} \; ; \; \frac{\boldsymbol{W}^\top \boldsymbol{y}}{\|\boldsymbol{W}^\top \boldsymbol{y}\|_2} \right\rangle \right) \tag{2.11}$$

Consequently, the contrastive loss can be re-written using the cosine similarity:

$$\mathcal{L}_c(\boldsymbol{x},\boldsymbol{y}) = \delta_{x,y}(1 - s(\boldsymbol{x},\boldsymbol{y}))^2 + (1-\delta_{x,y}) \max\{0; s(\boldsymbol{x},\boldsymbol{y}) - \beta\} \tag{2.12}$$

where $s(\boldsymbol{x},\boldsymbol{y})$ is the cosine similarity:

$$s(\boldsymbol{x},\boldsymbol{y}) = \left\langle \frac{\boldsymbol{W}^\top \boldsymbol{x}}{\|\boldsymbol{W}^\top \boldsymbol{x}\|_2} \; ; \; \frac{\boldsymbol{W}^\top \boldsymbol{y}}{\|\boldsymbol{W}^\top \boldsymbol{y}\|_2} \right\rangle \tag{2.13}$$

The contrastive loss is one of the most used loss functions due to its simplicity and its efficiency. Also, it usually requires few hyper-parameter tuning, and setting $\beta = 0.5$ experimentally leads to good and consistent results in deep metric learning. However, this loss function only consider global relationship between the representations, *i.e.*, two similar representations are trained to be close to each other while dissimilar are simply put far away by at least a margin β. This formulation is a good way to optimize the embedding if we consider metrics that only evaluate the very first retrieved elements such as the Recall@K (see subsection 2.6.2). However, if we evaluate ranking metrics such as the mean average precision (see subsection 2.6.2), this approach might be perfectible. Indeed, the triplet loss [152] is better suited for ranking metrics as it considers relative relationship between examples. The triplet loss is computed on a triplet of examples $(\boldsymbol{x}_q, \boldsymbol{x}_p, \boldsymbol{x}_n)$ with $\boldsymbol{x}_q$ the "query" representation, $\boldsymbol{x}_p$ a "positive" example (*i.e.*, an example that is similar to query) and $\boldsymbol{x}_n$ a "negative example" (*i.e.*, an example that is dissimilar to the query):

$$\mathcal{L}_t(\boldsymbol{x}_q, \boldsymbol{x}_p, \boldsymbol{x}_n) = \max\{0; s(\boldsymbol{x}_q, \boldsymbol{x}_n) - s(\boldsymbol{x}_q, \boldsymbol{x}_p) + \alpha\} \tag{2.14}$$

where α is a margin. Optimizing the triplet loss is a good proxy to maximize a ranking metric because a triplet loss that has a value equals to zero means that the ranking is perfect. As an overview of section 2.3, these two loss functions have numerous defaults that have been addressed in recently proposed loss functions but also better designs that have been developed.

Additionally to the loss functions, dedicated sampling strategies are primordial in deep metric learning to train the networks. Indeed, by considering pairs or triplets of examples, the number of combinations increases in $\mathcal{O}(N^2)$ and in $\mathcal{O}(N^3)$ for the pairs and the triplet loss functions respectively where N is the size of the dataset. Also, the number of available pairs of similar images is much smaller than the number of pairs of dissimilar images. Thus, during the training, this bias towards the pairs of dissimilar examples in a batch has to be handled in the loss functions, either by normalizing separately the pairs of similar and the pairs of dissimilar examples, using a specific loss design, *etc.* For example, original batch constructions have been performed by sampling $\frac{M}{2}$ pairs of similar examples and $\frac{M}{2}$ pairs of dissimilar examples to form a batch of $2M$ elements. Similarly for the triplet loss, we sample M query representations, M positive examples with respect to these queries (one for each query) and M negative examples (also one for each query) to build a batch of $3M$ elements. The advantage is that the normalization between pairs of similar and pairs of dissimilar examples is easy. However, this batch construction is sub-efficient because the number of pairs or triplets increases linearly in the batch size whereas the growth should be quadratic and cubic respectively.

A solution is to sample N different classes (or groups of similar examples) with M examples per class (or group) to build a batch of MN examples. By doing so, the number of pairs of similar examples is $MN(M-1)$ and the number of pairs of dissimilar examples is $M^2N(N-1)$. Interestingly, the number

of dissimilar pairs has a quadratic growth in the batch size and the number of similar examples is over-linear but not quadratic in the batch size. Despite this, this sampling strategy is still better in practice as more pairs are processed during the training. In the case of triplets, we have a similar observation. Indeed, with the same batch construction, we can compute $M^2(M-1)N(N-1)$ triplets in a single batch. The number of triplets does not have a cubic growth but it is still greater than a quadratic growth. By doing such batch construction, we manage to explore more pairs or triplets in a batch compared to the original naive implementation. Despite this improvement, when the training progresses, it still becomes nearly impossible to randomly sample pairs or triplets that generate loss to train the network. Thus, a standard practice is to perform mining of pairs or triplets. A naive way of implementing a mining strategy is, after an epoch, to predict all image representations of the training set. Next, we can compute the loss functions for each query image and build two sets of examples: the first one is the set of similar examples that have small similarities with the query and the second one is the set of dissimilar examples that have high similarities with the query. After, we build the corresponding pairs or triplets that generate loss. During the next epoch, instead of randomly sampling the pairs or triplets, we bias our batch construction by over-sampling these examples. Then, we repeat these steps for each epoch. This mining method is usually referred as *hard example mining* in the case we only sample the similar examples with the lowest similarity and the negative examples with the highest similarity. This can be relaxed by sampling within the top-k similar examples with the smallest similarities and within the top-k dissimilar examples with the highest similarities. We usually refer to this mining strategy as *semi-hard example mining*.

Both the hard and semi-hard example mining strategies tend to improve performances of the network by a large margin, especially for large datasets with a huge number of classes (or group of similar images). However, because these strategies are performed offline and because they have a large computation overload between each epoch, the training is much slower than before even if performances are increased. As a warm-up for the upcoming section 2.4, the search of informative samples can be performed much more efficiently by considering partial or hierarchical offline search [73, 160, 193] or even online example generation [111, 50].

As a conclusion, the training of deep metric learning network relies on the following things: First, we build the batch of examples by sampling pairs or triplets, either randomly using the batch construction composed of N different classes and M images per class or using mining strategies. Second, we extract the deep local features from the images. Third, we compute the global image representations. Fourth, we project these representations into the embedding space. Fifth, we compute the similarity between the images representations, using for example the cosine similarity. Finally, we compute the loss function such as the contrastive loss or the triplet loss and we optimize the weights of the network with gradient back-propagation. In the next sections, we present some recent work in deep metric learning according to the three following points: (1) loss functions (section 2.3), (2) mining strategies (section 2.4) and (3) ensemble methods (section 2.5).

2.3 Loss Functions

In deep metric learning, designing better loss functions is a major research direction to improve the generalization capabilities of the network. From the very first contrastive loss and triplet loss in deep metric learning, researchers have improved, modified, re-designed or simply proposed better loss functions to train our models. In a first time, we focus on the loss functions that enhance the design of the contrastive and the triplet loss [158, 173, 127, 182, 30, 57]. In a second time, we review different loss functions and we shed light on their improvements [175, 126, 143, 20, 183].

2.3.1 Contrastive and triplet losses improvements

Both contrastive loss and triplet loss suffer from a variety of defaults such a zero-gradient due to the max operator, only pairs or triplets consideration, they either only consider a global structure or a local one, relations between classes are not taken into account, *etc.* In consequence, recent work have explore modified version of these two loss functions or even new design to cope with such drawbacks.

Two simultaneous approaches have extended the contrastive loss and the triplet loss by proposing the binomial deviance [173] and the N-pair loss [158]. Their improvements cope with two drawbacks from the original formulations that are the consideration of larger tuples and the zero-gradient problem. Also, their proposed formulations perform batch-wise hard example mining which, in the case of large batch size, tends to slightly improve performances. The first part of the improvement is to use a soft version of the max operator. To do so, they rely on the classic log-sum-exp formulation that approximates, in a differentiable way, the max operator:

$$\max_x x \approx \frac{1}{\alpha} \log \left(\sum_x \mathrm{e}^{\alpha x} \right) \tag{2.15}$$

This smooth maximum is usually modified to be convex by adding $x = 0$ in the set of x values:

$$\max_x x \approx \frac{1}{\alpha} \log \left(1 + \sum_x \mathrm{e}^{\alpha x} \right) \tag{2.16}$$

Using this operator, we can perform a batch-wise hard example mining. Indeed, in the case of the triplet loss, we can consider a pair of similar examples $(\boldsymbol{x}_q, \boldsymbol{x}_p)$ and perform the mining of the hardest negative example by using the differentiable max operator. This modification, in the case of the triplet loss, has been named the N-pair loss [158]:

$$\mathcal{L}_{n-pair}(\boldsymbol{x}_q, \boldsymbol{x}_p, \{\boldsymbol{x}_n\}_n) = \log \left(1 + \sum_{\boldsymbol{x}_n} \mathrm{e}^{\langle \boldsymbol{x}_q \ ; \ \boldsymbol{x}_n \rangle - \langle \boldsymbol{x}_q \ ; \ \boldsymbol{x}_p \rangle} \right) \tag{2.17}$$

In the case of the contrastive loss, the binomial loss [173] exploits the smooth maximum but not the batch-wise hard example mining :

$$\mathcal{L}_b(\boldsymbol{x}, \boldsymbol{y}) = \delta_{\boldsymbol{x},\boldsymbol{y}} \ \log \left(1 + \mathrm{e}^{\beta - \langle \boldsymbol{x} \ ; \ \boldsymbol{y} \rangle} \right) + (1 - \delta_{\boldsymbol{x},\boldsymbol{y}}) \ \log \left(1 + \mathrm{e}^{\alpha(\langle \boldsymbol{x} \ ; \ \boldsymbol{y} \rangle - \beta)} \right) \tag{2.18}$$

where β is a margin and α the coefficient to sharpen the smooth maximum. Lifted Structure [127] also extends the binomial loss by doing batch-wise hard negative example mining with the smooth maximum:

$$\mathcal{L}_{struct}(\boldsymbol{x}_q, \boldsymbol{x}_p, \{\boldsymbol{x}_n\}_n) = \max \left\{ 0; \log \left(\sum_{\boldsymbol{x}_n} \mathrm{e}^{\beta - d(\boldsymbol{x}_q, \boldsymbol{x}_n)} + \mathrm{e}^{\beta - d(\boldsymbol{x}_p, \boldsymbol{x}_n)} \right) + d(\boldsymbol{x}_q, \boldsymbol{x}_p) \right\}^2 \tag{2.19}$$

where β is a margin. The lifted structure can further be improved into the Multi-similarity loss [182] by also considering a batch-wise example mining strategy but for the positive examples:

$$\mathcal{L}_{ms}(\boldsymbol{x}_q, \{\boldsymbol{x}_p\}_p, \{\boldsymbol{x}_n\}_n) = \frac{1}{\alpha} \log \left(1 + \sum_{\boldsymbol{x}_p} \mathrm{e}^{-\alpha(\langle \boldsymbol{x}_q \ ; \ \boldsymbol{x}_q \rangle - \beta)} \right) + \frac{1}{\gamma} \log \left(1 + \sum_{\boldsymbol{x}_n} \mathrm{e}^{\gamma(\langle \boldsymbol{x}_q \ ; \ \boldsymbol{x}_n \rangle - \beta)} \right) \tag{2.20}$$

where α and γ are hyper-parameters to sharpen the smooth maximum and β is a margin. To sum-up, all of these loss functions rely on the idea of replacing the non-smooth gradient of the max operator in the contrastive loss and in the triplet loss. By doing so, they extend the original formulations by considering batch-wise example mining. This design usually slightly improves performances of the deep network. Also, when combined with an offline example mining strategy, these loss functions have a better behavior compared to the triplet loss or the contrastive loss. In practice, they better exploit the mining approach than the original formulations and give higher performances for image retrieval tasks. Following the idea of improving the triplet loss, the quadruplet loss [30] extends it by considering a fourth example:

$$\begin{aligned} \mathcal{L}_q(\boldsymbol{x}_q, \boldsymbol{x}_p, \boldsymbol{x}_n, \boldsymbol{x}_m) = & \max \{0; \langle \boldsymbol{x}_q \ ; \ \boldsymbol{x}_n \rangle - \langle \boldsymbol{x}_q \ ; \ \boldsymbol{x}_p \rangle + \alpha_1\} \\ & + \max \{0; \langle \boldsymbol{x}_m \ ; \ \boldsymbol{x}_n \rangle - \langle \boldsymbol{x}_q \ ; \ \boldsymbol{x}_p \rangle + \alpha_2\} \end{aligned} \tag{2.21}$$

where $\boldsymbol{x}_m$ is the fourth example from a class different from the two others. By minimizing the second part of the equation, we enforce the intra-class distance to be smaller than the inter-class distance. This constraint naturally enhances the local structure formulation of the triplet loss by adding a global structure thanks to the second term of the equation. Another way to add a global information into the triplet loss formulation is to consider a hierarchical margin setting as it is done in HTL [57]. In HTL, the authors have modified the triplet loss by considering an adaptive margin $\alpha_{q,n}$ that is computed as follows:

$$\alpha_{q,n} = \alpha + d_l - d_q \tag{2.22}$$

where α is the fixed margin of the standard triplet loss, d_l is a threshold that depends on the level of the two classes in the tree hierarchy and d_q is the average distance in the class of $\boldsymbol{x}_q$. The second part of the equation takes into account the tree hierarchy: For a tree with L levels, the threshold $d_l = \frac{l(4-d_0)}{L} + d_0$ where d_0 is the average per class of average distances:

$$d_0 = \frac{1}{C}\sum_{c=1}^{C}\left(\frac{1}{n_c^2 - n_c}\sum_{i,j\in c}\|\boldsymbol{x}_i - \boldsymbol{x}_j\|_2^2\right) \tag{2.23}$$

where C is the number of classes in the training set and n_c is the number of examples in the c-th class. The value of l is the smallest value such that the average distance between the two classes is smaller than d_l. The tree hierarchy and values of d_l and d_q are updated at the end of each epoch.

2.3.2 New designs

In addition to the modifications in triplet and contrastive losses, recent contributions also have explored new design and formulation to enhance the generalization capacities of the deep metric learning network [175, 126, 20]. In Angular loss [175], the authors take into account the geometry of the embedding space (*i.e.*, the ℓ_2-normalized vector space) in the design of the loss function. Indeed, the Euclidean distance does not exploit the manifold's structure of the hyper-sphere (For further consideration see Part III). In a nutshell, the angular loss is formulated as follows:

$$\mathcal{L}_{ang}(\boldsymbol{x}_q, \boldsymbol{x}_p, \{\boldsymbol{x}_n\}_n) = \log\left(1 + \sum_{\boldsymbol{x}_n} \mathrm{e}^{4\tan^2\alpha\langle \boldsymbol{x}_a + \boldsymbol{x}_p \,;\, \boldsymbol{x}_n\rangle - 2(1+\tan^2\alpha)\langle \boldsymbol{x}_q \,;\, \boldsymbol{x}_p\rangle}\right) \tag{2.24}$$

where α is a margin expressed as an angle. The idea of the angular loss is to select the triplet that forms a small triangle where the shortest edges link the examples from the same class. By considering angles, the loss function is scale-invariant and rotational-invariant.

From another point of view, the metric learning task can be seen as a clustering task. Thus, a loss function can be designed to optimize the clustering of the representations using, for example, the Facility Location [126]. The idea of the Facility Location is to summarize the whole set of representations $\mathbb{X}$ by a subset of examples $\mathbb{S} \subset \mathbb{X}$:

$$F(\mathbb{X}, \mathbb{S}) = \sum_{\boldsymbol{x}\in\mathbb{X}} \min_{\boldsymbol{s}\in\mathbb{S}} \|\boldsymbol{x} - \boldsymbol{s}\|_2 \tag{2.25}$$

After that, we can compute using an oracle the quality of the clustering and the normalized mutual information. The idea of the loss in [126] is to maximize the clustering quality after the computation of F, to maximize the normalized mutual information that is empirically estimated on the examples and finally to minimize the function F above to find the best set $\mathbb{S}$. However, the finding of $\mathbb{S}$ is an assignment problem, so it is NP-hard. The authors use the algorithm from [101] which is based on sub-modular functions to find $\mathbb{S}$.

Another well-known formulation is the *learning to rank* formulation [113, 24, 198, 166, 121, 20]. In the learning to rank paradigm, the idea is to predict, from a query example and a list of elements the ranking of each element within the list with respect to its similarity to the query. In order to do so, methods like learning to rank [24] cast the problem as learning the probability distribution

of all possible permutation of the list. In this way, the optimization problem is defined as the minimization of the cross-entropy over the distributions of the permutations. A neural network is used to give similarity scores between the query and the list of examples that is then transformed into the probability, and the weights of the network are trained to minimize the cross entropy. FastAP [20] acts in a similar way by producing a differentiable formulation of the average precision. The authors rethink the precision and the recall as probability density functions such that, by quantifying the histogram of the distance, the precision and recall can be re-written as a finite sum of functions. By doing so, a network can be trained end-to-end to directly maximize the average precision metric.

2.4 Sampling Strategies

Even if the loss function plays an important role in the generalization properties of the deep networks, due to the exponential number of pairs, triplets or even tuples in a dataset, these networks are usually harder and harder to optimize while the training progresses. Indeed, the number of informative pairs, triplets or tuples has drastically been reduced and randomly sampling these tuples becomes nearly impossible. Thus, mining examples becomes a necessary step to resume the training and to further improves the performances of the deep networks. Still, as explain in section 2.2, naive mining strategies greatly slow the optimization process because of the computation overload at each epoch to find the relevant tuples. Recent work on mining strategies are usually designed to reduce this computation overload by exploring partial search [73, 122, 160, 193, 82]. Also, other strategies, to avoid the offline computation of these algorithms, have proposed to synthesize examples in order to train the network [111, 201, 50, 202].

2.4.1 Approximate example search

Due to the large computation overload of mining strategies in the entire training dataset, approximate search has become mandatory to reduce this time. To do so, one can rely on approximate nearest neighbor graphs. For example, smart mining [73] exploits the algorithm from [72] to build the graph of nearest neighbor and to perform mining. In the case of hard mining, the computation time is then reduced to $\mathcal{O}(N^2)$ instead of the original $\mathcal{O}(N^3)$ where N is the size of the dataset. For semi-hard mining, the computation is further reduced to $\mathcal{O}(\frac{N^2}{M})$ where M is the number of batch per epoch. This means that the computation time becomes quasi-linear in the size of the batch size which greatly reduces the computation overload of semi-hard mining strategies. DAMLRRM [193] also takes advantage of a graph formulation inspired by the idea of mining on manifold [205]. The authors have explored the use of asymmetric mining: a first batch is built from N classes and M examples per class and a second batch is composed of random examples. By doing so, the first batch enforces consistency in intra-class relationship by sampling informative positive examples from the class manifold. The second batch generates numerous and diverse negative examples to enforce discriminant inter-class relationship. Also, a minimum-cost spanning tree for each class is built in order to ease the sampling of hard positive and hard negative examples on the class manifolds. In a similar way, Iscen et al. [82] have proposed an unsupervised mining on manifold strategy to train deep networks for the unsupervised metric learning task. The authors define two similarity functions: the first one is based on an Euclidean similarity measure (*e.g.*, the cosine similarity) while the second one is a *manifold similarity*. This manifold similarity is computed from the nearest neighbor graph based on the previous cosine similarity. Then, to sample a new example, the idea is to exploit the dissimilarities between the cosine similarity and the manifold similarity. Indeed, an example with a large cosine similarity but with a small manifold distance can be considered as a hard positive example while an example with a small cosine similarity but with a large manifold distance is a negative example. Contrary to supervised hard example mining, in unsupervised hard example mining, also the queries have to be sampled. In order to do so, the authors have proposed to find the fixed points of the nearest neighbor graph by using random walk on graph [103].

From another point of view, one can consider to approximate the embedding space using proxies [122]. In order to do so, a set of M proxies are trained at the same time as the deep network to represent the whole embedding space. Then, instead of doing example mining, an example is

brought closer to is nearest proxy and pushed away from the others. This approach is very efficient as there are few parameters to update (only the proxies) and because it leads to no additional offline computation. Also, a large number of proxies can be used as their only role is to quantify the embedding space without constraints on semantic information (*e.g.*, a fixed number of proxies per class). This semantic information can be added by learning one proxy per class as it is done in [160]. Instead of learning a large set of proxies, the authors only use one proxy per class and, for a given class, examples can be sampled within the top-K nearest classes.

2.4.2 Hard example generation

Finding hard examples is the training set still leads to non-negligible computation overhead. Thus, recent contributions have explored the online generation of examples using GANs [201, 50, 202] or variational auto-encoders [111]. In DVML [111], a variational auto-encoder approach is used to build the image representations, which are supposed to follow a per-class Gaussian distribution. Then, artificial samples are generated by sampling from the Gaussian distribution with the learned parameters. The benefit of DVML is to be able to generate an infinite number of tuples composed by negative and positive artificial examples. However, with DVML, it is impossible to ensure that the generated example is a hard one compared to adversarial approaches [201, 50] as the sampling favors examples generated near the center of the corresponding class. In adversarial approaches such as DAML [50], only one hard negative example is generated with respect to a triplet (potentially a non-informative one) composed of a query, a positive and a negative sample. To do so, the generator is trained to maximize the triplet loss between the fixed query, the fixed positive and the generated hard negative while a ℓ_2 loss minimizes the distance between the real negative and the generated one. A similar idea is developed in HTG [201] where positive samples are also generated. The generator is trained to maximize a triplet loss between a fixed query, a generated positive and a generated negative while the discriminator is trained to correctly classify the generated examples according to their respective classes. Thus, the synthesized samples are designed to be discriminative which increases the margin between the classes. However, these approaches have two important issues: the first one is that can only generate a limited number of examples due to the architecture compared to variational approaches. The second and more important one is that adversarial generators are difficult to tune due to the contrary objectives of the DML network and the adversarial learning of the generator. On the one hand, if the adversarial loss is much lower than the DML loss, the generated examples tend to be at the center of the class manifold and the method faces the same problems as VAE generators. On the other hand, if the adversarial loss is much higher than the DML loss, some examples can be generated beyond the boundary of the class manifolds. So, the mining strategy produces examples with incorrect labels with respect to the training classes.

Zheng *et al.* faced the same issue and proposed HDML [202]. HDML tries to alleviate this effect by generating first an intermediate example, that may be outside its class manifold ; Then a generator re-projects this example into the class manifold. But, no guarantee is made that the re-projection has been done correctly. In case of a failure, they use the DML loss over the real examples to weight the generator loss: if the triplet is an easy one, the generator only slightly modifies the example to avoid the generation of an intermediate example, that would be too far from its class manifold. Moreover, the metric learning loss, that is computed on the generated triplets, is also weighted by the reconstruction loss: if the reconstruction is too bad, they do not take into account the new triplet to compute the DML loss on the generated triplets.

In conclusion, these early work on example generation have raised an interesting research direction as alternative methods for mining strategies. Indeed, the training time has been increased to a not insignificant extent by the adversarial objectives of GANs. Thus, the offline computation of mining strategies has been replaced by the online optimization of a GAN during the training. However, the latter have added new unresolved problems during the training procedure. Indeed, the tricky training of GANs might lead to examples that are generated outside of their class manifolds or simply non-informative examples. Approaches like HDML have partially addressed the problem but the issue still remains. In chapter 6, we propose a new way to generate examples by leveraging prototypes in a similar way to the proxies in [122, 160] but to condition the generation of examples. The generation of examples outside their class manifold is addressed by considering virtual classes that are trained

to be moved between the real classes. By doing so, the generation of example outside their class manifolds fall into the virtual classes: real classes are consequently not harmed and the manifold of the virtual is easily modified to handle such cases during the training.

2.5 Ensemble Methods

Ensemble methods have become an increasingly popular way to improve performances of deep metric learning network. From a finite set of *weak learners* that are trained for the same task, ensemble learning aims at finding the best combination of these weak learners so as to maximize the overall performances for the given task. More formally, we want to find the best weak learners $\{\phi_m\}_{m=1}^M$ and their corresponding weights $\{\alpha_m\}_{m=1}^M$ such that the learner Φ

$$\Phi(\boldsymbol{x}) = \sum_{m=1}^{M} \alpha_m \phi_m(\boldsymbol{x}), \tag{2.26}$$

has a smaller approximation error than than each weak learner. Theoretically, if all weak learners are independent, the variance of the full learner Φ decreases in $\mathcal{O}(\frac{1}{M})$. To do so, a solution is to train each weak learner on a different part of the training set. A well-known algorithm to train such ensemble is AdaBoost [54] which has been widely used for a large range of applications and tasks. In this section, we focus on the ensemble methods that have been applied to enhance deep networks for the deep metric learning tasks [129, 130, 197, 149, 194, 6].

HDC [197] follows this principle by building increasingly deeper network in a multi-scale approach. More precisely, a metric is first computed with a "shallow" network. If this network cannot correctly rank the given triplet, a max pooling layer and multiple convolution layers are added to build a deeper network with higher-level information. At the end, a global representation and an embedding layer are added and, if still this deeper network cannot correctly rank the given triplet, another max pooling layer and other convolution layers are added, and so on. Thus, deeper network with higher-level information are used as weak learners to build the embedding space and each weak learner is trained on harder and harder triplets. To reduce the computation cost, the authors have shared the lower-stage layers for each learner: even if the independence property is not respected, in practice performances are higher with HDC than without.

BIER [129, 130] takes advantage of online gradient boosting [15] to train the weak learners. The m-th learner is trained on a re-weighted training batch according to the negative gradient of the loss function computed on the ensemble based on the $m-1$ weak learners. By doing so, only hard examples are trained on the last weak learners. Similarly to HDC, low-level layers are shared across the weak learners and only the embedding space is divided into non-overlapping groups. The authors have also shown that initializing the weights of the embedding layers to be decorrelated between each weak learners tends to increase performances when using BIER.

In DREML [194], they propose an ensemble method based on bagging: different weak learners are trained on different part of the dataset, then, a voting step is performed. That is, all images passed trough each deep network and the output similarity is computed by averaging the similarity of all weak learners. Each class of the training dataset are merged into *meta-classes*, that is, groups of real training classes. By making each weak learner seeing different meta-classes, the ensemble can learn the global structure while each weak learner is trained on simplified set of classes. Contrary to HDC and BIER, DREML does not share parameters between the deep network which leads to very large models: Typically, their ensemble of 48 GoogleNet [163] with a global embedding size of 9216 leads to more than 2 billion parameters for less than 2% improvement over BIER which has a single GoogleNet network and an embedding of size 512.

EDMS [6] extends DREML by considering manifold similarities as in [205, 82] and proxy-based approximation as in [122]. Contrary to DREML, the meta-classes are not randomly built: they are clustered using proxies to find the optimal groups of training classes. By doing so, the similarity is supposed to be locally Euclidean and the standard loss functions can be used for each part of the training set. However, the global embedding is non-linear, so they exploit the closed form of the

random walk from [205] to compute manifold similarities as in [82] between each part of the training dataset. Thus, a manifold similarity is computed between the examples from a given part of the training set and all proxies while a cosine similarity is computed per part of the training set. During testing, this leads to simply compute the cosine similarity for each weak learner and average the values. Finally, as it is done in DREML, the combination of 25 GoogleNet models [163] with 128 dimensional embedding for each weak learner leads to a very large global model with more than 150 million parameters and a global embedding of size 6144 for 3% improvements.

D&C [149] exploits a divide and conquer approach to build an ensemble method. In a first time, images are passed trough the CNN to get the representation into a first embedding space. This embedding space is then clustered into K groups where each group corresponds to a weak learner. Then, the similarities for each learner are computed on a predefined subspace of the embedding space: the embedding space of size D used in the clustering is divided into K disjoint subspaces of size $\frac{D}{K}$. By doing so, the implementation is fast as only one network is used to compute the representations. Also, the embedding is re-used which makes the clustering easier as the embedding is well conditioned. At testing, the ensemble method is simply based on a bagging approach by concatenating the embedding of each weak learner: similarities are computed using the dot product between these representations.

In conclusion, recent contributions in ensemble methods for deep metric learning has two main approaches: on the one hand, they rely on common ensemble methods based on boosting strategies such as AdaBoost or Bagging but they are thought to be efficiently implemented with lots of parameter re-use. On the other hand, they are not limited in number of parameters and exploit the notion of meta-class to train each weak learners. Also, ensemble methods and mining on manifold approaches can be used together: each weak learner's embedding can be considered locally Euclidean while the whole embedding can be considered as a manifold and the similarity on manifold can be performed to improve performances.

2.6 Datasets and Evaluation Protocol

2.6.1 Datasets

In this section, we present the content-based image retrieval datasets that have been used to evaluate the models in this thesis. Our very first work on image retrieval have been evaluated on the well-known image retrieval datasets that are Oxford5k [136], Paris6k [137] and INRIA Holidays [88]. Oxford5k [136] is a dataset composed of 11 landmarks from the city of Oxford such as the "Oxford Christ Church" and "Oxford Radcliffe Camera". A set of images that contains these landmarks has been retrieved on Flickr. Also, some images that do not contain any of these landmarks have been retrieved in order to be used as distractors for the model. In total, the dataset consists in 5,062 high resolution (1024×768) images. For each landmark, 5 query regions have been selected in order to evaluate the retrieval performances on the dataset. Concerning the evaluation, images have four possible labels for each landmark: (1) good, which means that the object or building is clearly visible, (2) OK, which means that more than 25% of the object/building is visible, (3) junk, which means that less than 25% of the object/building is visible or because there is a huge distortion/occlusion and (4) Absent, which means that the object/building is not present. For a given query landmark, junk images are not taken into account in the computation of the metric. Also, the standard metric reported on this dataset is the mean average precision (see subsection 2.6.2 for details). Paris6k [137] is very similar to Oxford5k except the landmarks are objects/buildings from the city of Paris. This dataset has also 11 landmarks and 5 query regions per landmarks for a total of 6412 high-resolution images. It follows the same labelling approach of Oxford5k but also the same evaluation protocol and metric. Also, some images are corrupted in both Paris6k and Oxford5k, thus, they are never used during the evaluation.

INRIA Holidays [88] is a dataset which mainly contains personal holiday photos. Is is composed of 500 query images for 1491 high-resolution images in total. The dataset has a large variety of objects and scenes (natural, man-made, water, *etc.*). The first image of each group is the query image and

the correct retrieval results are the other images of the group. For the evaluation, the 500 query images are used to rank the elements from the rest of the 1490 images and the scores are average over these 500 query images. Also, the standard metric reported on this dataset is the mean average precision (see subsection 2.6.2 for details). One can note than these three datasets do not contain a training set of images. In order to train our models, we have used a subset of the validation set from the Places365 dataset [204]. Places365 is a scene classification dataset of 365 different scenes. It has more than 1.8 million images in the training set with the image number per class that can vary from 3000 to 5000. The validation set is composed by 50 images per class for a total of 18250 images. The testing set has 900 images per class for a total of 328500 images. The variety of scenes range from indoor scenes (bakery, restaurant, Jacuzzi, *etc.*) to outdoor scenes (mountain, runway, lake, volcano, *etc.*).

To evaluate our later contributions, we rely on the four fine-grained image datasets that are Cub-200-2011 [174], Cars-196 [102], Stanford Online Products [127] and In-Shop Clothes Retrieval [115]. Cub-200-2011 [174] is a fine-grained dataset composed of 200 bird species for 11788 images. Each image is annotated with bounding boxes, part locations, and attribute labels. Historically, this dataset has been first used for fine-grained classification tasks where methods exploit part-based approaches, second-order pooling, and so on. In the case of deep metric learning, the dataset is split into a training set and a testing set. The training set is composed of the first 100 bird species for a total of 5864 images while the testing set is composed of the other 100 bird species (*i.e.*, there is no overlap between the classes from the training and the testing set) for a total of 5924. For evaluation, all images are used as query and the scores are average over them. Also, the standard metrics reported on this dataset are the recall@K with $K \in \{1, 2, 4, 8, 16, 32\}$ and sometimes the normalized mutual information (see subsection 2.6.2 for details). Cars-196 [102] is similar to the birds dataset: it is a fine-grained dataset composed of 196 car models from multiple brands. The first 98 classes of the Cars-196 dataset, for a total of 8054 images, are used for training and the remaining 98 classes, for a total of 8131 images, are used for testing. The evaluation is the same as the birds dataset, including the reported metrics.

Stanford Online Products [127] is a fine-grained dataset composed of 22634 online products (classes) that comes from 11 macro-classes (boilers, microwaves, bikes, *etc.*) from eBay.com for a total of 120053 images. In average, each product has approximately 5.3 images with a minimum of 2 images and a maximum of 11 images. This dataset is divided into two parts: a training set, which is composed of 11318 classes for a total of 59551 images and a testing set which is composed of the rest 11316 classes for a total of 60502 images. For the evaluation, all images from the testing set are used as query to rank the rest of the images and the scores are average over them. Also, the standard metrics reported on this dataset are the recall@K with $K \in \{1, 10, 100, 1000\}$ and sometimes the normalized mutual information (see subsection 2.6.2 for details). In-Shop Clothes Retrieval [115] is an clothes dataset with a large variety of attributes, landmarks, as well as cross-pose correspondences. For the retrieval task, the dataset is composed of 7982 products (classes) for a total of 52712 images. This dataset is divided into three parts: a training set which is composed of 3997 for a total of 25882 images, a query set which is composed of 3985 classes for a total of 14218 images and a collection set which is composed of 3985 classes for a total of 12612 images. For the evaluation, all images from the query set are used to rank all images from the collection set and the scores are average over all queries. Also, the standard metrics reported on this dataset are the recall@K with $K \in \{1, 10, 20, 30, 40, 50\}$ and sometimes the normalized mutual information (see subsection 2.6.2 for details).

2.6.2 Metrics

To evaluate the benefits of state-of-the-art methods in image retrieval, researchers have reported three kinds of metrics, the ones that evaluate if the quality of the ranking, the ones that evaluate the quality of a clustering and the ones that evaluate the quality of the top-k retrieved elements. In the first case, the main metric to evaluate the quality of the ranking, which is used to evaluate datasets like Oxford5k, Paris6k and Holidays, is the mean average precision (mAP). the mAP computes the mean, for each query in the testing set, of the average precision (AP). The AP is defined as the area

under the precision-recall curve:

$$AP(q) = \sum_{n} P(n)\ \Delta R(n), \tag{2.27}$$

where $P(n)$ is the precision computed on the top-n retrieved elements (*i.e.*, the proportion of elements that are relevant with respect to the query among the top-n retrieved elements) and $\Delta R(n)$ is the difference between the recall for the top-n and the top-(n-1) retrieved elements (*i.e.*, 0 if the n-th element is not relevant for the query, $\frac{1}{n_q}$ otherwise where n_q is the number of relevant elements with respect to the query q). Practically, the precision is computed each time a relevant element for the query is retrieved and averaged over all relevant elements. The mean average precision is a difficult metric to optimize in practice as it strongly penalizes irrelevant elements that are ranked into the very first positions. For example, in the case of the average precision computed for 3 relevant elements, if an irrelevant one is ranked at the first position and the relevant ones follows it, the average precision decreases from 1 (all elements are correctly ranked into the three first positions) to $\frac{1}{3}\left(\frac{1}{2} + \frac{2}{3} + \frac{3}{4}\right) \approx 0.64$.

In the second case, the normalized mutual information (NMI) is one of the adopted metrics to evaluate the quality of a clustering. The NMI between a set labels Y and a set of clusters C is computed as follows:

$$NMI(Y;C) = \frac{2\ I(Y;C)}{H(Y) + H(C)}, \tag{2.28}$$

where $I(Y,C)$ is the mutual information between Y and C and H is the entropy. To compute each part of the NMI, the empirical of the mutual information and of the entropy are used:

$$I(Y;C) = H(Y) - H(Y|C) \tag{2.29}$$

$$H(Y) = -\sum_{i=1}^{|Y|} p_i \log_2 p_i \tag{2.30}$$

$$H(C) = -\sum_{j=1}^{|C|} q_j \log_2 q_j \tag{2.31}$$

$$H(Y|C) = -\sum_{j=1}^{|C|} q_j \sum_{i=1}^{|Y|} r_{i,j} \log_2 r_{i,j} \tag{2.32}$$

where p_i is the probability to have the i-th label, q_j is the probability to be in the j-th cluster and $r_{i,j}$ is the probability to have the i-th label and to be in the j-th cluster. The probabilities p_i, q_j and $r_{i,j}$ are estimated on the testing set:

$$p_i = \frac{n_i}{|Y|},\ q_j = \frac{m_j}{|Y|},\ r_{i,j} = \frac{l_{i,j}}{|Y|} \tag{2.33}$$

where n_i is the number of elements with label i, m_j is the number of elements that are in the j-th cluster and $l_{i,j}$ is the number of elements with label i that are in the j-th cluster.

In the last case, the Recall@K, is one of the most used metric to evaluate the top-k retrieved elements. The Recall@K is the average for each query of the presence or not of at least one relevant element in the top-k retrieved elements. That is, Recall@1 is the average number of queries that have their nearest neighbor that is a relevant element and Recall@10 is the average number of queries that have at least 1 relevant element in the 10 best retrieved elements. The term Recall@K is misleading and confusing with the standard denomination of recall. Indeed, the recall is usually used to evaluate clustering or ranking: recall is usually defined as the ratio of the number of relevant elements that have been tagged relevant and the total number of relevant elements. In the rest of this thesis, we will only use Recall@K as the average for each query of the presence or not of at least one relevant element in the top-k retrieved elements.

In conclusion, we follow the standard evaluation procedure to compare our proposed methods to the state-of-the-art ones. In the case of Oxford5k, Paris6k and Holidays, we only report the mAP using

the computation presented in this section and the evaluation protocols details in their respective parts of subsection 2.6.1. For Cub-200-2011, Cars-196, Stanford Online Products and In-Shop Clothes Retrieval, we report the Recall@K with K varying in $\{1, 2, 4, 8, 16, 32\}$ for the two first datasets, in $\{1, 10, 100, 1000\}$ for the third and in $\{1, 10, 20, 30, 40, 50\}$ for the last one.

2.7 Discussion

In this section, we first sum-up the entire chapter before we discuss some of the presented points in light of our contributions. By first formulating the problem of deep metric learning, we show that learning a metric can be seen as an optimization problem that aims at finding the best symmetric positive definite matrix that maximizes the similarity of similar images and that minimizes the similarity of dissimilar images. Then, we present the standard training procedure: the choice of a backbone network to extract deep local features, the computation of image representations, the embedding space, the choice of the loss functions (*e.g.*, the contrastive of the triplet loss) and two batch construction strategies to maximize the number of available pairs or triplets inside the batch. In a second part, we have presented recent contributions in deep metric learning on the three following points: the loss functions, the mining strategies and the ensemble methods. Concerning loss functions, research directions are mostly focused on either improving the design of the standard loss functions by solving some weaknesses or by proposing new loss functions with specific benefits. The research in mining strategies is split into two axes: the first one considers offline approaches to find the informative pairs, triplets or tuples to train the deep network by exploring approximate search, mining on manifolds, *etc.* The second one considers generation-based approaches in order to synthesize hard examples in an online manner, using variational, GANs or geometric approaches. Recent contributions in ensemble methods for deep metric learning tasks have become more and more important due to the large improvement in performances. These contributions extend the bagging and boosting strategies of standard ensemble learning approaches by building weak metric learning networks that are merged together to improve the overall performances. Finally, we have presented the datasets and the metrics that have been used in this thesis in order to evaluate the models.

We next discuss our three main contributions in light of this deep metric learning framework. In a first time, our contributions in tensor-based aggregation (see chapter 1) mostly focus on designing better global image representations that can next be projected into the embedding space before we learn a metric over them. The framework of deep metric learning is one of the most interesting because it directly evaluates the semantic information contained into the representations. Indeed, if the spatial correlation of *ISTA*, the dictionary-based second-order pooling of *JCF* or the implementation of *DIABLO* integrate more semantic information than state-of-the-art methods, performances of these image representations should be higher. Also, using a single linear projection to learn the embedding space has little impact on the image representation, except the rank if this projection also plays the role of dimensionality reduction. In a second time, our contributions in *HORDE* (see chapter 5) are from another recent research direction that does not consider loss functions, mining strategies or ensemble methods. Indeed, the idea is to strengthen and robustify the training procedure of the deep network in the deep metric learning context. For now, there is only few work in this direction. It is worth mentioning the work of Roy et al. [146] on gradient descent over the manifold of positive semi-definite matrices, that is, on the learned Mahalanobis matrix inherited from the embedding. It greatly differs from the standard approaches by exploiting the rotation-invariance property of the embedding during the training procedure. The embedding is directly optimized on this manifold, which makes easier and speeds up the convergence. In a complementary manner, *HORDE* is a regularization method that naturally extends the deep metric learning formulation to robustify the deep local features to occlusion, background variation, *etc.* By doing so, we strengthen the image representation and consequently the embedding space. In a third time, our contributions in *MIRAGE*(see chapter 6) tackle the problem raised in section 2.4 on the example generation methods. In order to do so, we represent each class by a proxy as it is done in [160] but, instead of using these proxies to train the network, we feed them into a conditional generator that synthesize virtual examples from the same class. Contrary to other generation-based methods, the presence of virtual classes greatly reduces the impact of examples that are generated outside their class manifold. Indeed, even if a generated exampled is synthesized outside its class manifold, it falls into a virtual

class which does not harm the rest of the embedding space. In the same way, if an example generated from a virtual class falls into the manifold of real class, it is easier to modify the shape of the virtual class manifold instead of the shape of the real class manifold. Last but not least, we have made the choice to not report the NMI metric on the dataset. Indeed, all of our contributions are not based on clustering methods which means that the NMI mostly evaluates our implementation of K-means instead of the embedding itself. In fact in our early tests, a K-means implementation with splitting and cluster re-allocation strategies gives nearly 5% more NMI than the naive K-means algorithm.

3

Second-Order Pooling

In this chapter, we present all the materials related to second-order pooling, from the formulation to the recent research trends.

Contents

3.1 Introduction **30**
3.2 Formulation **30**
3.3 Normalization **33**
3.4 Factorization **38**
3.5 Beyond Second-Order Pooling **43**
3.6 Discussion **45**

3.1 Introduction

Also known as covariance pooling or bilinear pooling in the case of multi-modal representations, second-order pooling is a representation learning strategy that considers the second-order statistics. In the computer vision community, higher-order statistics have become more and more used for a wide variety of tasks, such as fine-grained classification [56, 104, 110, 186], face recognition [34], pedestrian detection [171], person re-id [172], image super-resolution [38], action recognition [200], semantic segmentation [25], style and content separation [167], among others. Especially, most of recent contributions are either in second-order pooling for fine-grained image classification [104, 186] or in bilinear pooling for cross-modal approaches such as visual question answering [13, 14, 97].

When applied on top of a deep network, training in an end-to-end manner both the network parameters and the second-order representation has been shown to outperform standard approaches for fine-grained image analysis. Also, the extension of second-order to bilinear pooling, that is, by considering two modalities, is a natural way to solve approaches like visual question answering. However, using second-order statistics raises two problems. The first one is the dimensionality of the representation. Indeed, if we consider 1k dimensional features with a 1k classification task (*e.g.*, ImageNet [45]), the covariance matrix is a 1M dimensions and the linear classifier requires 1 billion parameters which is intractable during the training. Also, these representations are greatly subjected to the curse of dimensionality, and thus over-fitting. As a consequence, factorization methods have been explored to reduce the dimensionality of the representation while keeping performances of second-order statistics. The second one is the structure of the manifold. Indeed, covariance matrices are symmetric positive definite (SPD) matrices and the manifold of SPD matrices is known to be a Riemannian manifold, that is, a manifold for which the Euclidean distance is not suited. As such, geodesic distances on the SPD manifold (also referred as Riemannian metrics) [4, 132] have been proposed to tackle this issue. For instance, normalization of the SPD matrices is a widely adopted approach: by normalizing the SPD matrices, the SPD manifold is non-linearly transformed into an Euclidean-friendly space where standard Euclidean distance truly represent the structure of the data.

In this chapter, we give an overview of second-order representation learning. First, we formulate the representation and review the most standard approaches to compute these representations and to train them in the particular case of second-order pooling for classification. Then, we generalize this formulation to encompass bilinear pooling and covariance pooling approaches. Second, we review the recent contributions in second-order pooling according to the dedicated dimensionality reduction for the SPD matrices but also the geodesic distance and their related normalization methods. Finally, we review some methods that go beyond the second-order statistics, by exploring high-order pooling, Mixture pooling, Grassmann pooling, and so on.

3.2 Formulation

Second-order pooling aims at using the second-order moment (or the covariance matrix) of deep local features as a global representation. For a set of deep local features $\{\boldsymbol{x}_i \in \mathbb{R}^c\}_{i=1}^N$, we note $\boldsymbol{X} \in \mathbb{R}^{c \times N}$ the matrix of the deep features where the i-th column of $\boldsymbol{X}$ is $\boldsymbol{x}_i$. The second-order pooling of the deep local features, noted as $\boldsymbol{\Sigma} \in \mathbb{R}^{c \times c}$ is computed as follows:

$$\boldsymbol{\Sigma} = \frac{1}{N} \sum_{i=1}^{N} \boldsymbol{x}_i \boldsymbol{x}_i^\top = \frac{1}{N} \boldsymbol{X} \boldsymbol{X}^\top \tag{3.1}$$

In practice, to avoid computation issue due to null eigenvalues, a small positive value is sometimes added on the diagonal terms of the second-order moment. This is necessary in the case of the covariance matrix is low-rank to use some geodesic distances or normalization strategies. Also, even if this empirical estimator is biased, it does not harm the representation as it is usually ℓ_2-normalized.

The extension from second-order pooling to covariance pooling is simple as the second-order moment is only required to be re-normalized:

$$\widetilde{\boldsymbol{\Sigma}} = \frac{1}{N}\boldsymbol{X}\boldsymbol{X}^\top - \frac{1}{N^2}\boldsymbol{X}\mathbf{1}_N\mathbf{1}_N^\top\boldsymbol{X}^\top \tag{3.2}$$

In the case of bilinear pooling, we consider two sets of deep local features $\{\boldsymbol{x}_i \in \mathbb{R}^{c_1}\}_{i=1}^N$ represented by $\boldsymbol{X} \in \mathbb{R}^{c_1 \times N}$ and $\{\boldsymbol{y}_i \in \mathbb{R}^{c_2}\}_{i=1}^N$ represented by $\boldsymbol{Y} \in \mathbb{R}^{c_2 \times N}$. Bilinear pooling refers to the "cross-second-order moment" between the two sets of features:

$$\boldsymbol{\Sigma} = \frac{1}{N}\sum_{i=1}^{N}\boldsymbol{x}_i\boldsymbol{y}_i^\top = \frac{1}{N}\boldsymbol{X}\boldsymbol{Y}^\top \tag{3.3}$$

This representation can also be re-normalized to avoid computation issue as it is done with the second-order pooling. Also, the outer product between the means can be subtracted to produce the cross-covariance pooling:

$$\widetilde{\boldsymbol{\Sigma}} = \frac{1}{N}\boldsymbol{X}\boldsymbol{Y}^\top - \frac{1}{N^2}\boldsymbol{X}\mathbf{1}_N\mathbf{1}_N^\top\boldsymbol{Y}^\top \tag{3.4}$$

Another way to express these second-order representations is to use multi-linear algebra, also known as tensor algebra. In multi-linear algebra, we consider new mathematical objects that are *tensors*. Compared to vectors and matrices, vectors in $\mathbb{R}^n$ can be seen as first-order tensors (one dimension of size n) and matrices in $\mathbb{R}^{n\times m}$ can be seen as second-order tensors (two dimensions of size n and m respectively). Tensors are higher-order generalization of vectors and matrices. For example, a third-order tensor is an element from $\mathbb{R}^{n\times m\times p}$ with three dimensions of size n, m and p respectively, and so on. Concerning indexing of tensors, if we keep on with our third-order tensor, this tensor $\boldsymbol{\mathcal{T}} \in \mathbb{R}^{n\times m\times p}$ is indexed similarly to matrices: $\boldsymbol{\mathcal{T}}_{i,j,k}$ is the real value of tensor $\boldsymbol{\mathcal{T}}$ at position $1 \leq i \leq n, 1 \leq j \leq m$ and $1 \leq k \leq p$. In the rest of this thesis, we mostly rely on three tensor operations that are the Kronecker product ($\otimes$), the i-th modular product ($\times_i$) and the extension of the Frobenius dot product for matrices ($\langle \cdot\ ;\ \cdot \rangle_{\mathcal{F}}$). In the case of the Kronecker product, we consider two vectors $\boldsymbol{x} \in \mathbb{R}^n$ and $\boldsymbol{y} \in \mathbb{R}^m$ such that the second-order tensor $\boldsymbol{\mathcal{T}} \in \mathbb{R}^{n\times m}$ is computed as follows:

$$\boldsymbol{\mathcal{T}} = \boldsymbol{x} \otimes \boldsymbol{y} = \boldsymbol{x}\boldsymbol{y}^\top \tag{3.5}$$

In the case of two vectors, it is directly the outer product. This is easily generalized to higher-order tensors, *e.g.*, a third-order tensor $\boldsymbol{\mathcal{T}} \in \mathbb{R}^{n\times m\times p}$ is obtained from the Kronecker product between the three vectors $\boldsymbol{x} \in \mathbb{R}^n, \boldsymbol{y} \in \mathbb{R}^m$ and $\boldsymbol{z} \in \mathbb{R}^p$ as follows:

$$\boldsymbol{\mathcal{T}} = \boldsymbol{x} \otimes \boldsymbol{y} \otimes \boldsymbol{z} \tag{3.6}$$

such that $\boldsymbol{\mathcal{T}}_{i,j,k} = x_i y_j z_k$. In the case of the i-th modular product, it is similar to matrix-matrix and matrix-vector multiplications: For a tensor $\boldsymbol{\mathcal{T}} \in \mathbb{R}^{n\times m}$ and two vectors $\boldsymbol{x} \in \mathbb{R}^n$ and $\boldsymbol{y} \in \mathbb{R}^m$, the modular products are defined as follows:

$$\boldsymbol{\mathcal{T}} \times_1 \boldsymbol{x} = \boldsymbol{\mathcal{T}}\boldsymbol{x} \in \mathbb{R}^m \tag{3.7}$$

$$\boldsymbol{\mathcal{T}} \times_2 \boldsymbol{y} = \boldsymbol{\mathcal{T}}^\top\boldsymbol{y} \in \mathbb{R}^n \tag{3.8}$$

where $\cdot^\top$ is the transpose operator in the matrix sense. If we keep our example with a third-order tensor $\boldsymbol{\mathcal{T}} \in \mathbb{R}^{n\times m\times p}$ and the vectors $\boldsymbol{x} \in \mathbb{R}^n, \boldsymbol{y} \in \mathbb{R}^m$ and $\boldsymbol{z} \in \mathbb{R}^p$, the modular products are computed as follows:

$$\boldsymbol{\mathcal{T}}_{1_{j,k}} = (\boldsymbol{\mathcal{T}} \times_1 \boldsymbol{x})_{j,k} = \sum_s \boldsymbol{\mathcal{T}}_{s,j,k}\ x_s \tag{3.9}$$

$$\boldsymbol{\mathcal{T}}_{2_{i,k}} = (\boldsymbol{\mathcal{T}} \times_2 \boldsymbol{y})_{i,k} = \sum_u \boldsymbol{\mathcal{T}}_{i,u,k}\ y_u \tag{3.10}$$

$$\boldsymbol{\mathcal{T}}_{3_{i,j}} = (\boldsymbol{\mathcal{T}} \times_3 \boldsymbol{z})_{i,j} = \sum_v \boldsymbol{\mathcal{T}}_{i,j,v}\ z_v \tag{3.11}$$

with $\boldsymbol{\mathcal{T}}_1 \in \mathbb{R}^{m \times p}, \boldsymbol{\mathcal{T}}_2 \in \mathbb{R}^{n \times p}$ and $\boldsymbol{\mathcal{T}}_3 \in \mathbb{R}^{n \times m}$. To extend this presentation, if we consider a matrix $\boldsymbol{U} \in \mathbb{R}^{n \times n'}$, the first modular product between $\boldsymbol{\mathcal{T}}$ and $\boldsymbol{U}$ gives the tensor $\boldsymbol{\mathcal{T}}' \in \mathbb{R}^{n' \times m \times p}$ which is computed as follows:

$$\boldsymbol{\mathcal{T}}'_{i,j,k} = (\boldsymbol{\mathcal{T}} \times_1 \boldsymbol{U})_{i,j,k} = \sum_s \boldsymbol{\mathcal{T}}_{s,j,k} \, U_{s,i} \tag{3.12}$$

In the case of the Frobenius dot product, let us first consider two matrices $\boldsymbol{A}$ and $\boldsymbol{B} \in \mathbb{R}^{n \times n}$. The Frobenius dot product between the two matrices $\boldsymbol{A}$ and $\boldsymbol{B}$ is defined as follows:

$$\begin{aligned} \langle \boldsymbol{A} \; ; \; \boldsymbol{B} \rangle_{\mathcal{F}} &= \operatorname{Tr}\left(\boldsymbol{A}^\top \boldsymbol{B}\right) && (3.13) \\ &= \sum_{i,j} A_{i,j} B_{i,j} && (3.14) \\ &= \langle \operatorname{vec}(\boldsymbol{A}) \; ; \; \operatorname{vec}(\boldsymbol{B}) \rangle && (3.15) \end{aligned}$$

This is easily generalized to tensors: *E.g.*, for third-order tensors, The Frobenius dot product between two tensors $\boldsymbol{\mathcal{T}}, \boldsymbol{\mathcal{S}} \in \mathbb{R}^{n \times m \times p}$ is computed as follows:

$$\langle \boldsymbol{\mathcal{T}} \; ; \; \boldsymbol{\mathcal{S}} \rangle_{\mathcal{F}} = \sum_{i,j,k} \boldsymbol{\mathcal{T}}_{i,j,k} \boldsymbol{\mathcal{S}}_{i,j,k} \tag{3.16}$$

and the extension of the Frobenius norm $\|\cdot\|_{\mathcal{F}}$ follows in the same way. Also, if there is no ambiguity, the index $\mathcal{F}$ is usually dropped. This operation is mostly used in section 3.4 and chapter 4 in the case of dimensionality reduction: we want to find the optimal set of tensors $\{\boldsymbol{\mathcal{S}}_i\}_{i=1}^D$ such that $\sum_i \langle \boldsymbol{\mathcal{S}}_i \; ; \; \boldsymbol{\mathcal{T}} \rangle_{\mathcal{F}} \langle \boldsymbol{\mathcal{S}}_i \; ; \; \boldsymbol{\mathcal{U}} \rangle_{\mathcal{F}} \approx \langle \boldsymbol{\mathcal{T}} \; ; \; \boldsymbol{\mathcal{U}} \rangle_{\mathcal{F}}$. This means that the dot product between large tensors has to be approximated using vectors of size D, obtained from linear projections using the set of tensors $\{\boldsymbol{\mathcal{S}}_i\}_{i=1}^D$.

To illustrate this introduction to tensor algebra, we consider a tensor-based approach in visual question answering. Visual question answering is usually cast as a classification problem: we aim at predicting the correct answer among a total of d_a answers from the question text embedding $\boldsymbol{v}_q \in \mathbb{R}^{d_q}$ and the image representation $\boldsymbol{v}_v \in \mathbb{R}^{d_v}$. To do so, we build a third-order tensor $\boldsymbol{\mathcal{T}} \in \mathbb{R}^{d_a \times d_q \times d_v}$ that is trained end-to-end, such that:

$$\boldsymbol{v}_a = \boldsymbol{\mathcal{T}} \times_q \boldsymbol{v}_q \times_v \boldsymbol{v}_v \tag{3.17}$$

where $\times_q$ is the modular product for the modality related to the question and $\times_v$ is the modular product for the modality related to the image. Followed by a soft-max operation, this gives the probability of the answer most likely to be correct. By denoting $\boldsymbol{\mathcal{T}}_i \in \mathbb{R}^{d_q \times d_v}$ the i-th slice of the tensor $\boldsymbol{\mathcal{T}}$ along the first-order, the i-th logit of $\boldsymbol{v}_a$ is obtained using the Frobenius dot product and the Kronecker product:

$$v_{a_i} = \langle \boldsymbol{\mathcal{T}}_i \; ; \; \boldsymbol{v}_q \otimes \boldsymbol{v}_v \rangle_{\mathcal{F}} \tag{3.18}$$

However, this naive formulation is rarely used in visual question answering. Indeed, the standard implementations (taken from [13]) usually extracts visual features with a ResNet152 [74] which has $d_v = 2048$, the question embedding with a GRU [32] which has $d_q = 2000$ dimensions in order to predict $d_a = 2000$ answers. This leads to train a tensor $\boldsymbol{\mathcal{T}}$ with more than 8 billion parameters. Thus, more than having issues to store the network, it is also subjected to over-fitting due to the curse of dimensionality. Indeed, each answer's logit is obtained by a linear projection between the second-order representation $\boldsymbol{v}_q \otimes \boldsymbol{v}_v$ and $\boldsymbol{\mathcal{T}}_i$ which have more than 4 million dimensions. Also, the number of computation has a cubic growth in $\mathcal{O}(d_a d_q d_v)$ which might be intractable by using larger visual and textual representations or a higher number of pre-selected answers. As a warm-up for the upcoming section 3.4, factorization strategies of the tensor $\boldsymbol{\mathcal{T}}$ have been proposed to cope with the number of parameters and the computation cost.

Returning to the second-order pooling, it can be computed using the Kronecker product as follows:

$$\boldsymbol{\Sigma} = \frac{1}{N} \sum_{i=1}^{N} \boldsymbol{x}_i \otimes \boldsymbol{x}_i \tag{3.19}$$

It follows that a linear projection of $\boldsymbol{\Sigma}$ can be performed using the Frobenius dot product and a tensor $\boldsymbol{\mathcal{W}} \in \mathbb{R}^{c \times c}$:

$$\langle \boldsymbol{\mathcal{W}} \; ; \; \boldsymbol{\Sigma} \rangle_{\mathcal{F}} = \frac{1}{N} \sum_i \langle \boldsymbol{\mathcal{W}} \; ; \; \boldsymbol{x}_i \otimes \boldsymbol{x}_i \rangle_{\mathcal{F}} \tag{3.20}$$

$$= \frac{1}{N} \sum_i \sum_{k,l} \boldsymbol{\mathcal{W}}_{k,l} \boldsymbol{x}_{i_k} \boldsymbol{x}_{i_l} \tag{3.21}$$

$$= \frac{1}{N} \sum_i \boldsymbol{x}_i^\top \boldsymbol{\mathcal{W}} \boldsymbol{x}_i \tag{3.22}$$

where the last line is expressed with standard matrix operations.

After the computation of the representation, the question about how to train it still remains. In this case, we focus on the training procedure for fine-grained classification [110, 56]. The usual steps to train the network are: (1) extraction of deep local features using a pre-trained deep network, (2) computation of the second-order representation, (3) normalization of the representation if required, (4) signed-square rooting (SSR) [133] and ℓ_2-normalization, (5) classification using a fully-connected layer with soft-max activation and cross-entropy loss. The weight optimization is usually performed with stochastic gradient descent with momentum, gradient clipping and weight decay. The signed-square root normalization is inherited from the VLAD representation [133, 2] and tends to slightly improve the classification accuracy. Usually, the backbone network used in fine-grained image classification is VGG-16 [154] which has 512 dimensional features at the last convolution layer. This leads to 262k dimensional representation with the second-order pooling that are fed into the classifier after the SSR and the ℓ_2-norm. Similarly to visual question answering, the quadratic growth of the dimensionality in the second-order pooling limits the use of stronger backbone network such as ResNet152 [74] or Inception-V3 [164]. Thus, approaches that benefits from stronger backbone networks rely on factorization schemes such as the ones from section 3.4.

As a conclusion, we have shown the standard formulation of bilinear and second-order as well as the whole pipeline to extract the representations and to train them for fine-grained image classification. Also, we have introduced the tensor algebra and the standard computation that will widely be used in the next chapters. In the next sections, we present some of the recent work in bilinear pooling on the two following points: (1) normalization strategies in order to exploit the geometry of the SPD manifold (section 3.3) and (2) factorization methods to reduce the dimensionality of the representations (section 3.4).

3.3 Normalization

In this section, we first present a gentle introduction to the geometry of the SPD matrix manifold and the corresponding metrics specifically designed to such manifold. Then, we present the link between these distances and the normalization strategies in order to better handle the geometry of the SPD manifold with the Euclidean distance. Finally, we present some state-of-the-art methods to compute in an efficient way these normalizations.

The manifold of SPD matrices is clearly not a vector space. Indeed, the space is not closed under scalar product: *E.g.*, multiplying a SPD matrix with a negative scalar makes it negative definite. Specifically, the SPD manifold is known to be a Riemannian manifold which means that geodesic distances have to be considered to take the geometry of the space into account. As such, using the Euclidean distance (*e.g.*, with the Frobenius dot product from section 3.2) does not respect the geometry of the space, and this phenomenon has been illustrated in lots of computer vision tasks, *e.g.* [171, 132] where poor results have been obtained with the Euclidean distance. In the case of the SPD manifold, three important geodesic distances have been shown to respect the geometry of the manifold: they are the affine-invariant Riemannian metric [132], the Stein divergence (also known as Jensen-Bregman LogDet divergence [31]) and the Jeffrey divergence (also known as symmetric Kullback-Leibler divergence). For two SPD matrices $\boldsymbol{X}$ and $\boldsymbol{Y}$ in $\mathbb{R}^{n \times n}$, the affine-invariant

Riemannian metric d_R [132] is computed as follows:

$$d_R(\boldsymbol{X}, \boldsymbol{Y}) = \left\| \log\left(\sqrt{\boldsymbol{X}}\boldsymbol{Y}\sqrt{\boldsymbol{X}}\right) \right\|_{\mathcal{F}} \tag{3.23}$$

where $\sqrt{\boldsymbol{X}}$ is the square-root of the SPD matrix $\boldsymbol{X}$ such that $\sqrt{\boldsymbol{X}} = \boldsymbol{U}\sqrt{\boldsymbol{\Delta}}\boldsymbol{U}^\top$ with $\boldsymbol{X} = \boldsymbol{U}\boldsymbol{\Delta}\boldsymbol{U}^\top$ the eigen-decomposition of $\boldsymbol{X}$. Because $\boldsymbol{\Delta}$ is diagonal, $\sqrt{\boldsymbol{\Delta}}$ is computed by element-wise square-rooting. Similarly, the computation of $\log(\boldsymbol{X})$ is performed using the eigen-decomposition: $\log(\boldsymbol{X}) = \boldsymbol{U}\log(\boldsymbol{\Delta})\boldsymbol{U}^\top$ where $\log(\boldsymbol{\Delta})$ is the natural logarithm applied element-wise. The Stein divergence d_S [31] is a particular case of Jensen-Bregman divergence computed as follows:

$$d_S^2(\boldsymbol{X}, \boldsymbol{Y}) = \log\det\left(\frac{\boldsymbol{X}+\boldsymbol{Y}}{2}\right) - \log\det(\boldsymbol{X}\boldsymbol{Y}) \tag{3.24}$$

Finally, the Jeffrey divergence d_J is computed as follows:

$$d_J^2(\boldsymbol{X}, \boldsymbol{Y}) = \frac{1}{2}\operatorname{Tr}\left(\boldsymbol{X}^{-1}\boldsymbol{Y}\right) + \frac{1}{2}\operatorname{Tr}\left(\boldsymbol{Y}^{-1}\boldsymbol{X}\right) - n \tag{3.25}$$

Among the interesting properties of these metrics, one of the most important one is that they are invariant by affine transformation. That is, for an invertible matrix $\boldsymbol{W} \in \mathbb{R}^{n\times n}$, we have:

$$\begin{aligned} d_R(\boldsymbol{X}, \boldsymbol{Y}) &= d_R(\boldsymbol{W}\boldsymbol{X}\boldsymbol{W}^\top, \boldsymbol{W}\boldsymbol{Y}\boldsymbol{W}^\top) \\ d_S(\boldsymbol{X}, \boldsymbol{Y}) &= d_S(\boldsymbol{W}\boldsymbol{X}\boldsymbol{W}^\top, \boldsymbol{W}\boldsymbol{Y}\boldsymbol{W}^\top) \\ d_J(\boldsymbol{X}, \boldsymbol{Y}) &= d_J(\boldsymbol{W}\boldsymbol{X}\boldsymbol{W}^\top, \boldsymbol{W}\boldsymbol{Y}\boldsymbol{W}^\top) \end{aligned} \tag{3.26}$$

Note that this property will be exploited for dimensionality reduction in section 3.4. Also, it has been proven that all of these geodesic distances are equivalent, in the sense that the length of the curve between two points from the manifold are the same up to a scaling factor [69, 68]. This means that they all perfectly represent the geometry of the SPD manifold, and choosing one among of them is mostly a consideration of computation cost and numerical stability.

However, all of these geodesic distances share a common drawback: they are computationally unattractive. Indeed, in the case of the affine-invariant Riemannian metric, we need to compute two eigen-decompositions: one to compute the square-root of $\boldsymbol{X}$ and another one to compute the log of the product. In the case of the Stein divergence, we need to compute the determinant of two matrices. Finally, in the case of the Jeffrey divergence, we need to compute the inverse of $\boldsymbol{X}$ and $\boldsymbol{Y}$. Also, the Jeffrey divergence might be subjected to numerical instability in the inversion if the matrices are ill-conditioned. Furthermore, there is no linear separability between $\boldsymbol{X}$ and $\boldsymbol{Y}$, that is, the computation of matrix logarithm, matrix-square-root or determinant are necessary online. Consequently, image retrieval tasks as illustrated in chapter 2 cannot take advantage of these geodesic distances because of the large computation overload of eigen-decomposition, determinant and matrix inversion that are needed to be performed online.

Among the other metrics that have been proposed to replace the affine-invariant Riemannian metric is the log-Euclidean Riemannian metric d_{LE} [3] which is computed as follows:

$$d_{LE}(\boldsymbol{X}, \boldsymbol{Y}) = \|\log(\boldsymbol{X}) - \log(\boldsymbol{Y})\|_{\mathcal{F}} \tag{3.27}$$

This metric maps the SPD manifold to a flat Riemannian space in order to make the Euclidean distance a geometry-preserving distance in the logarithmic domain. This new distance is much more efficient in practice: For example in metric learning, one can compute the matrix logarithm on each covariance matrix offline and store the results. The search by similarity is simply and efficiently performed with the Euclidean distance. Furthermore, the log-Euclidean Riemannian metric has the affine-invariance property. This metric is widely adopted by the computer vision community because the similarity measure is simple: we compute offline the matrix logarithm of the second-order representation and the similarity measure is the standard Euclidean distance which is much faster than computing one of the metrics above. However, the log-Euclidean Riemannian metric is numerically unstable due to the log operation. Indeed, for ill-conditioned SPD matrices, the log has been shown to perform poorly [80, 105]. This is especially true in deep learning where covariance

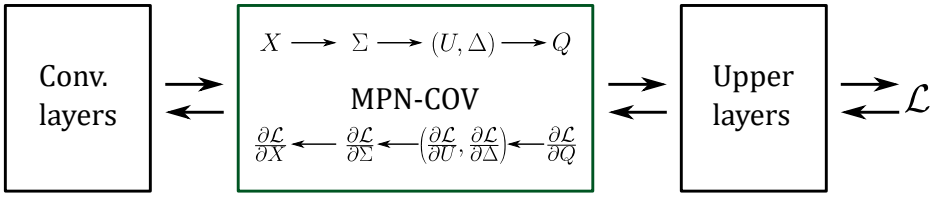


Figure 3.1: Matrix back-propagation overall scheme borrowed from Li et al. [105]. Deep features $\boldsymbol{X} \in \mathbb{R}^{c \times N}$ are extracted using a backbone network. They are followed by the second-order pooling $\boldsymbol{\Sigma} \in \mathbb{R}^{c \times c}$ and an eigen-decomposition $(\boldsymbol{U}, \boldsymbol{\Delta})$ in order to compute either the log-norm or the power-norm in $\boldsymbol{Q}$.

matrices are high-dimensional but estimated from few sparse features due to ReLU activation. To cope with this drawback, pow-Euclidean metrics [49] have been proposed as an alternative to the log-Euclidean. For a given $\alpha \neq 0$, the pow-Euclidean metric d_{p_α} is computed as follows:

$$d_{p_\alpha}(\boldsymbol{X}, \boldsymbol{Y}) = \frac{1}{\alpha} \left\| \boldsymbol{X}^\alpha - \boldsymbol{Y}^\alpha \right\|_{\mathcal{F}} \tag{3.28}$$

Similarly to the log-Euclidean, the fact that $\boldsymbol{X}$ and $\boldsymbol{Y}$ are separable is greatly interesting as the normalization can also be performed offline, while the comparison is performed online. However, contrary to the log-Euclidean, the pow-Euclidean is not a geodesic distance. Still, the pow-Euclidean distance "flattens" the SPD manifold such that almost everywhere the Euclidean distance becomes a geodesic distance after the power-normalization. Moreover, the pow-Euclidean metric is numerically stable compared to the log-Euclidean and it can be used on symmetric semi-definite positive matrices, that is, SPD matrices with low-rank. Interestingly, on can note that the limit of the pow-Euclidean metric for $\alpha \to 0$ leads to $d_{p_\alpha} \to d_{LE}$ [105]. Finally, $\alpha = \frac{1}{2}$ has been shown in [105] to be the optimal solution of the consistent regularized maximum likelihood estimation where the regularization is the von Neumann divergence [180], that is, with a geometry-aware regularizer for the SPD manifold.

Consequently, the use of the log-Euclidean metric or the pow-Euclidean metrics are preferred over the standard Euclidean distance. This shows the benefits of normalization in second-order pooling and bilinear pooling in order to exploit appropriate distances. Thus, recent contributions in normalization are mostly focused on improving the computation of either the matrix logarithm or the matrix square-root. Research directions include fast approximation of these approaches, improvement of the numerical stability or enhancement of the conditioning of the covariance matrix.

For example, the computation of matrix logarithm or matrix square-root requires the eigen decomposition of covariance matrices, which is not differentiable as it is. Thus, Ionescu et al. [80, 81] have proposed a matrix back-propagation method in order to have a differentiable formulation of the singular value decomposition and the eigen-decomposition. To do so, let us consider a set of features written with the matrix notation $\boldsymbol{X} \in \mathbb{R}^{c \times N}$. The full forward/backward computation scheme is summarized in Figure 3.1. The forward pass is composed of the following steps: (1) computation of the second-order representation $\boldsymbol{\Sigma} \in \mathbb{R}^{c \times c}$, (2) eigen-decomposition $\boldsymbol{\Sigma} = \boldsymbol{U} \boldsymbol{\Delta} \boldsymbol{U}^\top$, (3) normalization such that $\boldsymbol{Q} = \boldsymbol{U} f(\boldsymbol{\Delta}) \boldsymbol{U}^\top$ where f is either the log-normalization or the power-normalization. Lets suppose that we already know $\frac{\partial \mathcal{L}}{\partial \boldsymbol{Q}}$. Using the chain rule, we have the following equality:

$$\operatorname{Tr}\left(\frac{\partial \mathcal{L}}{\partial \boldsymbol{U}}^\top d\boldsymbol{U} + \frac{\partial \mathcal{L}}{\partial \boldsymbol{\Delta}}^\top d\boldsymbol{\Delta} \right) = \operatorname{Tr}\left(\frac{\partial \mathcal{L}}{\partial \boldsymbol{Q}}^\top d\boldsymbol{Q} \right) \tag{3.29}$$

where $d\boldsymbol{Q}$ denotes the variation on the matrix $\boldsymbol{Q}$. From the definition of $\boldsymbol{Q}$, we have $d\boldsymbol{Q} = d\boldsymbol{U} df(\boldsymbol{\Delta}) d\boldsymbol{U}$ where $df(\boldsymbol{\Delta})$ is computed element-wise because $\boldsymbol{\Delta}$ is diagonal. For example, the α power-norm leads to $df(\boldsymbol{\Delta}) = \alpha \boldsymbol{\Delta}^{\alpha-1} d\boldsymbol{\Delta}$. For generality, let us denote $df(\boldsymbol{\Delta}) = f'(\boldsymbol{\Delta}) d\boldsymbol{\Delta}$. Thus, the partial derivative for $\boldsymbol{U}$ and $\boldsymbol{\Delta}$ are ([105], Equation 7):

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{U}} = \left(\frac{\partial \mathcal{L}}{\partial \boldsymbol{Q}} + \frac{\partial \mathcal{L}}{\partial \boldsymbol{Q}}^\top \right) \boldsymbol{U} f(\boldsymbol{\Delta}) \tag{3.30}$$

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{\Delta}} = \left(f'(\boldsymbol{\Delta}) \boldsymbol{U}^\top \frac{\partial \mathcal{L}}{\partial \boldsymbol{Q}} \boldsymbol{U} \right)_{\text{diag}} \tag{3.31}$$

where $\boldsymbol{A}_{\text{diag}}$ is the matrix $\boldsymbol{A}$ with all off-diagonal elements set to 0. Continuing with the chain rule, knowing $\frac{\partial \mathcal{L}}{\partial \boldsymbol{U}}$ and $\frac{\partial \mathcal{L}}{\partial \boldsymbol{\Delta}}$, $\frac{\partial \mathcal{L}}{\partial \boldsymbol{\Sigma}}$ respects the following constraint:

$$\text{Tr}\left(\frac{\partial \mathcal{L}}{\partial \boldsymbol{\Sigma}}^{\top} d\boldsymbol{\Sigma}\right) = \text{Tr}\left(\frac{\partial \mathcal{L}}{\partial \boldsymbol{U}}^{\top} d\boldsymbol{U} + \frac{\partial \mathcal{L}}{\partial \boldsymbol{\Delta}}^{\top} d\boldsymbol{\Delta}\right) \tag{3.32}$$

After some arrangements, $\frac{\partial \mathcal{L}}{\partial \boldsymbol{\Sigma}}$ is computed with Equation 10 from [105]:

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{\Sigma}} = \boldsymbol{U}\left(\boldsymbol{K}^{\top} \odot \left(\boldsymbol{U}^{\top}\frac{\partial \mathcal{L}}{\partial \boldsymbol{U}}\right) + \left(\frac{\partial \mathcal{L}}{\partial \boldsymbol{\Delta}}\right)_{\text{diag}}\right)\boldsymbol{U}^{\top} \tag{3.33}$$

where $\odot$ is the element-wise multiplication (or Hadamard product) and $K_{i,j} = \frac{1}{\Delta_i - \Delta_j}$ if $i \neq j$ and 0 otherwise. Finally, the chain rule of the gradient loss with respect to the input feature map $\boldsymbol{X}$ is given by the following equation:

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{X}} = \boldsymbol{X}\left(\frac{\partial \mathcal{L}}{\partial \boldsymbol{\Sigma}} + \frac{\partial \mathcal{L}}{\partial \boldsymbol{\Sigma}}^{\top}\right) \tag{3.34}$$

By doing so, deep network with power normalization or logarithm normalization can be trained end-to-end. This leads to a large performance increase, even on large-scale datasets [105] such as the ILSVRC 2012 [45]. However, the back-propagation formulation is subjected to numerical instability if two eigenvalues or singular values are close to each other. Indeed, the term $K_{i,j} = \frac{1}{\Delta_i - \Delta_j}$ becomes very large and leads to numerical error. Also, the computation of the singular value decomposition and the eigen-decomposition have bad CUDA implementations: numerical instability with 32bits floating point operations, slow implementation, *etc.* Thus, a solution is to use the better CPU implementation which are faster but still slowed down because of the data exchange between CPU and GPU. As a consequence, alternative methods to the computation of the singular value decomposition have been proposed in order to reduce the computation. In the case of matrix square-root, *i.e.*, when $\boldsymbol{Q} = \sqrt{\boldsymbol{\Sigma}}$, one way is to consider the following Lyapunov equation [109]:

$$\sqrt{\boldsymbol{\Sigma}}d\boldsymbol{Q} + d\boldsymbol{Q}\sqrt{\boldsymbol{\Sigma}} = d\boldsymbol{\Sigma} \tag{3.35}$$

From which we get the chain rule:

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{Q}} = \sqrt{\boldsymbol{\Sigma}}\frac{\partial \mathcal{L}}{\partial \boldsymbol{\Sigma}} + \frac{\partial \mathcal{L}}{\partial \boldsymbol{\Sigma}}\sqrt{\boldsymbol{\Sigma}} \tag{3.36}$$

This equation has the following closed-form:

$$\text{vec}\left(\frac{\partial \mathcal{L}}{\partial \boldsymbol{\Sigma}}\right) = (\sqrt{\boldsymbol{\Sigma}} \star \boldsymbol{I} + \boldsymbol{I} \star \sqrt{\boldsymbol{\Sigma}})^{-1}\text{vec}\left(\frac{\partial \mathcal{L}}{\partial \boldsymbol{\Sigma}}\right) \tag{3.37}$$

where $\star$ is the Kronecker product between matrices. For example, let us consider two matrices $\boldsymbol{A}, \boldsymbol{B}$ in $\mathbb{R}^{2\times 2}$ such that $\boldsymbol{A} = \begin{pmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{pmatrix}$ and $\boldsymbol{B} = \begin{pmatrix} B_{1,1} & B_{1,2} \\ B_{2,1} & B_{2,2} \end{pmatrix}$. The matrix $\boldsymbol{C} = \boldsymbol{A} \star \boldsymbol{B} \in \mathbb{R}^{4\times 4}$ is computed as follows:

$$\boldsymbol{C} = \begin{pmatrix} A_{1,1}B_{1,1} & A_{1,1}B_{1,2} & A_{1,2}B_{1,1} & A_{1,2}B_{1,2} \\ A_{1,1}B_{2,1} & A_{1,1}B_{2,2} & A_{1,2}B_{2,1} & A_{1,2}B_{2,2} \\ A_{2,1}B_{1,1} & A_{2,1}B_{1,2} & A_{2,2}B_{1,1} & A_{2,2}B_{1,2} \\ A_{2,1}B_{2,1} & A_{2,1}B_{2,2} & A_{2,2}B_{2,1} & A_{2,2}B_{2,2} \end{pmatrix} \tag{3.38}$$

More generally, the Kronecker product between two matrices $\boldsymbol{A} \in \mathbb{R}^{n\times m}$ and $\boldsymbol{B} \in \mathbb{R}^{p\times q}$ gives a matrix $\boldsymbol{C} \in \mathbb{R}^{np\times mq}$ defined by block: the top-left block is $A_{1,1}\boldsymbol{B}$, the block to its right is $A_{1,2}\boldsymbol{B}$, and so on. Compared to the Kronecker product defined in Equation 3.6, the Kronecker product between two matrices can be seen as computing the Kronecker product between the two matrices, permuting some dimensions and reshaping the tensor. To avoid future confusion between Kronecker product and Kronecker product between matrices, we will only use the computation from Equation 3.6 to refer to the Kronecker product in the rest of this thesis.

By using this closed-form, the numerical instability of the matrix back-propagation [80] is avoided. However, the closed-form of the Lyapunov equation requires to compute the inverse of a $c^2 \times c^2$ for c-dimensional features which leads to a complexity in $\mathcal{O}(c^6)$. Thus, this formulation is usually intractable without reducing the dimensionality of the deep features. An efficient way to solve the Lyapunov equation is to use the Bartels-Stewart algorithm [9] which first simplifies the problem by applying the Schur decomposition, then solves the easier problem before recomputing the original matrix from the decomposition. By doing so, the complexity is reduced from $\mathcal{O}(c^6)$ to $\mathcal{O}(c^3)$ and still has the numerical stability, thanks to the Schur decomposition. This approach leads to strong performances on fine-grained classification tasks [109]. Another way to compute the eigen-decomposition is to use a divide-and-conquer approach and the power-iteration together [124]. By doing so, the backward of the singular value decomposition and the eigen-decomposition have been proven to be stable. However, the power-iteration does not have a fixed number of steps. Indeed, if two eigenvalues are close to each other, the power-iteration methods need a large number of steps to converge to the correct result. Thus, in the case of deep learning, we cannot directly use this approach by unrolling the iterations. To solve this issue, Wang et al. [181] exploits a hybrid approach: for the forward-pass, the computation relies on the power-iterations [124] because it is more stable. For the backward-pass, instead of using directly the chain rule over the power-iterations, they replace each eigen-vector for each step of the power-iteration by the real one (*i.e.*, the one computed in the forward-pass). By doing so, the computation of the backward-pass is simplified as follows (Equation 4 from [181]):

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{\Sigma}} = \left(\sum_{k=1}^{K} \frac{\boldsymbol{\Sigma}^{k-1} \left(\boldsymbol{I} - \boldsymbol{v}\boldsymbol{v}^\top \right)}{\|\boldsymbol{\Sigma}\boldsymbol{v}\|_2^k} \right) \frac{\partial \mathcal{L}}{\partial \boldsymbol{v}^{(K)}} \boldsymbol{v}^\top \tag{3.39}$$

where $\boldsymbol{\Sigma}^0 = \boldsymbol{I}$ and $\frac{\partial \mathcal{L}}{\partial \boldsymbol{v}^{(K)}}$ is the gradient of the loss with respect to the last iteration of the power-norm. While this approach still requires to set the number of power-iterations to compute the backward-pass, in practice, setting $K = 20$ is enough to have $\left(\frac{\sigma_i}{\sigma_1}\right)^K \leq 0.05$ where σ_1 is the largest eigenvalue and σ_i is another eigenvalue. In the particular case of matrix-square-root, one can also consider iterative approaches such as the Denman-Beavers iterations which are computed as follows (see [75], Chapter 6.7):

$$\begin{aligned} \boldsymbol{Y}_{k+1} &= \frac{1}{2} \left(\boldsymbol{Y}_k + \boldsymbol{Z}_k^{-1} \right) \\ \boldsymbol{Z}_{k+1} &= \frac{1}{2} \left(\boldsymbol{Y}_k^{-1} + \boldsymbol{Z}_k \right) \end{aligned} \tag{3.40}$$

where $\boldsymbol{Y}_0 = \boldsymbol{\Sigma}$ and $\boldsymbol{Z} = \boldsymbol{I}$. As k growth, the matrices $\boldsymbol{Y}$ and $\boldsymbol{Z}$ converge respectively to $\boldsymbol{\Sigma}^{\frac{1}{2}}$ and $\boldsymbol{\Sigma}^{-\frac{1}{2}}$ with a quadratic growth. This approach still has two main issues: First, the above equation only converges locally, that it if and only if $\|\boldsymbol{\Sigma} - \boldsymbol{I}\|_{\mathcal{F}} < 1$. Second, the above equation still relies on matrix inversion. These iteration-based approaches have been shown to belong to a larger set of methods [76], that is, there exists two polynomials $p_{k,l}$ and $q_{k,l}$ such that:

$$\begin{aligned} \boldsymbol{Y}_{k+1} &= \boldsymbol{Y}_k \; p_{k,l} \left(\boldsymbol{Z}_k \boldsymbol{Y}_k \right) \; \; q_{k,l} \left(\boldsymbol{Z}_k \boldsymbol{Y}_k \right)^{-1} \\ \boldsymbol{Z}_{k+1} &= \left(\boldsymbol{Z}_k \boldsymbol{Y}_k \right) q_{k,l} \left(\boldsymbol{Z}_k \boldsymbol{Y}_k \right)^{-1} \; \; \boldsymbol{Z}_k \end{aligned} \tag{3.41}$$

Specifically, the case $k = 0, l = 1$ is known as the Newton-Schulz iterations:

$$\begin{aligned} \boldsymbol{Y}_{k+1} &= \frac{1}{2} \boldsymbol{Y}_k \left(3\boldsymbol{I} - \boldsymbol{Z}_k \boldsymbol{Y}_k \right) \\ \boldsymbol{Z}_{k+1} &= \frac{1}{2} \left(3\boldsymbol{I} - \boldsymbol{Z}_k \boldsymbol{Y}_k \right) \boldsymbol{Z}_k \end{aligned} \tag{3.42}$$

The Newton-Schulz iterations are GPU-friendly because they only rely on matrix product which is suitable for parallel computing. This approach has been explored to compute the matrix-square-root of the covariance matrix in the forward-pass [109] but also in both forward and backward passes [104]. In practice, the local convergence of these approaches are handled by scaling the covariance matrix, *e.g.* with its trace or its Frobenius norm [104]. A post-compensation of the square-root of the trace or the Frobenius norm can be performed in order to cancel the effect of the pre-compensation.

As a conclusion, normalization is tightly connected with the geometry of the SPD manifold which is known to be Riemannian. While geodesic distances such as the affine-invariant Riemannian norm or the Jensen-Bregman divergence can be used to perfectly describe the SPD manifold, they are not practical for many tasks, *e.g.* metric learning. Indeed, at query time, eigen-decomposition, matrix inversion and/or matrix determinant have to be computed for all images from which we want to evaluate the similarity with the query. Thus, these distances are replaced by the either log-Euclidean which is a geodesic distance but which cannot handle low-rank SPD matrices or pow-Euclidean distance that "flatten" the Riemannian space: The Euclidean distance is still not a geodesic distance, but it approximates a geodesic distance better with this normalization than without. Thanks to the log-Euclidean and the pow-Euclidean formulations, the normalization can be performed offline and the distance becomes the simple Euclidean distance to measure similarities which is a hundred of times faster than the above Riemannian distances. It follows that most of the state-of-the-art methods propose better computation approaches to fasten the matrix power-norm or log-norm using matrix back-propagation approaches, closed-form computation, iterative approaches, and so on.

3.4 Factorization

As illustrated in the visual question answering example from section 3.2, the dimensionality of second-order representations is too large to be used as it is. Especially, when recent contributions show that deeper network with larger representation size leads to higher performances, the dimensionality of the second-order representation is the main limitation for these pooling strategies. To cope with this drawback, dimensionality reduction strategies are considered.

Linear projection such as principal component analysis (PCA) can be used to reduce the dimensionality of the representation. By keeping only the D eigenvectors related to the D highest eigenvalues, the c^2-dimensional second-order representation can be reduced to D. However, this approach, while being simple, is not efficient in practice for many reasons: First, the number of parameters increases with a cubic growth in $\mathcal{O}(c^2D)$ which means that backbone network with small dimensionality are required. Second, the computation of the PCA is also in $\mathcal{O}(c^2D)$ due to the eigen-decomposition and the matrix multiplication. Third, D cannot be too small without greatly reducing performances. Indeed, Gao et al. [56] have reported results on the Cub-200-2011 dataset for the classification task with VGG-M [27]: The second-order moment with $c = 512$ has its performances dropped from 77.6% accuracy without dimensionality reduction to 63.8% using a PCA with $D = 16384$. Even if we can train the linear projection and the network together, the performance still drops to 76.2% with $D = 16384$. This is less than 1.5% below the baseline with less than 7% of the original dimensionality but at the cost of 4.3 billion parameters. Because of this large number of parameters, recent contributions have explored better dimensionality reduction strategies, which usually rely on matrix factorization.

For example, Gao et al. [56] have proposed two factorizations based on random projection strategies. To measure the similarity between two second-order representations $\boldsymbol{\Sigma}_1 = \frac{1}{N}\sum_i \boldsymbol{x}_i \otimes \boldsymbol{x}_i$ and $\boldsymbol{\Sigma}_2 = \frac{1}{N}\sum_j \boldsymbol{x}_j \otimes \boldsymbol{x}_j$, they consider the Frobenius dot product which can be re-written as follows:

$$\begin{aligned}
\langle \boldsymbol{\Sigma}_1 \ ; \ \boldsymbol{\Sigma}_2 \rangle_{\mathcal{F}} &= \frac{1}{N^2} \sum_i \sum_j \langle \boldsymbol{x}_i \otimes \boldsymbol{x}_i \ ; \ \boldsymbol{x}_j \otimes \boldsymbol{x}_j \rangle_{\mathcal{F}} \\
&= \frac{1}{N^2} \sum_i \sum_j \langle \boldsymbol{x}_i \ ; \ \boldsymbol{x}_j \rangle^2 \\
&\approx \left\langle \frac{1}{N} \sum_i \boldsymbol{\phi}(\boldsymbol{x}_i) \ ; \ \frac{1}{N} \sum_j \boldsymbol{\phi}(\boldsymbol{x}_j) \right\rangle
\end{aligned} \tag{3.43}$$

which is a second-order polynomial kernel matching between the deep features. Thus, this kernel can be expressed using the kernel trick with a function $\boldsymbol{\phi}$. Then, they propose to exploit to random projection methods named Random Maclaurin [93] and Tensor Sketch [135] in order to explicitly express the function $\boldsymbol{\phi}$.

The RM algorithm defines two random vectors $\boldsymbol{w}_1 \in \{-1,+1\}^c$ and $\boldsymbol{w}_2 \in \{-1,+1\}^c$ which are uniformly and independently sampled within the set $\{-1,+1\}$. The function ϕ is defined as $\phi(\boldsymbol{x}_i) = \langle \boldsymbol{w}_1 \ ; \ \boldsymbol{x}_i \rangle \langle \boldsymbol{w}_2 \ ; \ \boldsymbol{x}_i \rangle$. Thus, in expectation we have the following result:

$$\begin{aligned}\mathbb{E}_{\boldsymbol{w}_1,\boldsymbol{w}_2}\left[\phi(\boldsymbol{x}_i)\phi(\boldsymbol{x}_j)\right] &= \mathbb{E}_{\boldsymbol{w}_1,\boldsymbol{w}_2}\left[\langle \boldsymbol{w}_1 \ ; \ \boldsymbol{x}_i \rangle \langle \boldsymbol{w}_2 \ ; \ \boldsymbol{x}_i \rangle \langle \boldsymbol{w}_1 \ ; \ \boldsymbol{x}_j \rangle \langle \boldsymbol{w}_2 \ ; \ \boldsymbol{x}_j \rangle\right] \\ &= \mathbb{E}_{\boldsymbol{w}_1}\left[\langle \boldsymbol{w}_1 \ ; \ \boldsymbol{x}_i \rangle \langle \boldsymbol{w}_1 \ ; \ \boldsymbol{x}_j \rangle\right] \mathbb{E}_{\boldsymbol{w}_2}\left[\langle \boldsymbol{w}_2 \ ; \ \boldsymbol{x}_i \rangle \langle \boldsymbol{w}_2 \ ; \ \boldsymbol{x}_j \rangle\right] \\ &= \langle \boldsymbol{x}_i \ ; \ \boldsymbol{x}_j \rangle^2 \end{aligned} \tag{3.44}$$

This result comes from the facts that $\mathbb{E}[\boldsymbol{w}_i] = \boldsymbol{0}_c$ and $\mathbb{E}[\boldsymbol{w}_i \boldsymbol{w}_i^\top] = \boldsymbol{I}_c$ as each vector entry has a mean of 0, a variance of 1 and is independent from the other entries. The expectation is then replaced by its empirical estimator: The Random Maclaurin algorithm considers two random matrices $\boldsymbol{W}_1 \in \{-1,+1\}^{c \times D}$ and $\boldsymbol{W}_2 \in \{-1,+1\}^{c \times D}$ from which all entries are uniformly and independently sampled within $\{-1,+1\}$. Thus, by considering the k-th column of $\boldsymbol{W}_1$ and $\boldsymbol{W}_2$ named $\boldsymbol{w}_{1,k}$ and $\boldsymbol{w}_{2,k}$ respectively, we have $\phi_k(\boldsymbol{x}_i) = \langle \boldsymbol{w}_{1,k} \ ; \ \boldsymbol{x}_i \rangle \langle \boldsymbol{w}_{2,k} \ ; \ \boldsymbol{x}_i \rangle$. By doing so, the above equation becomes:

$$\begin{aligned}\mathbb{E}\left[\phi_k(\boldsymbol{x}_i)\phi_k(\boldsymbol{x}_j)\right] &\approx \frac{1}{D}\sum_{k=1}^{D} \phi_k(\boldsymbol{x}_i)\phi_k(\boldsymbol{x}_j) \\ &= \left\langle \frac{1}{\sqrt{D}}\boldsymbol{\phi}(\boldsymbol{x}_i) \ ; \ \frac{1}{\sqrt{D}}\boldsymbol{\phi}(\boldsymbol{x}_j) \right\rangle \end{aligned} \tag{3.45}$$

where $\boldsymbol{\phi}(\boldsymbol{x}_i) = \left(\left(\boldsymbol{W}_1^\top \boldsymbol{x}_i\right) \odot \left(\boldsymbol{W}_2^\top \boldsymbol{x}_i\right)\right)$. This expression is very efficient in practice as it can be implemented with a 1×1 convolution of the deep features with D filters followed by an element-wise product between the two feature maps and finally a global average pooling of these deep features. Also, the number of parameters and the computation complexity have a quadratic growth in $\mathcal{O}(cD)$ which is much more efficient than directly computing a PCA on the second-order representation.

The TS algorithm takes advantage of sketching functions in order to provide the desired approximation which is both efficient in number of parameters and in computation. In TS, two sketching functions $\boldsymbol{\psi}_1(\cdot \ ; \ \boldsymbol{h}_1, \boldsymbol{s}_1)$ and $\boldsymbol{\psi}_2(\cdot \ ; \ \boldsymbol{h}_2, \boldsymbol{s}_2)$ are defined where $\boldsymbol{h}_1 \in [D]^c, \boldsymbol{h}_2 \in [D]^c, \boldsymbol{s}_1 \in \{-1,+1\}^c, \boldsymbol{s}_2 \in \{-1,+1\}^c$ are uniformly and independently sampled in their respective sets. Thus, the functions $\boldsymbol{\psi}_k(\cdot \ ; \ \boldsymbol{h}_k, \boldsymbol{s}_k)$ for a feature $\boldsymbol{x}_i$ is computed such that:

$$\psi_{k,j}(\boldsymbol{x}_i \ ; \ \boldsymbol{h}_k, \boldsymbol{s}_k) = \sum_{t, h_{k,t}=j} s_{k,t} x_{i,t} \tag{3.46}$$

where $\psi_{k,j}$ is the j-th dimension of $\boldsymbol{\psi}_k$ and $h_{k,t}$ is the t-th dimension of $\boldsymbol{h}_k$. These sketching functions can be used to approximate the second-order polynomial kernel thanks to the two following properties (see [135] for proof):

$$\begin{cases} \mathbb{E}\left[\langle \boldsymbol{\psi}_1(\boldsymbol{x}_i \ ; \ \boldsymbol{h}_1, \boldsymbol{s}_1) \ ; \ \boldsymbol{\psi}_2(\boldsymbol{x}_j \ ; \ \boldsymbol{h}_2, \boldsymbol{s}_2)\rangle\right] = \langle \boldsymbol{x}_i \ ; \ \boldsymbol{x}_j \rangle \\ \boldsymbol{\psi}(\boldsymbol{x}_i \otimes \boldsymbol{x}_i \ ; \ \boldsymbol{h}, \boldsymbol{s}) = \boldsymbol{\psi}_1(\boldsymbol{x}_i \ ; \ \boldsymbol{h}_1, \boldsymbol{s}_1) * \boldsymbol{\psi}_2(\boldsymbol{x}_i \ ; \ \boldsymbol{h}_2, \boldsymbol{s}_2) \end{cases} \tag{3.47}$$

where $*$ is the convolution operator. In practice, the computation of TS is performed with the Fourier transform: This greatly reduces the computation but also it withdraws unintended high frequencies that are introduced by the hashing functions. The full computation for TS function, $\boldsymbol{\phi}_{\mathrm{TS}}$ is performed as follows:

$$\boldsymbol{\phi}_{\mathrm{TS}}(\boldsymbol{x}_i) = \mathrm{FFT}^{-1}\left(\mathrm{FFT}\left(\boldsymbol{\psi}_1(\boldsymbol{x}_i \ ; \ \boldsymbol{h}_1, \boldsymbol{s}_1)\right) \odot \mathrm{FFT}\left(\boldsymbol{\psi}_2(\boldsymbol{x}_i \ ; \ \boldsymbol{h}_2, \boldsymbol{s}_2)\right)\right) \tag{3.48}$$

The output dimensionality is then set by retaining only the D smallest frequencies in the Fourier transform. TS algorithm is much more efficient in terms of parameters and computation complexity than the direct projection of the second-order representation but also than the RM algorithm. Indeed, the number of parameters is in $\mathcal{O}(c)$ and the complexity is in $\mathcal{O}(D \log D + c)$ for TS compared to RM where both are in $\mathcal{O}(cD)$. Also, performances obtained from RM and TS are much more interesting than using a PCA, even when the weights are learned: the accuracy on the Cub-200-2011 dataset is increased by more than 1% with a representation of size $D = 16384$, and performances are the same compared to the representation without dimensionality reduction. Thus, TS algorithm has

been widely used to reduce the dimensionality of second-order representations [56, 55, 64, 161]. Furthermore, reducing the dimensionality to $D = 4096$ only decreases the accuracy to 76.8% which is still higher than the plain PCA with $D = 16384$. The RM algorithm is between the plain PCA and the TS algorithm in terms of performances. For more ablations, we refer the reader to the Figure 2 from [56]. However, the major problem of TS, and to a lesser extent the one of RM, is that we cannot learn the parameters. Thus, lots of recent contributions in factorization for the second-order representation are based on finding efficient factorization of the costly linear projection computed on the second-order representations.

These factorizations strategies start from a plain linear projection of the second order representation in order to produce the output representation $\boldsymbol{z} \in \mathbb{R}^D$:

$$\boldsymbol{z} = \sum_i \boldsymbol{z}(\boldsymbol{x}_i) = \sum_i \boldsymbol{\mathcal{W}} \times_2 \boldsymbol{x}_i \times_3 \boldsymbol{x}_i \tag{3.49}$$

where $\boldsymbol{\mathcal{W}} \in \mathbb{R}^{D \times c \times c}$. As already mentioned, due to the large number of parameters in $\boldsymbol{\mathcal{W}}$, state-of-the-art methods have used factorization strategies of the tensor $\boldsymbol{\mathcal{W}}$ in order to reduce the number of parameters and the computation cost. The first and simplest strategy is to consider a *rank-one approximation* of the k-th slice of the tensor $\boldsymbol{\mathcal{W}}$, named $\boldsymbol{W}_k \in \mathbb{R}^{c \times c}$. By denoting z_k the k-th dimension of $\boldsymbol{z}$, the above equation can be re-written using the Frobenius dot product:

$$z_k(\boldsymbol{x}_i) = \langle \boldsymbol{W}_k \ ; \ \boldsymbol{x}_i \otimes \boldsymbol{x}_i \rangle_{\mathcal{F}} \tag{3.50}$$

$$= \boldsymbol{x}_i^\top \boldsymbol{W}_k \ \boldsymbol{x}_i \tag{3.51}$$

Thus, the matrix $\boldsymbol{W}_k$ is approximated using the following rank-one approximation $\boldsymbol{W}_k = \lambda_k \boldsymbol{u}_k \boldsymbol{v}_k^\top = \lambda_k \boldsymbol{u}_k \otimes \boldsymbol{v}_k$. The computation of $z_k(\boldsymbol{x}_i)$ is simplified as follows:

$$z_k(\boldsymbol{x}_i) = \lambda_k \langle \boldsymbol{u}_k \ ; \ \boldsymbol{x}_i \rangle \langle \boldsymbol{v}_k \ ; \ \boldsymbol{x}_i \rangle \tag{3.52}$$

A solution to get the best rank-one approximation of $\boldsymbol{W}_k$ is to compute the singular value decomposition of $\boldsymbol{W}_k$ and to keep the vectors $\boldsymbol{u}$ and $\boldsymbol{v}$ which correspond with the largest singular value λ. Also, another preferred solution is to learn the vectors $\boldsymbol{u}_k$ and $\boldsymbol{v}_k$ (λ_k is integrated within the norm of these two vectors) in order to learn a factorization that integrates the semantic information. By doing so, this factorization can be easily re-written by considering two matrices $\boldsymbol{U} \in \mathbb{R}^{c \times D}$ and $\boldsymbol{V} \in \mathbb{R}^{c \times D}$ such that:

$$\boldsymbol{z}(\boldsymbol{x}_i) = \left(\boldsymbol{U}^\top \boldsymbol{x}_i\right) \odot \left(\boldsymbol{V}^\top \boldsymbol{x}_i\right) \tag{3.53}$$

This approximation is widely used [19, 196] thanks to its simplicity and efficiency: Both the number of parameters and the computation complexity have a quadratic growth in $\mathcal{O}(cD)$. Interestingly, the above factorization is very similar to the RM algorithm. The main difference is that, in this case, all parameters are learned in order to integrate the semantic information during the training. Empirically, Gao et al. [56] reported results on the Cub-200-2011 dataset for classification task with this factorization (*i.e.*, RM algorithm where the weights are trainable): This slightly improves performances compared to the non-learnable weights (see Figure 2 from [56]).

A direct extension of the rank-one approximation is to consider *multi-rank approximation*. That is, the factorization of the matrix $\boldsymbol{W}_k \in \mathbb{R}^{c \times c}$ is replaced by $\boldsymbol{W}_k = \sum_{r=1}^{R} \lambda_r \boldsymbol{u}_{k,r} \otimes \boldsymbol{v}_{k,r}$ where R is the chosen rank. By linearity, the rank-one approximation is directly extended as follows:

$$\boldsymbol{z}(\boldsymbol{x}_i) = \sum_{r=1}^{R} \left(\boldsymbol{U}_r^\top \boldsymbol{x}_i\right) \odot \left(\boldsymbol{V}_r^\top \boldsymbol{x}_i\right) \tag{3.54}$$

This approach has been less used than the rank-one approximation because the improvement in terms of performances is small while the number of parameters has a cubic growth in $\mathcal{O}(RcD)$ the same as the computation complexity. In practice, this approach is more subjected to over-fitting than the rank-one approximation which greatly reduces its benefits. Still, some contributions consider such factorization [186, 108, 100] with feature dropout, small number of parameters, and so on.

The main drawback of the rank-one and the multi-rank approximations is that they consider all output dimensions independently. This leads to a large number of parameters and potential output

dimensions that are highly correlated. To cope with this issue, Kim et al. [97] have proposed a *shared multi-rank* approximation named Hadamard Product Bilinear Pooling (HPBP). The idea is, instead of having R projections that are learned per output dimensions, HPBP use a larger rank to produce R projections that are then shared in order to recombine the output representation. More precisely, HPBP defines three matrices $\boldsymbol{U} \in \mathbb{R}^{c \times R}, \boldsymbol{V} \in \mathbb{R}^{c \times R}$ and $\boldsymbol{P} \in \mathbb{R}^{R \times D}$ such that the output representation $\boldsymbol{z}(\boldsymbol{x}_i)$ is computed as follows:

$$\boldsymbol{z}(\boldsymbol{x}_i) = \boldsymbol{P}^\top \left(\left(\boldsymbol{U}^\top \boldsymbol{x}_i \right) \odot \left(\boldsymbol{V}^\top \boldsymbol{x}_i \right) \right) \tag{3.55}$$

For the k-th output dimension of $\boldsymbol{z}(\boldsymbol{x}_i)$, the computation is performed as follows:

$$z_k(\boldsymbol{x}_i) = \sum_{r=1}^{R} p_{k,r} \left\langle \boldsymbol{u}_r \ ; \ \boldsymbol{x}_i \right\rangle \left\langle \boldsymbol{v}_r \ ; \ \boldsymbol{x}_i \right\rangle \tag{3.56}$$

Compared to the rank-one or the multi-rank, HPBP has completely reversed the factorization. That is, where the two first approaches learn independently the low-rank approximation for each matrix $\boldsymbol{W}_k$, HPBP consider a set of vectors and learn the weights of each projection. Interestingly, if we consider the sets $\{\boldsymbol{u}_r\}_{r=1}^R$ and $\{\boldsymbol{v}_r\}_{r=1}^R$ as two bases in $\mathbb{R}^c$, the set $\{\boldsymbol{u}_r \otimes \boldsymbol{v}_s\}_{r,s=1}^R$ is a vector basis for the vector space $\mathbb{R}^{c^2}$. This means that HPBP directly learns the weights of the linear combination through the matrix $\boldsymbol{P}$ and the matrices $\boldsymbol{U}$ and $\boldsymbol{V}$ are two bases selected in order to have a sparse reconstruction of the vectors. Indeed, having $R^2 \leq c^2$ means that $c^2 - R^2$ weights for some combination $\boldsymbol{u}_r \otimes \boldsymbol{v}_s$ are set to 0. Finally, this factorization is both interesting in number of parameters and in computation complexity as it is a compromise between the rank-one and the multi-rank factorization: They are both in $\mathcal{O}(DR + Rc)$ instead of $\mathcal{O}(Dc)$ and $\mathcal{O}(RDc)$ for the rank-one and the multi-rank factorizations respectively.

In all previous cases, the matrices $\boldsymbol{U}$ and $\boldsymbol{V}$ are totally independent and one column from $\boldsymbol{U}$ corresponds with the column at the same position in $\boldsymbol{V}$. That is, the factorization only considers the cases $\boldsymbol{u}_r \otimes \boldsymbol{v}_r$ and forgets about the cases $\boldsymbol{u}_r \otimes \boldsymbol{v}_s$ when $s \neq r$. In the case of the rank-one factorization, if we consider all interactions, the computation of $\boldsymbol{z}(\boldsymbol{x}_i)$ is modified as follows:

$$\begin{aligned} \boldsymbol{z}(\boldsymbol{x}_i) &= \operatorname{vec}\left(\left(\boldsymbol{U}^\top \boldsymbol{x}_i \right) \left(\boldsymbol{V}^\top \boldsymbol{x}_i \right)^\top \right) \\ &= \operatorname{vec}\left(\boldsymbol{U}^\top \boldsymbol{x}_i \boldsymbol{x}_i^\top \boldsymbol{V} \right) \end{aligned} \tag{3.57}$$

This factorization is easily implemented with two 1×1 convolutions followed by a bilinear pooling. Also, both the number of parameters and the computation complexity have a quadratic growth in $\mathcal{O}(cR)$ where $R \leq c$ and the dimensionality of the representation is R^2. This is much more efficient than any previous factorization. In practice, the dimensionality can be decreased to similar values than TS or the multi-rank factorization [100]. Furthermore, by setting $\boldsymbol{V} = \boldsymbol{U}$, this formulation is the affine transformation mentioned in section 3.3. Thus, combined with the geodesic distances that have been presented in section 3.3, this property can be used as a geometry-aware dimensionality reduction [69, 68].

For most of all the previous factorizations, each slice $\boldsymbol{W}_k$ of the full tensor $\boldsymbol{\mathcal{W}}$ are recomputed independently to each other. To cope with this issue, one can rely on tensor factorization methods such as the canonical polyadic (CP) decomposition. For example, in the case of the third-order tensor $\boldsymbol{\mathcal{W}} \in \mathbb{R}^{D \times c \times c}$, the CP decomposition is a low-rank factorization of the tensor which is computed as follows:

$$\boldsymbol{\mathcal{W}} = \sum_{r=1}^{R} \lambda_r \ \boldsymbol{w}_r \otimes \boldsymbol{u}_r \otimes \boldsymbol{v}_r \tag{3.58}$$

where R is the rank of the decomposition and for all r, $\boldsymbol{w}_r \in \mathbb{R}^D, \boldsymbol{u}_r \in \mathbb{R}^c$ and $\boldsymbol{v}_r \in \mathbb{R}^c$. By doing so, the computation of the output representation $\boldsymbol{z}(\boldsymbol{x}_i) \in \mathbb{R}^D$ is computed as follows:

$$\begin{aligned} \boldsymbol{z}(\boldsymbol{x}_i) &= \boldsymbol{\mathcal{W}} \times_2 \boldsymbol{x}_i \times_3 \boldsymbol{x}_i \\ &= \sum_{r=1}^{R} \lambda_r \left\langle \boldsymbol{u}_r \ ; \ \boldsymbol{x}_i \right\rangle \left\langle \boldsymbol{v}_r \ ; \ \boldsymbol{x}_i \right\rangle \boldsymbol{w}_r \end{aligned} \tag{3.59}$$

Interestingly, the CP decomposition is exactly the same factorization as HPBP but with another interpretation. However, in the case of matrices, the optimal factorization (in the sense of the mean square error) always exists, is well-defined and can be found using, *e.g.*, the singular value decomposition. In the case of tensors with a number of orders greater or equal to three, the best rank-R factorization is an ill-posed problem [43] and does not necessary exist (see examples 7.1 in [77] and a study on these tensors in [43]) and if so, finding the best decomposition is NP-hard [77].

Another problem with the CP decomposition is that it does not consider independently each order. For example, in the case of a tensor $\boldsymbol{\mathcal{T}} \in \mathbb{R}^{2,20,20}$, the rank of the first order is at most 2 which means that we do not need more than two components to represent the first order of the tensor. To cope with this issue, Tucker decomposition has been designed as an extension of the singular value decomposition for matrices to tensors. If we consider the Tucker decomposition of the third-order tensor $\boldsymbol{\mathcal{W}} \in \mathbb{R}^{D \times c \times c}$ for the factorization, the Tucker decomposition of $\boldsymbol{\mathcal{W}}$ is computed as follows:

$$\boldsymbol{\mathcal{W}} = \boldsymbol{\mathcal{W}}_c \times_1 \boldsymbol{U}_1^\top \times_2 \boldsymbol{U}_2^\top \times_3 \boldsymbol{U}_3^\top \tag{3.60}$$

where $\boldsymbol{\mathcal{W}}_c \in \mathbb{R}^{d \times c_1 \times c_1}$ is a third-order tensor with $d < D$ and $c_1 < c$ usually referred as the core tensor and $\boldsymbol{U}_1 \in \mathbb{R}^{D \times d}, \boldsymbol{U}_2 \in \mathbb{R}^{c \times c_1}$ and $\boldsymbol{U}_3 \in \mathbb{R}^{c \times c_1}$ are projection matrices. Compared to the singular value decomposition, the core tensor plays the same role as the singular values with a generalization to tensors and the matrices $\boldsymbol{U}_k$ play the same role as the left and right projection matrices. However, as in the case of the CP decomposition, finding the optimal factorization is NP-hard. Still, a factorization is usually got from the high-order singular value decomposition [42] which gives a systematic approach to find a solution for the factorization. Also, an approximation bound is given with respect to the real best tensor approximation:

$$\|\boldsymbol{\mathcal{W}} - \widetilde{\boldsymbol{\mathcal{W}}}\|_{\mathcal{F}} \leq \sqrt{N} \|\boldsymbol{\mathcal{W}} - \boldsymbol{\mathcal{W}}^*\|_{\mathcal{F}} \tag{3.61}$$

where $\boldsymbol{\mathcal{W}}$ is the full tensor that we want to approximate, $\widetilde{\boldsymbol{\mathcal{W}}}$ is the tensor obtained from the high-order singular value decomposition, $\boldsymbol{\mathcal{W}}^*$ is the optimal tensor and N is the number of order for the tensor $\boldsymbol{\mathcal{W}}$ (*i.e.*, 3 in our case). Tucker decomposition has been used to factorize projection matrices for visual question answering tasks [13, 14]. In MUTAN [13], the authors have proposed this factorization in order to reduce the size of the projection tensor to generate the answers. Furthermore, as the size of the core tensor is still too large, this core tensor has further been factorized using CP decomposition. By doing so, they also show that some directions in this factorization have specialized to recognize specific patterns which can be used to interpret the answer of the proposed method. This approach have led to excellent results on visual question answering datasets. BLOCK [14] further extends this Tucker decomposition with a sparse structure in the core tensor. Instead of the CP decomposition, BLOCK has a block-diagonal sparse structure approach. By doing so, bilinear product is performed between a reduced set of dimensions per block. Also, the use of a block-diagonal structure supposes that some dimensions are not correlated and their projections are set to 0. This naturally tends to reduce over-fitting by structurally not taking into account every correlations but the only relevant one.

In conclusion, factorization strategies are mandatory to most of tasks because of the dimensionality of second-order representations. For example, a baseline model for a visual question answering task would lead to more than 8 billion parameters only to learn the classifier above the bilinear pooling of the visual embedding and the question embedding. Low-rank factorizations are usually used to reduce the dimensionality of these second-order representations by finding a low-rank approximation of the projection matrix. These approximations can be extended by sharing the projectors between output dimensions to reduce the number of parameters or by considering affine transformation with geodesic distances to have a geometry-aware dimensionality reduction. Also, tensor decomposition such as Tucker and CP decompositions can be exploited to factorize high-order tensors for an additional boost in performances as it is done in MUTAN and BLOCK. An open question for the following section can be raised: How can we both integrate the normalization strategies from section 3.3 with most of the factorization proposed in this section.

3.5 Beyond Second-Order Pooling

In this section, we present related work that extends second-order representations by considering normalization and factorization together, dictionary learning, higher-order, *etc.* One but simple way to extend second-order representation is to bring back the first-order representation into a new representation which is usually referred as Gaussian pooling [179, 64]. Wang et al. [179] have proposed G^{2}DeNet, a Global Gaussian Distribution Embedding Network, which both integrates the first and second-order in the representation on a deep network. By slightly modifying the deep features as $\tilde{\boldsymbol{x}}_i = [x_{i,1}, \dots, x_{i,c}, 1]^\top \in \mathbb{R}^{c+1}$ where $x_{i,j}$ is the j-th entry of $\boldsymbol{x}_i$ and by denoting $\widetilde{\boldsymbol{X}} \in \mathbb{R}^{c+1 \times N}$ where its j-th column is $\frac{1}{\sqrt{N}}\boldsymbol{x}_i$, the Gaussian pooling is computed as follows:

$$\boldsymbol{G} = \widetilde{\boldsymbol{X}}\widetilde{\boldsymbol{X}}^\top = \begin{pmatrix} \boldsymbol{\Sigma} & \boldsymbol{\mu} \\ \boldsymbol{\mu}^\top & 1 \end{pmatrix} \tag{3.62}$$

Because the set of d-dimensional Gaussian is known to be a Riemannian manifold [156], one can use the log-Euclidean distance or the pow-Euclidean distance instead of the standard Euclidean distance in order to take into account the geometry of this space. Thus, G^{2}DeNet computes the matrix-square-root of $\boldsymbol{G}$ and uses this representation with the Euclidean distance for classification tasks. A direct extension of Gaussian pooling to mixture of Gaussian pooling [177, 178] have been proposed to cope with the performance loss due to linear dimensionality reduction. Instead of learning one projection matrix with a factorization method, they propose to learn a set of projection matrices which are chosen by a gating strategy [177] or by optimizing the maximum likelihood estimator with von Neumann regularization [178]. Also, second-order representations can also explicitly encode scaling information through hierarchical bilinear pooling [195]. By computing bilinear pooling between different feature maps at different depth, performances of deep network with second-order information are further increased.

An open question that comes from the section 3.4 on factorization strategies has been raised: How can we integrate both factorization and normalization in the second-order representation. Indeed, except for the affine transformation-based method or the plain projection, because the covariance is never computed in the other cases, one cannot use the normalization strategies from section 3.3 in order to have a distance that respect the geometry of the Riemannian manifold. However, having factorization a that avoids the direct computation of the second-order representation is interesting to reduce the computation complexity and to avoid a large memory allocation to store the second-order representations. Gou et al. [64] have modified the formulation in G^{2}DeNet in order to pre-normalize the deep features before doing the covariance or the Gaussian pooling. By considering the singular value decomposition of the feature map $\widetilde{\boldsymbol{X}} = \boldsymbol{U}\boldsymbol{S}\boldsymbol{V}^\top$, the matrix-square-root of the Gaussian pooling is computed as follows:

$$\boldsymbol{G}^{\frac{1}{2}} = \boldsymbol{U}(\boldsymbol{S}\boldsymbol{S}^\top)^{\frac{1}{2}}\boldsymbol{U}^\top \tag{3.63}$$

However, the matrix-square-root of $\boldsymbol{S}$ is ill-defined as $\boldsymbol{S} \in \mathbb{R}^{c+1 \times N}$. Thus, they have proposed to add an helper matrix $\boldsymbol{A} = [\boldsymbol{I}_{c+1}|\boldsymbol{0}] \in \mathbb{R}^{c+1 \times N}$ such that $\boldsymbol{S} = \widetilde{\boldsymbol{S}}\boldsymbol{A}$. By substituting $\boldsymbol{S}$ by $\widetilde{\boldsymbol{S}}$ and by using the fact that $\boldsymbol{A}\boldsymbol{A}^\top = \boldsymbol{I}_{c+1}$, the matrix-square-root is now well-defined:

$$\begin{aligned} \boldsymbol{G}^{\frac{1}{2}} &= \boldsymbol{U}(\widetilde{\boldsymbol{S}}\boldsymbol{A}\boldsymbol{A}^\top\widetilde{\boldsymbol{S}})^{\frac{1}{2}}\boldsymbol{U}^\top && (3.64) \\ &= \boldsymbol{U}\widetilde{\boldsymbol{S}}^{\frac{1}{2}}\widetilde{\boldsymbol{S}}^{\frac{1}{2}}\boldsymbol{U}^\top && (3.65) \\ &= \boldsymbol{Y}\boldsymbol{Y}^\top && (3.66) \end{aligned}$$

where $\boldsymbol{Y} = \boldsymbol{U}\widetilde{\boldsymbol{S}}^{\frac{1}{2}}\boldsymbol{A}$ in order to keep the same number of samples as in the feature map $\boldsymbol{X}$. Consequently, the normalization can be performed on the deep features such that the covariance pooling (or Gaussian pooling) can be performed directly on these deep features. By doing so, one can use all factorizations from section 3.4 on these normalized features in order to avoid the direct computation of the second-order representations, but with the advantages of the normalization. Experimentally, performances of the representation with both the normalization and the factorization are much better than ones with only the factorization.

Recent work also considers the Grassmannian manifold [67, 170, 86, 186], a special case of Riemannian manifolds. A Grassmannian manifold $\mathcal{G}(k,c)$ is a smooth and differentiable manifold based on a vector space $\mathbb{R}^c$ which is composed of all k-dimensional linear subspace of $\mathbb{R}^c$. Each element in $\mathcal{G}(k,c)$ can be represented by an orthogonal matrix $\boldsymbol{U}$ of size $c \times k$ such that $\boldsymbol{U}^\top\boldsymbol{U} = \boldsymbol{I}_k$. A particular property of this manifold is that a given point in $\mathcal{G}(k,c)$ can be represented by many matrices: two matrices $\boldsymbol{U}_1$ and $\boldsymbol{U}_2$ are related to the same point if and only if they span the same vector subspace. Equivalently, $\boldsymbol{U}_1$ and $\boldsymbol{U}_2$ in $\mathbb{R}^{c\times k}$ are related to the same point in $\mathcal{G}(k,c)$ if there exists two matrices $\boldsymbol{R}_1$ and $\boldsymbol{R}_2$ in $\mathcal{GL}_k$ (*i.e.*, the group of orthogonal matrices in $\mathbb{R}^k$) such that $\boldsymbol{U}_1\boldsymbol{R}_1 = \boldsymbol{U}_2\boldsymbol{R}_2$. For computer vision applications, such Grassmannian manifold appears when we consider the k eigenvectors related to the k largest eigenvalues of the eigen-decomposition or for non-square matrices, the k singular vectors of the left projection from the singular value decomposition of the feature map. That is, for two feature maps and their respective singular value decomposition $\boldsymbol{X}_1 = \boldsymbol{U}_1\boldsymbol{S}_1\boldsymbol{V}_1^\top$ and $\boldsymbol{X}_2 = \boldsymbol{U}_2\boldsymbol{S}_2\boldsymbol{V}_2^\top$ in $\mathbb{R}^{c\times N}$, one can use the k first singular vectors from $\boldsymbol{U}_1$ and $\boldsymbol{U}_2$, denoted $\boldsymbol{U}_1^{(k)}$ and $\boldsymbol{U}_2^{(k)}$ respectively, as the extracted features. Then, the two feature maps are similar if $\boldsymbol{U}_1^{(k)}$ and $\boldsymbol{U}_2^{(k)}$ are the same point in $\mathcal{G}(k,c)$, that is, if they span the same subspace. Similarly to the Riemannian manifold of the SPD matrices, we can consider geodesic distances on the Grassmannian manifold. A natural way to compare two points in a Grassmannian manifold is to consider the principal angles between $\boldsymbol{U}_1$ and $\boldsymbol{U}_2$ which is a geodesic distance for Grassmannian manifold [189]. Practically, the computation is easily performed using the singular value decomposition of the Gram matrix between $\boldsymbol{U}_1$ and $\boldsymbol{U}_2$:

$$\boldsymbol{U}_1^\top\boldsymbol{U}_2 = \boldsymbol{P}\cos(\boldsymbol{\Theta})\boldsymbol{Q}^\top \tag{3.67}$$

where $\cos(\boldsymbol{\Theta})$ is the diagonal matrix composed of the cosines of principal angles. Thus, a geodesic distance can be derived from these principal angles [67, 86, 186]:

$$\begin{aligned} d^2(\boldsymbol{U}_1,\boldsymbol{U}_2) &= k - \|\boldsymbol{U}_1^\top\boldsymbol{U}_2\|_\mathcal{F}^2 \\ &= \frac{1}{2}\|\boldsymbol{U}_1\boldsymbol{U}_1^\top - \boldsymbol{U}_2\boldsymbol{U}_2^\top\|_\mathcal{F}^2 \end{aligned} \tag{3.68}$$

This geodesic distance is similar to an Euclidean distance between the bilinear pooling of the eigenvectors, but it is done implicitly and with respect to the geometry of the Grassmannian manifold. Wei et al. [186] take advantage of this formulation to implicitly learn a bilinear classifier upon the Grassmannian manifold and outperform most of the state-of-the-art methods with much smaller representation and fewer parameters.

Finally, recent contributions also explore higher-order pooling. However, using higher-order interactions between deep features cannot be implemented in practice due to the dimensionality of higher-order information. Thus, these approaches necessarily rely on factorization strategies that are usually extended from the factorizations from section 3.4. For example, Kernel Pooling [37] explicitly compute the high-order moments in order to approximate a non-linear kernel method. To do so, they rely on the tensor sketch factorization to approximate higher-order moments. These moments are then concatenated into a single representation that is used for classification. Similarly, Cai et al. [19] exploit polynomial kernel in order to approximate higher-order moments and learn a classifier on top of the concatenation of the higher-order moments.

In conclusion, we have presented recent contributions that have gone beyond second-order representations such as Gaussian pooling, hierarchical bilinear pooling or mixture of Gaussian pooling. Also, following the section 3.4 on factorization methods, we have presented a strategy to integrate normalizations with most of the factorization methods. Finally, we have presented recent contributions on higher-order pooling. By aggregating higher-order statistics, these approaches further improve performances of deep networks. Still, high-order pooling is an open problem from many points of view: First, while the geometry of SPD matrices is well studied, the geometry of higher-order moments manifolds is, to the best of our knowledge, not well understood in our context. Second, as most tensor problems are NP-Hard [77], finding normalizations or good reconstructions is not tractable in practice. Finally, higher-order moments estimation in the case of deep learning might also be an issue as the dimensionality of higher-order moments increases exponentially whereas the number of deep features is greatly limited to avoid computational issues.

3.6 Discussion

In this chapter, we have given an overview of the use of second-order statistics in computer vision. After the formulation of covariance pooling, second-order pooling and bilinear pooling, we have discussed the geometry of the symmetric positive definite matrix space. SPD matrices are known for a long time to belong to a Riemannian manifold, that is, a smooth and differentiable manifold. This has led to question the use of Euclidean distance in Riemannian space. Indeed, due to the particular geometry of Riemannian manifolds, the Euclidean distance does not truly represent the space of SPD matrices. To solve this issue, we have presented some geodesic distances such as the Affine-Invariant Riemannian Metric, or distances that are derived from divergences such as the LogDet divergence or the Jeffrey divergence. However, these distances are not tractable for most of the deep learning tasks because the online computation involves eigen-decomposition, matrix inversion, and so on. Thus, the log-Euclidean distance has been proposed to cope with this issue: The matrix logarithm can be performed offline while the online computation of the distance is simply done with the Euclidean distance. Still, the log-Euclidean distance leads to numerical instability when SPD matrices are low-rank. This is solved by the pow-Euclidean distance which is inspired by the log-Euclidean distance but which replaces the matrix logarithm with power-norm. While withdrawing the numerical instability, the pow-Euclidean is not a geodesic distance on the Riemannian manifold: It only "flattens" the Riemannian space in order to make the Euclidean norm more representative for the SPD manifold. However, the normalization is still slow as it is usually performed by computing eigen-decomposition of the SPD matrix. Thus, normalization strategies have been proposed in order to speedup the computation, by considering more stable back-propagation, faster or approximate computation, and so on. From another point of view, the dimensionality of second-order statistics have led to only consider small feature dimension, which limits the usage of recent deep networks that exploit deeper architectures with higher feature dimensions in order to improve performances. Thus, dimensionality reduction techniques are considered in order to have practical and tractable second-order representations. Among the recent methods, factorization techniques have emerged as a simple but efficient way to reduce the dimensionality. By considering a plain linear projection of the second-order representation, factorization methods enforce structural constraints into the projection, by considering low-rank approximation, approximation of kernel or even tensor-based decomposition. Most of these approaches are interesting because they do not compute directly the second-order representation. That is, during the training, we do not have to store very large intermediate representations for the back-propagation. However, such approaches and normalization methods cannot be used together: This greatly limits the performance because only Euclidean distance can be used on factorized representations. Finally, we have presented recent work that have gone beyond second-order representations by considering Gaussian pooling (*i.e.*, both first and second-order representations), mixture of Gaussian pooling, other Riemannian manifold such as the Grassmannian manifold, higher-order pooling, and so on.

We next discuss four out of five contributions at the light of these second-order representations. In chapter 4, we revisit dictionary learning to extend second-order based representations. First, we build a normalization and a two-step dimensionality reduction in order to strengthen and to compress the STA [138] representation named *ISTA*. We show that our improved dictionary-based second-order representation *ISTA*, while being an off-the-shelf representation, outperforms deep networks with trainable first-order representations on top of them. Then, we extend this representation by designing a specific factorization named *JCF* that exploits the dictionary learning of the representation in order to build very compact representations. This factorization makes the representation end-to-end trainable and outperforms most of state-of-the-art methods by taking advantage of both dictionary learning and second-order representations. Finally, we make the link between attention maps and dictionary learning and we present *DIABLO* which gives a framework that generalizes VLAD [91], VLAT [139, 140], STA, *ISTA*, *JCF* and so on. In chapter 5, we present *HORDE*, a regularization strategy for deep metric learning that takes advantage of high-order moments. By extending the RM algorithm and the rank-one factorization from section 3.4, we approximate higher-order moments and use them as complementary representations during the training to regularize the local deep features. By doing so, we show empirically and theoretically that it forces the distribution of deep features that are extracted from images that come from a similar class to be the same while the

distribution for dissimilar classes are forced to be different. *HORDE* highlights another interest of higher-order statistics and the benefits of learning compact representations for them.

Part II

Contributions

4

Dictionary-based tensor aggregation

In this chapter, we present our contributions on image representations based on dictionary learning and on a tensor framework.

Contents

4.1 Prologue **50**
4.2 Improved Spatial Tensor Aggregation **51**
4.2.1 Introduction 51
4.2.2 Spatial tensor aggregation 52
4.2.3 Local feature pre-processing 52
4.2.4 Normalization 54
4.2.5 Factorization 54
4.2.6 Experiments 55
4.2.7 Limitations 56
4.3 Joint Codebook Factorization **57**
4.3.1 Introduction 57
4.3.2 VLAT representation 57
4.3.3 Differential dictionary learning 58
4.3.4 Modified representation 58
4.3.5 Factorization 59
4.3.6 Low-rank approximation 59
4.3.7 Ablation studies 60
4.3.8 Comparison to the state-of-the-art 61
4.3.9 Discussion 61
4.4 Dictionary learning as attention map **62**
4.4.1 Introduction 62
4.4.2 Method overview 62
4.4.3 Attention strategies 63
4.4.4 Feature-wise selection 64
4.4.5 Dimension-wise selection 65
4.4.6 Implementation details 66
4.4.7 Ablation studies 66
4.4.8 Comparison to the state-of-the-art 69
4.5 Global Framework **70**
4.6 Conclusion **71**

4.1 Prologue

Research papers detailed:

- **Pierre Jacob**, David Picard, Aymeric Histace, Edouard Klein. *"DIABLO: Dictionary-based attention block for deep metric learning"*. Pattern Recognition Letters, volume 135, pages 99 – 105, July 2020.
- **Pierre Jacob**, David Picard, Aymeric Histace, Edouard Klein. *"Efficient Codebook and Factorization for Second Order Representation Learning"*. Proceedings of the IEEE International Conference on Image Processing (ICIP), Sept. 2019.
- **Pierre Jacob**, David Picard, Aymeric Histace, Edouard Klein. *"Leveraging Implicit Spatial Information in Global Features for Image Retrieval"*. Proceedings of the IEEE International Conference on Image Processing (ICIP), Oct. 2018.

Context:
We provide in this chapter our contributions on dictionary-based tensor aggregation: (1) dimensionality reduction and normalization for the Spatial Tensor Aggregation [138], (2) differentiable formulation and factorization for end-to-end trainable tensor aggregation and (3) a generalized formulation which links dictionary learning to attention maps.

Aggregation methods have been widely studied in the past few years as a scalable alternative to feature matching in the case of content-based image retrieval tasks. The idea is to build a global representation of the image and to use these representations as proxies for the analysis. This approach has a wide field of applications, such as image retrieval or classification on images or multi-modal tasks such as visual question answering and multi-modal retrieval. Thus, learning rich and compact representations is a very important topic for the computer vision community. To build these representations, methods like Bag-of-Features (BoF) [36], VLAD [91] or Fisher Vectors [134, 150] are few among others that have changed the way of analyzing images for large-scale content-based search. Indeed, one image can be represented by a global representation with a small memory footprint, typically a few kilobytes. Then, the search part is directly done on the representations by computing a similarity measure: with simple similarity measures such as the cosine similarity, the search can be performed in a few milliseconds on image collections with more than a million of images. Even if these methods are efficient when they are used on shallow features such as the SIFT [117], the end-to-end training paradigm of deep learning have made these image representations less efficient than very simple and differentiable representations such as the global average pooling. Yet, these representations can be improved to be differentiable by considering *e.g.*, Radial Basis Function (RBF) for differentiable BoF [131], soft-assignment in VLAD to build NetVLAD [1] or simply by making the parameters learnable in the Fisher Vectors [165]. Thus, current research on global representations are mostly focused on developing global representation with more semantic information, *e.g.* by considering attention maps [98], Local Subspace Projections [47, 107] or dictionary learning [114] among others.

From another point of view, these aggregation methods can be seen as statistical pooling. Indeed, Bag-of-Features is the 0^{th} order moment (count of features), global average pooling and VLAD are 1^{st} order (without and with a dictionary approach), we can consider higher-order statistics, such that second-order,*e.g.*, Bilinear Pooling and Fisher Vectors. The use of higher-order statistics has been shown to be an interesting research direction for the fine-grained analysis of images: using Bilinear Pooling leads to higher results on fine-grained image classification tasks [110] than using first-order statistics (around 15-20% more accuracy on 3 fine-grained datasets). However, even if the increase in performances is unquestionable, second-order strategies suffer from a collection of drawbacks: quadratically increasing dimensionality, costly dimensionality reduction, difficulty to be trained and lack of a proper adapted pooling among others. The two main downsides, namely the high dimensional output representations and the sub-efficient pooling schemes, have been widely studied over the last decade. On the one hand, the dimensionality issue has been studied through factorization scheme, either representation oriented [56, 97] or task oriented [100]. While these factorization schemes are efficient in term of computation cost and number of parameters, the intermediate representation is still very large (typically 10k dimensions) and hinders the training process, while

using lower dimension greatly deteriorates performances. On the other hand, it is well-known that global average pooling schemes aggregate unrelated features. Thus, dictionary-based approaches are excellent alternatives to standard pooling strategies. However, using a dictionary on second-order features leads to an unreasonably large model, since the already large feature has to be duplicated for each dictionary entry. This is the case for example in MFAFVNet [107] for which the second order layer alone (*i.e.*, without the CNN part) costs as much as an entire ResNet50.

In this chapter, we present our contributions in tensor-based aggregation by building compact representations that integrate second-order statistics and dictionary learning. In a first time, we build upon the recently proposed Spatial Tensor Aggregation [138] to improve this off-the-shelf representation by designing normalization and factorization approaches. In a second time, we extend the VLAT representation [139, 140] by slightly modifying the representation, by making it differentiable and by integrating the dimensionality reduction into the representation. The proposed factorization relies on mathematical properties between the dot product and the Kronecker product. By doing so, we show that high-order statistical aggregation can be implicitly computed which decreases both the number of required parameters and the computation cost. Experimentally, our proposed scheme obtains good results on the metric learning tasks where the representation size is strongly constrained. In a third time, we show the link between attention maps and dictionary learning by extending the ABE architecture [98] to dictionary learning. By replacing the optimization objective by a structural constraint in the architecture, we show that ABE can leads to higher performances while being easier to train. Finally, we discuss the link between the differentiable formulation of Spatial Tensor Aggregation and our dictionary version of ABE.

Contributions:
The contributions of this work are to provide new dictionary-based image representations. These representations are evaluated on deep metric learning tasks where the dimensionality of these representations is a strong criterion. After an improvement of the Spatial Tensor Aggregation method, we modify the VLAT representation and we made it differentiable such that we can train the network in an end-to-end manner. Then, the ABE and NetVLAD representations are discussed by considering a framework that links attention-based strategies and dictionary-based ones to build a stronger image representation. Finally, we discuss the links between our propositions and other image representations.

4.2 Improved Spatial Tensor Aggregation

4.2.1 Introduction

In this section, we are interested in image retrieval with a special focus on how to integrate spatial information in the process. Two competing strategies are popular for solving the image retrieval problem. The first one involves computing local descriptors in both the query and the target images and count the number of matching descriptors between the query and the target. A geometric consistency check is then applied to remove matches that are incoherent with the layout of both images (*e.g.*, descriptors that are spatially close in the query should also be spatially close in the target) [88]. As noted in [106], this spatial consistency check is crucial in descriptor matching techniques as it holds a major contribution to their good performance. However, despite efficient descriptors hashing techniques [60, 90], matching based strategies still come at a significant computational and storage cost.

The second strategy addresses these shortcomings by computing a global representation for each image and then using a similarity measure on these representations as a proxy for the visual similarity [91, 139, 150]. Recently, global features have become very competitive when compared to local descriptors matching on challenging datasets [1]. However, global features usually do not allow to perform a geometric consistency check because all spatial information is lost during the computation of the representation, and consequently do not benefit from the associated performance gain.

In this work, we intend to integrate spatial information into global features so as to perform implicit geometric consistency check when computing the similarity between the resulting features. We

propose to integrate this information by modeling correlations between nearby features using a tensor framework. Tensor embeddings have recently become popular for computing visual features [48, 139, 140]. In particular, we build on the ideas proposed in the Spatial Tensor Aggregation (STA) of [138] to propose a new global feature called ISTA. Our contributions are the following:

- We correct all the flaws in the original STA, namely the lack of proper centering, alignment and normalization, leading to the Improved STA (ISTA).
- We propose an adapted two-step dimension reduction method to cope with the high intermediate dimension of the STA.
- Finally, our proposed model is able to obtain state of the art results on well known image retrieval datasets (Holidays, Oxford and Paris).

The remaining of this work is organized as follows: First, we present the STA representation. Then, we develop our Improved STA model for which we show all the corrections as well as the proposed new steps and give intuition about their goals. Next, we present experiments comparing our model to the state of the art on three well known datasets, namely Holidays, Oxford5k and Paris6k.

4.2.2 Spatial tensor aggregation

In this section, we give a reminder of the STA representation presented in section 1.4. The STA representation stands for "Spatial Tensor Aggregation". It has been designed to integrate local spatial information into the representation by considering the correlation of nearby local features. For a given image $\boldsymbol{\mathcal{I}}$ from which local features $\boldsymbol{x} \in \mathbb{R}^c$ are extracted, for a given neighborhood Ω and for an assignment function $\boldsymbol{h}$, the STA representation $\boldsymbol{\mathcal{T}}(\boldsymbol{\mathcal{I}})$ is computed following Equation 1.13 from section 1.4:

$$\boldsymbol{\mathcal{T}}(\boldsymbol{\mathcal{I}}) = \sum_{\boldsymbol{x} \in \boldsymbol{\mathcal{I}}} \sum_{\boldsymbol{x}_u \in \Omega(\boldsymbol{x})} \boldsymbol{h}(\boldsymbol{x}) \otimes \boldsymbol{h}(\boldsymbol{x}_u) \otimes \boldsymbol{x} \otimes \boldsymbol{x}_u \tag{4.1}$$

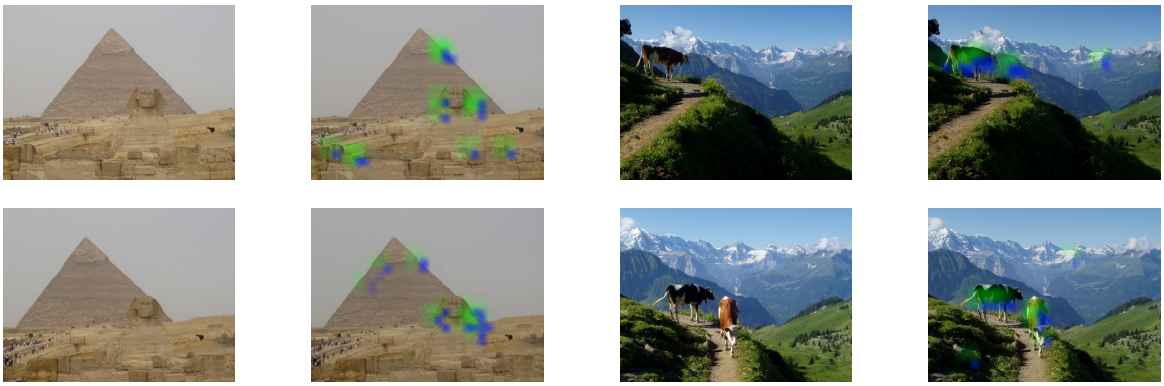
The assignment function $\boldsymbol{h}$ is based on a dictionary $\mathbb{D} = \{\boldsymbol{d}_k\}_k$ of size N: we assign the feature to its nearest dictionary entry and $\boldsymbol{h}(\boldsymbol{x})$ is constructed as a vector with all entries at 0 except the one related to the nearest dictionary which is set to 1. However in practice, this representation is not tractable. Indeed, the representation increases in $\mathcal{O}(N^2c^2)$ which leads to very large representation. For example, using a ResNet50 [74] with $c = 2048$ and a dictionary of size 10 leads to a single representation of size $4{\cdot}1O^8$. On top of requiring 100Go of memory to store 1,000 images, the curse of dimensionality might leads to bad retrieval performances. Also, this formulation does not exploit the improvements realized on the VLAD representation [2], such as signed-square root, intra-projection, centering, *etc.* In the next section, we extend these improvements to the STA representation by taking into account the specific formulation of STA.

4.2.3 Local feature pre-processing

In improved VLAD representation [2, 44], features are processed using the following steps: First, the residual between the feature and its dictionary entry is computed. Then, this residual is projected into the eigenspace of the cluster (using PCA).

In the case of the STA, we propose a similar approach. Given a dictionary of the feature space $\mathbb{D} = \{\boldsymbol{d}_k\}_k$, we first aggregate the residue between (1) a 4-th order tensor computed from a feature and one of its nearby feature and (2) the average of the 4-th order tensors for the relevant pair of dictionary entries. A more convenient form of Equation 1.13 is to define the tensor $\boldsymbol{\mathcal{T}}_{k,l}$ for a given pair of dictionary entries (k, l) using the following equation:

$$\boldsymbol{\mathcal{T}}_{k,l}(\boldsymbol{\mathcal{I}}) = \sum_{\substack{\boldsymbol{x} \in \boldsymbol{\mathcal{I}}_k \\ \boldsymbol{x}_u \in \boldsymbol{\mathcal{V}}_l(\boldsymbol{x})}} \boldsymbol{x}\boldsymbol{x}_u^\top \tag{4.2}$$



original images 1st best pair original images 1st best pair

Figure 4.1: Illustration of the implicit neighborhood matching. For both pairs of similar images, we show the pair of spatially coupled visual codebook entries that contributed the most to the similarity. We color in green the areas corresponding to descriptors belonging to the first codebook entry and in blue their respective neighbors belonging to the second codebook entry.This neighborhood encoding allows to focus on similar regions (*e.g.*, the Sphinx head) with identical local spatial layout of the codebook entries. *(Figure best viewed in color)*

where $\boldsymbol{\mathcal{I}}_k = \left\{ \boldsymbol{x} \in \boldsymbol{\mathcal{I}} \mid \boldsymbol{d}_k = \underset{\boldsymbol{d}_n \in \mathbb{D}}{\arg\min} \ \|\boldsymbol{d}_n - \boldsymbol{x}\|_2^2 \right\}$ and $\boldsymbol{\mathcal{V}}_l(\boldsymbol{x}) = \left\{ \boldsymbol{x}_u \in \Omega(\boldsymbol{x}) \mid \boldsymbol{d}_l = \underset{\boldsymbol{d}_n \in \mathbb{D}}{\arg\min} \ \|\boldsymbol{d}_n - \boldsymbol{x}_u\|_2^2 \right\}$ are respectively the set of local features from image $\boldsymbol{\mathcal{I}}$ that belongs to the k-th dictionary entry and the set of local features from the neighborhood of $\boldsymbol{x}$ that belongs to the l-th dictionary entry. By concatenating these tensors for all ordered pairs of dictionary entries, we recover the original 4th order tensor $\boldsymbol{\mathcal{T}}$.

To perform the centering, we compute the average 4-th order tensor $\boldsymbol{\mathcal{T}}$. With the same observation, we define $\widetilde{\boldsymbol{\mathcal{T}}}_{k,l}$ the partial average tensor for the given pair of dictionary entries (k, l) and estimate it over a set of samples $\mathbb{S}_{k,l}$. $\mathbb{S}_{k,l}$ is the set of pairs $(\boldsymbol{x}, \boldsymbol{x}_u)$ where $\boldsymbol{x}$ belongs to the k-th dictionary entry and $\boldsymbol{x}_u$ belongs to the neighborhood of $\boldsymbol{x}$ and to the l-th dictionary entry:

$$\widetilde{\boldsymbol{\mathcal{T}}}_{k,l} = \frac{1}{|\mathbb{S}_{k,l}|} \sum_{\boldsymbol{x}, \boldsymbol{x}_u \in \mathbb{S}_{k,l}} \boldsymbol{x} \ \boldsymbol{x}_u^\top \tag{4.3}$$

Finally, we are able to define the partial image representation $\boldsymbol{\mathcal{S}}_{k,l}$ based on the aggregation of residues between the tensor $\boldsymbol{\mathcal{T}}_{k,l}$ and its estimation $\widetilde{\boldsymbol{\mathcal{T}}}_{k,l}$:

$$\boldsymbol{\mathcal{S}}_{k,l}(\boldsymbol{\mathcal{I}}) = \sum_{\substack{\boldsymbol{x} \in \boldsymbol{\mathcal{I}}_k \\ \boldsymbol{x}_u \in \boldsymbol{\mathcal{V}}_l}} (\boldsymbol{x}\boldsymbol{x}_u^\top - \widetilde{\boldsymbol{\mathcal{T}}}_{k,l}) \tag{4.4}$$

Since $\boldsymbol{\mathcal{T}}_{k,l}(\boldsymbol{\mathcal{I}})$ is not symmetric, projecting the descriptors into the eigenspace of their cluster pair is not as straightforward as for VLAD. We propose to use the Singular Values Decomposition (SVD) to perform this projection. For a given pair of dictionary entries (k, l), we compute the SVD of $\widetilde{\boldsymbol{\mathcal{T}}}_{k,l}$:

$$\widetilde{\boldsymbol{\mathcal{T}}}_{k,l} = \boldsymbol{U}_{k,l} \boldsymbol{L}_{k,l} \boldsymbol{V}_{k,l}^\top \tag{4.5}$$

We can inject Equation 4.5 into Equation 4.4 by multiplying by $\boldsymbol{U}^\top$ on the left and by multiply by $\boldsymbol{V}$ on the right:

$$\boldsymbol{\mathcal{S}}_{k,l}^{(p)}(\boldsymbol{\mathcal{I}}) = \sum_{\substack{\boldsymbol{x} \in \boldsymbol{\mathcal{I}}_k \\ \boldsymbol{x}_u \in \boldsymbol{\mathcal{V}}_l}} (\boldsymbol{U}_{k,l}^\top \boldsymbol{x}) \ (\boldsymbol{V}_{k,l}^\top \boldsymbol{x}_u)^\top - \boldsymbol{L}_{k,l} \tag{4.6}$$

Using the result from Equation 4.6 has two advantages: On the first hand, each descriptor is projected into the eigenspace which leads to decorrelated components in the resulting feature. On the second hand, only the singular values $\boldsymbol{L}$ are needed for the centering. Furthermore, we can keep only the largest singular values in order to remove irrelevant correlation between nearby descriptors while also reducing the size of the resulting features. In that aspect, we propose an adaptive strategy that keeps a variable number of components for each pair of clusters so as to keep a fixed amount of the explained variance (*e.g.*, 80%). The full signature is the concatenation of all $\boldsymbol{\mathcal{S}}_{k,l}^{(p)}(\boldsymbol{\mathcal{I}})$ for all pairs (k, l).

4.2.4 Normalization

For a dictionary size N, STA computes N^2 pair of dictionary entries. Without a specific normalization, each tensor $\boldsymbol{\mathcal{T}}_{k,l}$ has the same impact on the final representation. To reduce the influence of a potential noisy dictionary entries, we propose to normalize separately the cases $k = l$ from the cases $k \neq l$. The reason behind the proposed approach is that both cases correspond to different contexts in the image. The cases $k = l$ correspond to self-similar regions in the image (*e.g.*, textures), while the cases $k \neq l$ correspond to transitions between different patterns. As such, a dictionary entry can have a negative impact in one context but not in the other.

We propose to normalize in a cross-entry way by considering similarly ranked eigen-components. Since feature pairs have been projected into the eigenspace for each dictionary entry, processing components independently should not lead to any loss of information, with the added benefit of having a signature invariant to the number of features while keeping track of the distribution among dictionary entry pairs.

For a component $s_{k,l}(i, j)$ of $\boldsymbol{\mathcal{S}}_{k,l}^{(p)}$, the normalization is computed as follow:

$$\forall i, j, s_{k,l}(i,j) = \begin{cases} \dfrac{s_{k,l}(i,j)}{\sqrt{\sum_n^N s_{n,n}^2(i,j)}} \text{ for } k = l \\ \dfrac{s_{k,l}(i,j)}{\sqrt{\sum_{n\neq m}^{N,N} s_{n,m}^2(i,j)}}, \text{ else} \end{cases} \tag{4.7}$$

This normalization is computed after a power normalization ($\alpha = 0.5$) and is followed by a l_2 normalization on the full image representation.

4.2.5 Factorization

After the centering and normalization, the resulting features have a dimension in the order of N^2D^2 where D is the number of component retained in the preprocessing. This usually leads to intermediate features of size close to 10^6, which we now propose to reduce. Dimension reduction is usually achieved using PCA on a set of representations by keeping components associated with the largest eigenvalues. In our case, PCA is tractable neither with the classic approach nor with the Gram matrix decomposition due to the high dimension of the aggregated features. To cope with such a high dimension, we propose a two step reduction scheme. First, we propose to perform a sparse reduction that considers components related to a specific dictionary entry pair. We call this step *Block Reduction* due to the block diagonal structure of the resulting projection matrix. Then, we perform a full projection on the resulting vectors.

The block reduction is performed independently for each $\boldsymbol{\mathcal{S}}_{k,l}^{(p)}$ of the normalized feature. This projection should map the feature block in a target space with fewer dimension while retaining the inner product. Indeed, since the inner product on the full signature is the sum of the inner products over all blocks $\boldsymbol{\mathcal{S}}_{k,l}^{(p)}$, retaining the inner product on blocks is a sufficient condition to retain the full similarity. This is achieved by performing a low-rank approximation of a Gram matrix $\boldsymbol{G}_{k,l}$ computed on a large set of sampled blocks.

Method	Holidays	Oxford	Paris
Crow [92]	85.1	70.8	79.7
NetVLAD [1]	88.3	69.1	78.5
SLEM [142]	91.7	71.7	-
STA [138]	72.3	-	-
ISTA (VGG16)	91.7	71.6	82.2
ISTA (MobileNet)	**94.4**	**77.1**	**88.8**

Table 4.1: Comparison with the state-of-the-art in mAP.

In practice, we compute for each pair (k, l) the projection matrix $\boldsymbol{P}_{k,l}$ using the SVD of a large set of blocks $\boldsymbol{\mathcal{S}}_{k,l}^{(p)}$ and retaining a number of components proportional to the original number of dimension using the following equations:

$$\begin{aligned} \forall (i,j), \boldsymbol{G}_{k,l}(i,j) &= \langle \boldsymbol{\mathcal{S}}_{k,l}(\boldsymbol{\mathcal{I}}) \ ; \ \boldsymbol{\mathcal{S}}_{k,l}(\boldsymbol{\mathcal{J}}) \rangle \\ \boldsymbol{G}_{k,l} &= \boldsymbol{V}_{k,l} \boldsymbol{L}_{k,l} \boldsymbol{V}_{k,l}^{\top} \\ \boldsymbol{P}_{k,l} &= \boldsymbol{L}_{k,l}^{\frac{1}{2}} \boldsymbol{V}_{k,l}^{\top} {\boldsymbol{\mathcal{S}}_{k,l}^{(p)}}^{\top} \end{aligned} \tag{4.8}$$

Each of these block-wise projection $\boldsymbol{P}_{k,l}$ is then zero padded to match the full size of the input features and the padded projections are concatenated. This result in a single sparse projection matrix $\boldsymbol{P} = [\boldsymbol{P}_{k,l}]_{k,l}$ with a block diagonal structure corresponding to the different pairs of dictionary entries. Once the sparse projection is done, we perform the second reduction on the resulting features. Similarly to the first step, we aim at retaining the original inner product, which can be performed by finding a low-rank approximation of the Gram matrix using the SVD of a large set of features obtained by the sparse projection. As shown in [87], performing a whitening (*i.e.*, equalizing the variance in the target space) at this stage can lead to a significant improvement, which is what we also observe. Remark that performing a whitening at this stage is only possible if no whitening was done during the block reduction stage as it would lead to all correlation between blocks being of equal importance.

4.2.6 Experiments

In this section, we compare our proposed ISTA approach with recent approaches in the literature. We start by giving technical details and present the datasets on which the evaluation is performed before we comment on the results.

4.2.6.1 Experiments pipeline

As features extractor, we use the following off-the-shelf CNNs: VGG16 [154] (cut after block5) and MobileNet [78] (cut after block 11) both pre-trained on ImageNet [45]. All parameters are computed on 20k images taken from Places 365 [204] validation set with a dictionary size of 32. In all our experiments, raw dimension is 670k, which corresponds to keeping 80% of variance for VGG16 and 95% for MobileNet in the preprocessing. The block projections are computed using 8192 images from Flickr100k and we kept 40% of the original dimension. The full reduction is computed on 25k images from Flickr100k to reduce the final dimension to 22k. We compute the representations at 2 image resolutions (512px and 1024px) while conserving the image ratio and we sum the two representations.

We test our model with 3 image retrieval datasets: INRIA Holidays [88] (1941 images, 500 queries), Oxford5k [136] (5062 images, 55 queries) and Paris6k [137] (6412 images, 55 queries). We report the mean average precision (mAP) for each datasets while using the full image as query.

4.2.6.2 Results

In this part, we compare ISTA on Holidays, Oxford5k and Paris6k against the state-of-the-art in Table Table 4.1. Results with VGG16 as features extractor are similar to others methods that performs fine-tuning on the whole architecture. However, these features were harder to compress than those extracted with MobileNet: +3% mAP on Holidays and +5.5% mAP on Oxford. We are able to outperform state-of-the-art representations on Holidays with 94.4% of mAP versus 91.7% mAP for NetVLAD with polynomial-kernel SLEM. On Oxford5k (resp. Paris6k), ISTA also outperforms the state-of-the-art obtained by the same methods by more than 5% (resp. 10%). Remark that most of the methods reported in Table 4.1 fine-tunes all parameters, whereas we do not perform it for ISTA.

As an illustration, we show on Figure 4.2 two queries that where among the most improved by ISTA over NetVLAD. For a given query, we show the associated top 5 ranked images for both methods, as well as masks of the regions that contributed the most to the ranking. As we can see, NetVLAD focuses on regions that may look similar taken independently from their context (like the sky in the first query), but that are not very distinctive. In contrast, our ISTA method focuses on strongly structured patterns that are much more distinctive.

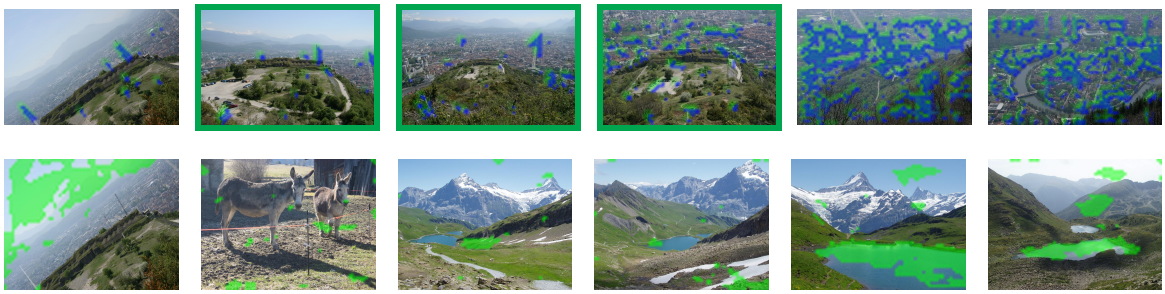


Figure 4.2: Sample of queries improved by performing implicit spatial consistency check. Queries are on the first column, while the top 5 results retrieved by ISTA (first row) and NetVLAD (second row) respectively. Relevant images in the top 5 are outlined in green. Color masks in the images depict the region that contributed the most to the similarity. *(Figure best viewed in color)*

4.2.7 Limitations

In this part, we discuss the limitations of the *ISTA* representation. The most limiting part of the method is that it is not differentiable. Indeed, due to the hard assignment of the dictionary, we cannot train in an end-to-end manner the network in order to fully take advantage of the deep learning backbone network. Also, the multi-step factorizations are not designed to be discriminant which means that encoded information might be irrelevant for the task of image retrieval. Thus, the *ISTA* representation might be improved by considering *semantic factorization*, *i.e*, dimensionality reduction method that only encode the relevant semantic information of the representation to perform the retrieval task.

To make the dictionary part trainable, we present in the next section a way to compute a differentiable dictionary assignment which is inspired from NetVLAD [1]. By slightly modifying the VLAT representation [139], we can made the representation end-to-end trainable in order to train the backbone network, the dictionary and the representation together. However, the image representation has a too large dimensionality to be trained end-to-end without proper factorization. Thus, the second contribution of the following section is a two-steps factorization that exploits a property between the Kronecker product and the dot product so as to integrates the factorization within the representation. Moreover, the following differentiable representation is made such that the computation of the

fourth order tensor is implicit: This means that the proposed representation leverages the dictionary learning and the second-order pooling but without the memory footprint and dimensionality of VLAT. Finally, we discuss the links between state-of-the-art tensor-based aggregation methods and we show how to make the STA representation differentiable by slightly modifying the representation from the next section.

4.3 Joint Codebook Factorization

4.3.1 Introduction

In this section, we propose a differentiable representation, inspired from VLAT [139], that leverages dictionary learning and second-order pooling. To do so, we rethink the global representation to exploit the symmetry in the representation and a property between the dot product and the Kronecker product. The proposed factorization scheme greatly reduces the computation of the image representation by not directly computing the fourth-order tensor but still implicitly encodes the information from the dictionary learning and the second-order pooling. Our main results are the following:

- We first show that state-of-the-art factorization schemes can be improved by the use of a codebook pooling, albeit at a prohibitive cost.
- We then propose our main contribution, a joint codebook and factorization scheme that achieves higher results at a much reduced cost.

Since our approach focuses on representation learning and is task agnostic, we validate it in a retrieval context on several image datasets to show the relevance of the learned representations. We show that our model achieves competitive results on these datasets at a very reasonable cost.

The remaining of this section is organized as follows: First, we present our factorization scheme with the dictionary strategy and how we improve its integration. Second, we run ablation studies on the Stanford Online Products dataset [127]. Finally, we compare our approach to the state-of-the-art methods on three image retrieval datasets (Stanford Online Products, CUB-200-2011, Cars-196).

4.3.2 VLAT representation

In this section, we give a reminder of the VLAT representation presented in section 1.4. Vectors of Locally Aggregated Tensors (VLAT) [139, 140] is another image representation that combines dictionary learning and second-order pooling. This representation is the direct extension of VLAD by considering the covariance matrices of the deep local features instead of the features themselves. For a given dictionary $\mathbb{D} = \{\boldsymbol{d}_k\}_k$ and deep features $\boldsymbol{x}$ extracted from image $\boldsymbol{\mathcal{I}}$, the VLAT representation $\boldsymbol{T}_k$ for the dictionary entry k is computed using the following equation:

$$\boldsymbol{T}_k = \frac{1}{|\boldsymbol{\mathcal{I}}_k|} \sum_{\boldsymbol{x} \in \boldsymbol{\mathcal{I}}_k} (\boldsymbol{x} - \boldsymbol{\mu}_k) \otimes (\boldsymbol{x} - \boldsymbol{\mu}_k) - \widehat{\boldsymbol{T}}_k \tag{4.9}$$

where $\boldsymbol{\mathcal{I}}_k = \left\{ \boldsymbol{x} \in \boldsymbol{\mathcal{I}} \mid \boldsymbol{d}_k = \underset{\boldsymbol{d}_n \in \mathbb{D}}{\arg\min} \|\boldsymbol{d}_n - \boldsymbol{x}\|_2^2 \right\}$ is the set of local features from image $\boldsymbol{\mathcal{I}}$ that belong to the k-th dictionary entry, $\boldsymbol{\mu}_k$ is the average feature from the k-th dictionary entry and $\widehat{\boldsymbol{T}}_k$ is the average tensor for the k-th dictionary entry.

This VLAT representation is also linked to the Fisher Vectors and to the Spatial Tensor Aggregation. Indeed, by considering the covariance matrices of deep features with a hard assignment function for the dictionary part, Fisher Vectors can be seen as a relaxation of the assignment by considering Gaussian Mixture Models. Also, VLAT can be viewed as special case of Spatial Tensor Aggregation if we consider an empty neighborhood.

Similarly to VLAD, the assignment function in VLAT can be modified to make this image representation differentiable. However, the dimensionality of the VLAT representation, which is in $\mathcal{O}(Nc^2)$

with N the dictionary size and c the feature dimension, limits the use of this representation during the training. Indeed, the memory usage to store the representations and their gradients is much larger than standard representations which limits the dimensionality of the features and the dictionary size. Thus, in addition to the differentiable assignment function, we develop a specific factorization scheme to reduce the dimensionality of this representation. This factorization fully takes advantage of the dictionary-based aggregation compared to generic approaches. Also, it is designed to avoid the explicit computation of the covariance matrices: this strongly reduces the computation cost and the memory storage required during the training.

4.3.3 Differential dictionary learning

To make the full representation end-to-end trainable, we need to modify the assignment function $\boldsymbol{h}$ to be differentiable. As a reminder, the function $\boldsymbol{h}$ is a one-hot encoder function defined over a dictionary $\mathbb{D} = \{\boldsymbol{d}_k\}_k$ which gives a vector full of 0's with only one entry with a 1 at position k if the corresponding feature belongs to the k-th dictionary entry $\boldsymbol{d}_k$:

$$h_k(\boldsymbol{x}) = \begin{cases} 1 & \text{if } \boldsymbol{d}_k = \underset{\boldsymbol{d}_l \in \mathbb{D}}{\arg\min} \ \|\boldsymbol{d}_l - \boldsymbol{x}\|_2^2 \\ 0 & \text{else.} \end{cases} \tag{4.10}$$

To make this formulation differentiable, we replace the arg min operator by the soft max one:

$$h_k(\boldsymbol{x}) = \frac{\mathrm{e}^{-\alpha \ \|\boldsymbol{x}-\boldsymbol{d}_k\|_2^2}}{\sum_l \mathrm{e}^{-\alpha \ \|\boldsymbol{x}-\boldsymbol{d}_l\|_2^2}} \tag{4.11}$$

where α is a hyper-parameter to control the hardness of the assignment. This strategy is also exploited in NetVLAD [1] in the same way to made the VLAD representation differentiable. However, in NetVLAD, the authors do not design a specific dimensionality reduction procedure because the dimensionality of the representation is tractable in practice with a simple linear projection. In the case of VLAT, due to the covariance matrices, we cannot simply add a linear projection at the end of the representation without strongly reducing performances of the representation and strongly increasing the number of parameters. Thus, in a first time, we revisit the VLAT representation in subsection 4.3.4 to ease the factorization part. Then, in a second time, we design the factorization by exploiting the property of the revisited representation but also by exploiting the property between the dot product and the Kronecker product.

4.3.4 Modified representation

In this work, we start with a modified VLAT representation. In this representation, we have replaced the covariance matrices by the second-order moment (*i.e.*, we have simply removed the centering of the deep features) and we have also removed the centering with the average tensor. In our earlier experiments, these elements that have improved performances in the shallow representation have little impact on the performance when it is trained end-to-end. Also, we duplicate the dictionary assignment function to keep the generalization of bilinear pooling (two dictionaries can be learned, one per modality) or for STA based strategies (two nearby features may belong to different dictionary entries). Then, the raw representation can be expressed as follows:

$$\boldsymbol{\mathcal{T}}(\boldsymbol{\mathcal{I}}) = \sum_{\boldsymbol{x} \in \boldsymbol{\mathcal{I}}} \boldsymbol{h}(\boldsymbol{x}) \otimes \boldsymbol{h}(\boldsymbol{x}) \otimes \boldsymbol{x} \otimes \boldsymbol{x} \tag{4.12}$$

where $\boldsymbol{h}$ is the assignment function to the dictionary. To reduce the dimensionality of the representation, we want to find the optimal linear projection $\boldsymbol{W} \in \mathbb{R}^{N^2c^2 \times D}$ where c is the dimensionality of the local deep features, N is the dictionary size and D is the size of the representation after dimensionality reduction. This projection matrix $\boldsymbol{W}$ cannot directly be used because of the large number of parameters induced by the representation. Thus, we propose a dedicated factorization scheme to reduce the number of parameters and the computation time by exploiting the property between the dot product and the Kronecker product.

4.3.5 Factorization

Following subsection 4.3.4, for a given feature $\boldsymbol{x}$ extracted from the image $\boldsymbol{\mathcal{I}}$, we project it to build the output feature $\boldsymbol{z}(\boldsymbol{x}) \in \mathbb{R}^D$. These output features are then pooled to build the output representation $\boldsymbol{z}$:

$$\boldsymbol{z} = \sum_{\boldsymbol{x}} \boldsymbol{z}(\boldsymbol{x}) = \sum_{\boldsymbol{x}} \boldsymbol{W}^T (\boldsymbol{h}(\boldsymbol{x}) \otimes \boldsymbol{x} \otimes \boldsymbol{h}(\boldsymbol{x}) \otimes \boldsymbol{x}) \tag{4.13}$$

In the following, we use the notation z_i that refers to the i-th dimension of the output representation $\boldsymbol{z}$ and $z_i(\boldsymbol{x})$ the i-th dimension of the output feature $\boldsymbol{z}(\boldsymbol{x})$, that is:

$$z_i(\boldsymbol{x}) = \langle \boldsymbol{w}_i \ ; \ \boldsymbol{h}(\boldsymbol{x}) \otimes \boldsymbol{x} \otimes \boldsymbol{h}(\boldsymbol{x}) \otimes \boldsymbol{x} \rangle \tag{4.14}$$

where $\boldsymbol{w}_i \in \mathbb{R}^{N^2 d^2}$ is the i-th column of $\boldsymbol{W}$. Due to the large number of parameters induced by this projection matrix, we enforce the rank-one decomposition $\boldsymbol{w}_i = \boldsymbol{p}_i \otimes \boldsymbol{q}_i$ where $(\boldsymbol{p}_i, \boldsymbol{q}_i) \in (\mathbb{R}^{Nd})^2$ to split the modalities. This first factorization leads to the following output feature $z_i(\boldsymbol{x})$:

$$z_i(\boldsymbol{x}) = \langle \boldsymbol{p}_i \ ; \ \boldsymbol{h}(\boldsymbol{x}) \otimes \boldsymbol{x} \rangle \, \langle \boldsymbol{q}_i \ ; \ \boldsymbol{h}(\boldsymbol{x}) \otimes \boldsymbol{x} \rangle \tag{4.15}$$

However, this representation is still too large to be computed directly. Then, we enforce two supplementary factorizations:

$$\boldsymbol{p}_i = \sum_j \boldsymbol{e}^{(j)} \otimes \boldsymbol{u}_{i,j} \tag{4.16}$$

$$\boldsymbol{q}_i = \sum_j \boldsymbol{e}^{(j)} \otimes \boldsymbol{v}_{i,j} \tag{4.17}$$

where $\boldsymbol{e}^{(j)} \in \mathbb{R}^N$ is the j-th vector from the natural basis of $\mathbb{R}^N$ and $(\boldsymbol{u}_{i,j}, \boldsymbol{v}_{i,j}) \in (\mathbb{R}^d)^2$. The decompositions of $\boldsymbol{p}_i$ and $\boldsymbol{q}_i$ play the same roles as intra-projection in VLAD [44]. Indeed, if we consider $\boldsymbol{h}(\cdot)$ as a hard assignment function, the only computed projection is the one assigned to the corresponding dictionary entry. Thus, this model learns a projection matrix for each dictionary entry.

Furthermore, by exploiting the same property used Equation 4.15, the following equation can be compacted such as:

$$z_i(\boldsymbol{x}) = \left(\boldsymbol{h}(\boldsymbol{x})^T \boldsymbol{U}_i^T \boldsymbol{x}\right) \left(\boldsymbol{h}(\boldsymbol{x})^T \boldsymbol{V}_i^T \boldsymbol{x}\right) \tag{4.18}$$

where $\boldsymbol{U}_i \in \mathbb{R}^{d \times N}$ and $\boldsymbol{V}_i \in \mathbb{R}^{d \times N}$ are the matrices concatenating the projections of all entries of the dictionary for the i-th output dimension. We call it *Joint Codebook and Factorization*, *JCF*-N.

To summarize, this factorization integrates dictionary learning and second-order pooling in a simple and efficient way. It is simple because the left and right parts of Equation 4.18 are easily implemented with 1×1 convolutions and broadcasted matrix multiplications. Then, an element-wise product is used to compute the output representation. It is efficient because the fourth-order tensor is never directly computed which greatly reduce the memory footprint and the computation cost. Still, the number of parameters is limiting as it increases in $\mathcal{O}(dDN)$. As such, using large dictionaries may become intractable.

4.3.6 Low-rank approximation

In the previous factorization, one projector is learned to map all features that belong to a given codebook entry for each entry of the codebook. The proposed idea is, instead of using a one-to-one correspondence, we learn a set of projectors that is shared across the codebook. The reasoning behind is that projectors from different codebook entries are unlikely to be all orthogonal. By doing such hypothesis (*i.e.*, the vector space spaned by the combination of all the projection matrices has a lower dimension than the codebook itself), we can have smaller models with nearly no loss in performances. To check this hypothesis, we extend the proposed factorization from Equation 4.18.

We want to generate $\boldsymbol{U}_i$ from $\{\widetilde{\boldsymbol{U}}_i\}_{i\in\{1,\dots,R\}}$ and $\boldsymbol{V}_i$ from $\{\widetilde{\boldsymbol{V}}_i\}_{i\in\{1,\dots,R\}}$ where R is the number of projections in the set. Then the two new enforced factorization of $\boldsymbol{p}_i$ and $\boldsymbol{q}_i$ are:

$$\boldsymbol{p}_i(\boldsymbol{x}) = \sum_r f_{p,r}\Big(\boldsymbol{h}(\boldsymbol{x})\Big)\boldsymbol{e}^{(r)} \otimes \widetilde{\boldsymbol{u}}_{i,r} \quad \text{and} \tag{4.19}$$

$$\boldsymbol{q}_i(\boldsymbol{x}) = \sum_r f_{q,r}\Big(\boldsymbol{h}(\boldsymbol{x})\Big)\boldsymbol{e}^{(r)} \otimes \widetilde{\boldsymbol{v}}_{i,r} \tag{4.20}$$

where $\boldsymbol{f}_p$ and $\boldsymbol{f}_q$ are two functions from $\mathbb{R}^N$ to $\mathbb{R}^R$ that transform the codebook assignment into a set of coefficient which generate their respective projection matrices. Similarly to Equation 4.18, we have:

$$z_i(\boldsymbol{x}) = \left(\boldsymbol{f}_p\Big(\boldsymbol{h}(\boldsymbol{x})\Big)^T \widetilde{\boldsymbol{U}}_i^T\boldsymbol{x}\right)\left(\boldsymbol{f}_q\Big(\boldsymbol{h}(\boldsymbol{x})\Big)^T \widetilde{\boldsymbol{V}}_i^T\boldsymbol{x}\right) \tag{4.21}$$

In this paper, we only study the case of a linear projection:

$$z_i(\boldsymbol{x}) = \left(\boldsymbol{h}(\boldsymbol{x})^T\boldsymbol{A}\widetilde{\boldsymbol{U}}_i^T\boldsymbol{x}\right)\left(\boldsymbol{h}(\boldsymbol{x})^T\boldsymbol{B}\widetilde{\boldsymbol{V}}_i^T\boldsymbol{x}\right) \tag{4.22}$$

where $(\boldsymbol{A},\boldsymbol{B}) \in (\mathbb{R}^{N\times R})^2$. Equation 4.22 is more efficient in terms of parameters than Equation 4.18 as it requires $\frac{R}{N}$ times lesser parameters and computation. We call this approach *JCF*-N-R.

4.3.7 Ablation studies

4.3.7.1 Bilinear pooling and codebook strategy

In this section, we demonstrate both the relevance of second-order information for retrieval tasks and the influence of the codebook on our method. We report recall@1 on Stanford Online Products in Table 4.2 for the different configuration detailed below. First, as a reference, we train a *Baseline* network, *i.e.*, which consists in the average of the features reduced to 512 dimensions (first order model). Then we implement bilinear pooling and extend it naively to a codebook strategy. The objective is to demonstrate that such strategy performs well, but at an intractable cost. Results are reported in the left part of Table 4.2. This experiment confirms the interest of second-order information in image retrieval with a improvement of 2% over the baseline, while using a 512 dimension representation. Furthermore, using a codebook strategy with few codewords enhances bilinear pooling by 1% more. However, the number of parameters becomes intractable for codebook of size greater than 4: this naive strategy requires 270M parameters to extend this model to a codebook with a size of 8.

Using the factorization from Equation 4.18 greatly reduces the required number of parameters and allows the exploration of larger codebook. In the case of the factorization alone, the small representation dimension leads to poor performances and are only slightly retrieved using a codebook. On the opposite, our factorization which exploits both the larger codebook and the low-rank approximation is able to reach higher performances (+4% between BP and *JCF*-32-32) with nearly four times less parameters.

4.3.7.2 Sharing projections

In this part, we study the impact of the sharing projection. We use the same training procedure as in the previous section. For each codebook size, we train architecture with a different number of projections R, allowing to compare architectures without the sharing process to architectures with greater codebook size but with the same number of parameters by sharing projectors. We compare the full *JCF*-N with $N \in \{4, 16, 32\}$ using Equation 4.18 to *JCF*-N-R with $R \in \{4, 8, 16\}$ using Equation 4.22. Results are reported in the right part of Table 4.2. Sharing projectors leads to smaller models with few loss in performances, and using richer dictionary allows more compression with superior results. In the next section, we compare our best model *JCF*-32-32 and its shared version with four times less parameters *JCF*-32-8 against state-of-the-art methods.

Method	*Baseline*	BP		*JCF*-N-R								
N	-	-	4	-	4	16			32			
R	-	-	-	-	-	4	8	16	4	8	16	32
Parameters	1M	34M	135M	0.8M	1.6M	1.6M	2.6M	4.7M	1.6M	2.6M	4.7M	8.9M
R@1	63.8	65.9	67.1	65.0	65.5	**68.2**	**68.3**	**69.8**	**68.1**	**69.4**	**69.7**	**70.6**

Table 4.2: Comparison of codebook strategy in terms of parameters and performances between the *Baseline*, BP and our *JCF* for different codebook size (N) and low-rank approximation (R) on Stanford Online Products.

r@	1	10	100	1000
LiftedStruct [127]	62.1	79.8	91.3	97.4
Binomial deviance [173]	65.5	82.3	92.3	97.6
N-pair loss [158]	67.7	83.8	93.0	97.8
HDC [197]	69.5	84.4	92.8	97.7
Margin [190]	72.7	86.2	93.8	98.0
BIER [129]	72.7	86.5	94.0	98.0
Proxy-NCA [122]	73.7	-	-	-
HTL [57]	74.8	88.3	94.8	98.4
JCF-32	**77.4**	89.9	**95.8**	98.6
JCF-32-8	76.6	**90.0**	**95.8**	**98.7**

Table 4.3: Comparison with the state-of-the-art on Stanford Online Products dataset. **bold** scores are the current state-of-the-art and underlined are the second ones.

4.3.8 Comparison to the state-of-the-art

In this section, we compare our method to the state-of-the-art on 3 retrieval datasets: Stanford Online Products [127], CUB-200-2011 [174] and Cars-196 [102]. For Stanford Online Products and CUB-200-2011, we use the same train/test split as [127]. For Cars-196, we use the same as [129]. We report the standard recall@K with $K \in \{1, 10, 100, 1000\}$ for Stanford Online Products and with $K \in \{1, 2, 4, 8\}$ for the other two in Table 4.3 and Table 4.4 respectively. We implement the codebook factorization from Equation 4.18 with a codebook size of 32 (denoted *JCF*-32). At the time of submission of this work, *JCF*-32 outperforms state-of-the-art methods on the three datasets. Our low-rank approximation *JCF*-32-8, which cost 4 times less also leads to state-of-the-art performances on two of them with a loss between 1-2% consistent with our ablation studies. In the case of Cars-196 however, performances are much more lower than the full model. We argue that the variety introduced by the colors, the shapes, *etc.* in cars requires more projections to be estimated, as it is observed for the full model.

4.3.9 Discussion

In this section, we discuss about the low-rank factorization of *JCF* in light of the chapter 3. The low-rank formulation $\boldsymbol{A}\widetilde{\boldsymbol{U}}_i^T$ in Equation 4.22 is a way to approximate the full matrix $\boldsymbol{U}_i^\top$ using the multi-rank approximation of section 3.4. Indeed, it can be re-written as follows:

$$\boldsymbol{U}_i^\top \approx \boldsymbol{A}\widetilde{\boldsymbol{U}}_i^\top = \sum_{r=1}^{R} \boldsymbol{a}_r \widetilde{\boldsymbol{u}}_{i,r}^\top \tag{4.23}$$

which is a muti-rank approximation of the matrix with a rank of R with $\boldsymbol{a}_r$ the r-th column of $\boldsymbol{A}$ and $\widetilde{\boldsymbol{u}}_{i,r}$ the r-th column of $\widetilde{\boldsymbol{U}}_i$. Interestingly, this factorization is one of the simplest one that leads to a small approximation error where parameters can be trained end-to-end. However, the required number of parameters has a cubic growth in $\mathcal{O}(dDR)$ where d is the size of the deep features, D is the size of the output representation and R is the rank of the approximation. Also, the parameter gain is in $\frac{N}{R}$ where N is the dictionary size which cannot save a large amount of parameters. Thus, in light

	CUB-200-2011				Cars-196			
r@	1	2	4	8	1	2	4	8
[173]	52.8	64.4	74.7	83.9	-	-	-	-
[158]	51.0	63.3	74.3	83.2	71.1	79.7	86.5	91.6
[197]	53.6	65.7	77.0	85.6	73.7	83.2	89.5	93.8
[129]	55.3	67.2	76.9	85.1	78.0	85.8	91.1	95.1
[57]	57.1	68.8	78.7	86.5	81.4	88.0	92.7	95.7
JCF-32	**60.1**	**72.1**	**81.7**	**88.3**	**82.6**	**89.2**	**93.5**	**96.0**
JCF-32-8	58.1	70.4	80.3	87.6	74.2	83.4	89.7	93.9

Table 4.4: Comparison with the state-of-the-art on CUB-200-2011 and Cars-196 datasets. **bold** scores are the current state-of-the-art and underlined are second.

of the factorization presented in chapter 3, one can also consider the shared multi-rank approach. Instead of having a low rank factorization of the matrix $\boldsymbol{U}_i$ for each output dimension i, the idea is to have a larger set of projectors that is shared across all dimensions to approximate all matrices $\boldsymbol{U}_i$. By doing so, the number of parameters has still a cubic growth in $\mathcal{O}(RND)$, but the parameter gain is in $\frac{d}{R}$. Also, note that in the case of *JCF* the value of R is in $\mathcal{O}(N)$ because we want to approximate N projectors with only R. In the case of the shared multi-rank approach, having more than d projectors is not necessary because we want to approximate projectors in $\mathbb{R}^d$ which means that the value of R is in $\mathcal{O}(d)$. Finally, one can consider to extend *JCF* to high-order moments. We detail in depth this extension in section 5.5 as long as the use of *JCF* in *HORDE*.

4.4 Dictionary learning as attention map

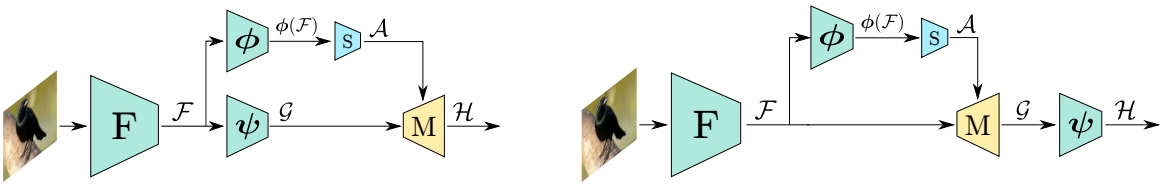
4.4.1 Introduction

In this section, we make the link between attention maps to build stronger image representations as it is done in ABE [98] and dictionary learning. By doing so, we show that the ABE representation and NetVLAD are very similar in their approaches. Thus, we study in details the differences in the strategies of ABE and NetVLAD in order to understand their strength. In order to do so, we evaluate attention strategies named *pre-attention* and *post-attention* in the deep metric learning framework, together with two selection strategies named *feature-wise* and *dimension-wise* selection. This study leads to the *DIABLO* method, a DIctionary-based Attention BLOck method that produces robust image representations by taking advantage of both ABE and NetVLAD benefits. We show in practice that *DIABLO* consistently improves the state-of-the-art on four deep metric learning datasets (Cub-200-2011, Cars-196, Stanford Online Products and In-Shop Clothes Retrieval).

The remaining of this section is organized as follows: First, we give an overview of the proposed architecture, then we detail the attention strategies and the feature-wise and dimension-wise selection methods. Second, we show ablation studies on the selection and attention strategies, the dictionary size and the comparison to ABE. Third, we compare our approach to the state-of-the-art methods on four image retrieval datasets (Cub-200-2011, Cars-196, Stanford Online Products and In-Shop Clothes Retrieval). Finally, we discuss our previous contributions in light of *DIABLO*.

4.4.2 Method overview

We give an overview of the method illustrated in Figure 4.3. We start by extracting a deep feature map $\boldsymbol{\mathcal{F}} \in \mathbb{R}^{h \times w \times c}$ using the local feature extractor F where h and w are the height and width of the feature map and c is the deep feature dimension. We further process the extracted feature map using a non-linear function $\boldsymbol{\phi}$ implemented by a convolutional neural network (CNN). In order to compute the N attention maps $\boldsymbol{\mathcal{A}}$ where N is the dictionary size, we pass the feature map into the selection block S, that is either the feature-wise selection (subsection 4.4.4) or the dimension-wise selection (subsection 4.4.5).



(a) Post-attention strategy

(b) Pre-attention strategy

Figure 4.3: Illustration of post and pre-attention strategies. In pre-attention, we first compute the attention maps and then we process each of them. In post-attention, the feature map is further processed before computing the attention maps. In pre-attention, we first compute the attention maps and then we process each of them. F is a feature extractor and $\boldsymbol{\phi}$ and $\boldsymbol{\psi}$ are two non-linear transformation of the deep features. The blocks $\mathcal{S}$ correspond to either Equation 4.30 or Equation 4.34. The blocks $\mathcal{M}$ are illustrated in Figure 4.4.

In the post-attention setup (Figure 4.3a), the feature map $\mathcal{F}$ is further processed with a non-linear function $\boldsymbol{\psi}$ (implemented by a CNN) and transformed into a feature map $\mathcal{G}$. It is then combined with the attention maps $\boldsymbol{\mathcal{A}}$ using the block M to produce the N new feature maps. In the pre-attention setup (Figure 4.3b), the feature map $\mathcal{F}$ is directly combined with the attention maps $\boldsymbol{\mathcal{A}}$ using the block M. In this case, we further process these N feature maps using a non-linear function $\boldsymbol{\psi}$. The different blocks M used to combine the attention maps and the original feature map are illustrated in Figure 4.4 and detailed latter in subsection 4.4.4 and subsection 4.4.5.

These N feature maps are pooled using a global average pooling with adding an embedding layer for each branch. So, the output representation is obtained by concatenating these N branches.

4.4.3 Attention strategies

This section focuses on computing a set of attention maps $\boldsymbol{\mathcal{A}}$ using one of the selection blocks S and its corresponding dictionary $\boldsymbol{\mathcal{D}}$ as well as the feature map $\boldsymbol{\mathcal{F}} \in \mathbb{R}^{h \times w \times c}$. For this purpose, we process the feature map $\boldsymbol{\mathcal{F}}$ with a non-linear function $\boldsymbol{\phi} : \mathbb{R}^c \rightarrow \mathbb{R}^m$ implemented by a CNN. Then, the set of attention maps are computed by a selection block S such that $\boldsymbol{\mathcal{A}} = \mathrm{S}(\boldsymbol{\phi}(\boldsymbol{\mathcal{F}}); \boldsymbol{\mathcal{D}})$. $\boldsymbol{\mathcal{A}}$ is used with one of the following attention strategies.

4.4.3.1 Post-attention

In the post-attention strategy illustrated in Figure 4.3a, we process the feature map $\boldsymbol{\mathcal{F}}$ into an intermediate feature map $\boldsymbol{\mathcal{G}}$:

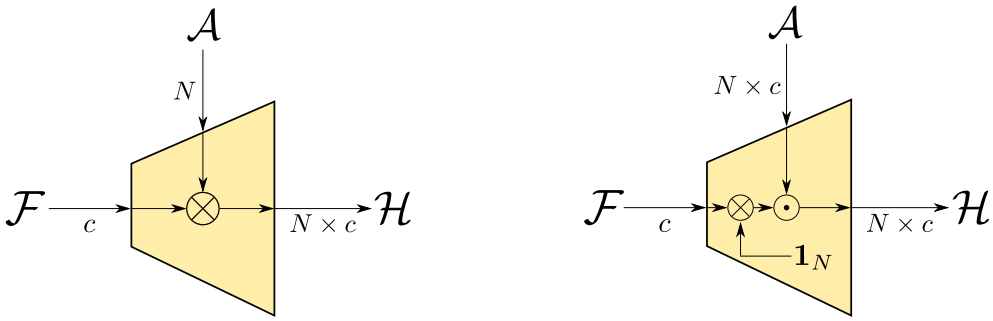
$$\boldsymbol{\mathcal{G}} = \boldsymbol{\psi}\left(\boldsymbol{\mathcal{F}}\right) \tag{4.24}$$

Then, we combine this feature map $\boldsymbol{\mathcal{G}}$ with the attention maps $\boldsymbol{\mathcal{A}}$ using the merging block M:

$$\boldsymbol{\mathcal{H}} = M(\boldsymbol{\mathcal{G}}; \boldsymbol{\mathcal{A}}) \tag{4.25}$$

with $\boldsymbol{\mathcal{H}}$ representing the set of feature maps obtained by the post-attention strategy. From these, we perform a spatial pooling of the local features and we add an embedding layer to generate the corresponding image representation.

The main idea of post-attention consists of aggregating only the related features, unlike global pooling that aggregates unrelated features. For example, features that describe the background are aggregated using a different attention block than those which represent the desired object. One can note that this approach is strongly related to NetVLAD for which the function $\boldsymbol{\psi}$ is a centering. However, it differs from NetVLAD in two ways: First, because it learns a non-linear clustering of the local features using the function $\boldsymbol{\phi}$; second because it learns a non-linear pre-processing of the deep features using the function $\boldsymbol{\psi}$.



(a) Feature-wise selection (b) Dimension-wise selection

Figure 4.4: Illustration of the merging block M for feature-wise and dimension-wise selection strategies. The feature-wise selection is directly implemented using Kronecker product ($\otimes$). The dimension-wise selection first duplicate N times the feature map before computing the Hadamard product ($\odot$) with the attention maps. Reported dimensions are without the spatial dimensions $h \times w$.

4.4.3.2 Pre-attention

In the pre-attention strategy illustrated in Figure 4.3b, we perform the operation in reverse order. In order to do so, we combine the feature map $\boldsymbol{\mathcal{F}}$ with the set of attention maps $\boldsymbol{\mathcal{A}}$ using the merging block M to produce the set of N (one per attention block) feature maps $\boldsymbol{\mathcal{G}}$:

$$\boldsymbol{\mathcal{G}} = M(\boldsymbol{\mathcal{F}}; \boldsymbol{\mathcal{A}}) \tag{4.26}$$

Then, we process these N feature maps using a non-linear function $\boldsymbol{\psi}$ to generate the feature maps $\boldsymbol{\mathcal{H}}$:

$$\boldsymbol{\mathcal{H}} = \boldsymbol{\psi}\left(\boldsymbol{\mathcal{G}}\right) \tag{4.27}$$

From these N feature maps, we pool the local features and we add an embedding layer to generate the corresponding image representation.

The pre-attention strategy is similar to a refinement approach. Indeed, the attention selects dimensions or features, and a refinement function $\boldsymbol{\psi}$ improves these extracted features before that they are aggregated. Hence, the attention maps role is to select only the relevant information for a given attention block. The function $\boldsymbol{\psi}$ is trained to refine this information so that it improves generalization.

4.4.4 Feature-wise selection

In this section, we give details about the computation of the selection block S and the merging block M for the feature-wise selection illustrated in Figure 4.4a. We start from a feature map $\boldsymbol{\mathcal{F}} \in \mathbb{R}^{h \times w \times c}$ from which we compute a set of attention maps $\boldsymbol{\mathcal{A}} = \{\boldsymbol{\mathcal{A}}^{(n)} \in \mathbb{R}^{h \times w \times c}\}_n$ using a dictionary $\mathcal{D}$ such that $\boldsymbol{\mathcal{A}} = \mathrm{S}(\boldsymbol{\mathcal{F}}; \mathcal{D})$. Before computing the feature-wise selection, the feature map $\boldsymbol{\mathcal{F}}$ is processed by a non-linear function $\boldsymbol{\phi} : \mathbb{R}^c \to \mathbb{R}^m$ implemented by a CNN. We denote $\boldsymbol{f}_{i,j} \in \mathbb{R}^c$ a feature from $\boldsymbol{\mathcal{F}}$ at spatial location (i, j) and $\boldsymbol{\phi}(\boldsymbol{f}_{i,j}) \in \mathbb{R}^m$ its transformation in $\boldsymbol{\phi}(\boldsymbol{\mathcal{F}})$ that is in the same spatial location.

4.4.4.1 Selection block

In the feature-wise selection, the objective is to assign each feature $\boldsymbol{f}_{i,j}$ from the feature map $\boldsymbol{\mathcal{F}}$ to one of the dictionary entries $\boldsymbol{\mathcal{D}} = \{\boldsymbol{d}^{(n)} \in \mathbb{R}^m\}_n$. To do so, we consider the cosine similarity between

the transformed feature $\boldsymbol{\phi}(\boldsymbol{f}_{i,j})$ and the dictionary entry:

$$s(\boldsymbol{\phi}(\boldsymbol{f}_{i,j}), \boldsymbol{d}^{(n)}) = \frac{\left\langle \boldsymbol{\phi}(\boldsymbol{f}_{i,j}) \ ; \ \boldsymbol{d}^{(n)} \right\rangle}{\|\boldsymbol{\phi}(\boldsymbol{f}_{i,j})\|_2 \|\boldsymbol{d}^{(n)}\|_2} \tag{4.28}$$

Using the cosine similarity has a main advantage when compared to the Euclidean distance: during the training, it is more stable by means of the ℓ_2-normalization.

From this similarity measure, we compute $\boldsymbol{\mathcal{A}}^{(n)}_{i,j,k}$, the weight for the n-th feature-wise attention map and k-th dimension of the feature in spatial location (i, j):

$$\boldsymbol{\mathcal{A}}^{(n)}_{i,j,k} = \begin{cases} 1 & \text{if } \boldsymbol{d}^{(n)} = \underset{\boldsymbol{d}^{(l)} \in \ \boldsymbol{\mathcal{D}}}{\arg\max} \ s(\boldsymbol{\phi}(\boldsymbol{f}_{i,j}), \boldsymbol{d}^{(l)}) \\ 0 & \text{else.} \end{cases} \tag{4.29}$$

However, this one-hot encoder is not differentiable due to the arg max operator. We relax the constraint using the soft-max operator to train in an end-to-end way the deep network:

$$\boldsymbol{\mathcal{A}}^{(n)}_{i,j,k} = \frac{\mathrm{e}^{\alpha \ s(\boldsymbol{\phi}(\boldsymbol{f}_{i,j}), \boldsymbol{d}^{(n)})}}{\sum_l \mathrm{e}^{\alpha \ s(\boldsymbol{\phi}(\boldsymbol{f}_{i,j}), \boldsymbol{d}^{(l)})}} \tag{4.30}$$

such that, α is a hyper-parameter to control the hardness of the assignment. For this formulation, a given feature $\boldsymbol{f}_{i,j}$ is assigned to a dictionary entry $\boldsymbol{d}^{(n)}$ according to the similarity between $\boldsymbol{\phi}(\boldsymbol{f}_{i,j})$ and $\boldsymbol{d}^{(n)}$. We show in subsection 4.4.7 that the feature-wise selection increases the performance of the attention-based models when compared to the baseline model (without attention map). Then, we detail the block M to merge the feature-wise selection based attention map with the raw features.

4.4.4.2 Merging block

The combination of the attention maps $\boldsymbol{\mathcal{A}}$ and the feature map $\boldsymbol{\mathcal{F}}$ is illustrated in Figure 4.4a. For the k-th dimension of the n-th feature map in spatial location (i, j), the corresponding entry in $\boldsymbol{\mathcal{H}}$, $\boldsymbol{\mathcal{H}}^{(n)}_{i,j,k}$ is computed using the following equation:

$$\boldsymbol{\mathcal{H}}^{(n)}_{i,j,k} = \boldsymbol{\mathcal{A}}^{(n)}_{i,j,k} \ \boldsymbol{\mathcal{F}}_{i,j,k} \tag{4.31}$$

It must be considered that $\boldsymbol{\mathcal{A}}^{(n)}_{i,j,k}$ has the same value independently from the value of k. Thus, Equation 4.31 is easily implemented using the Kronecker product ($\otimes$) as illustrated in Figure 4.4a:

$$\boldsymbol{\mathcal{H}}_{i,j} = \boldsymbol{\mathcal{A}}_{i,j} \otimes \boldsymbol{\mathcal{F}}_{i,j} \tag{4.32}$$

where $\boldsymbol{\mathcal{A}}_{i,j}$ is composed by the N-th assignment weights and $\boldsymbol{f}_{i,j} \in \mathbb{R}^c$ is the feature for all spatial locations (i, j).

4.4.5 Dimension-wise selection

In this section, we extend the feature-wise selection from the subsection 4.4.4 to the dimension-wise selection. Similarly, we give details about the attention maps computation through the selection block S before that we explain the merging strategy with the block M.

4.4.5.1 Selection block

The selection block is composed of a set of directions per dictionary entry $\boldsymbol{\mathcal{D}} = \{\boldsymbol{d}^{(n)}_k \in \mathbb{R}^m\}_{n,k}$ of size $N \times c$, on the contrary to the feature-wise selection that has a dictionary $\boldsymbol{\mathcal{D}} = \{\boldsymbol{d}^{(n)} \in \mathbb{R}^m\}_n$ of size N.

Thus, for a given feature $\boldsymbol{f}_{i,j} \in \boldsymbol{\mathcal{F}}$ in spatial location (i, j), the cosine similarity is computed between the transformed feature $\boldsymbol{\phi}(\boldsymbol{f}_{i,j})$ and the k-th direction of the n-th dictionary entry $\boldsymbol{d}_k^{(n)} \in \boldsymbol{\mathcal{D}}$:

$$s(\boldsymbol{\phi}(\boldsymbol{f}_{i,j}), \boldsymbol{d}_k^{(n)}) = \frac{\left\langle \boldsymbol{\phi}(\boldsymbol{f}_{i,j}) \ ; \ \boldsymbol{d}_k^{(n)} \right\rangle}{\|\boldsymbol{\phi}(\boldsymbol{f}_{i,j})\|_2 \|\boldsymbol{d}_k^{(n)}\|_2} \tag{4.33}$$

Then, one entry $\boldsymbol{\mathcal{A}}_{i,j,k}^{(n)}$ from the attention map $\boldsymbol{\mathcal{A}}$ is computed using the following equation:

$$\boldsymbol{\mathcal{A}}_{i,j,k}^{(n)} = \frac{\mathrm{e}^{\alpha \ s(\boldsymbol{\phi}(\boldsymbol{f}_{i,j}), \boldsymbol{d}_k^{(n)})}}{\sum_l \mathrm{e}^{\alpha \ s(\boldsymbol{\phi}(\boldsymbol{f}_{i,j}), \boldsymbol{d}_k^{(l)})}} \tag{4.34}$$

The attention map has an attention weight for each dimension of the input feature and for each of the N attention blocks. subsection 4.4.7 highlights that the dimension-wise selection produces stronger image representations than the feature-wise selection, showing much higher performances.

4.4.5.2 Merging block

The merging block M is used in the case of dimension-wise selection as illustrated in Figure 4.4b. The entry $\boldsymbol{\mathcal{H}}_{i,j,k}^{(n)}$ in $\boldsymbol{\mathcal{H}}$ is computed for the k-th dimension of the n-th feature map in the spatial location (i, j) using the following equation:

$$\boldsymbol{\mathcal{H}}_{i,j,k}^{(n)} = \boldsymbol{\mathcal{A}}_{i,j,k}^{(n)} \ \boldsymbol{\mathcal{F}}_{i,j,k} \tag{4.35}$$

Note that $\boldsymbol{\mathcal{A}}_{i,j,k}^{(n)}$ depends on the value of k. The computation can easily be re-written using the Kronecker product and the Hadamard product ($\odot$) as illustrated in Figure 4.4b. Using the Kronecker product, we duplicate N times the local feature $\boldsymbol{f}_{i,j}$ Then, $\boldsymbol{\mathcal{H}}$ is computed using the element-wise product between $\boldsymbol{\mathcal{A}}$ and the duplicated feature map:

$$\boldsymbol{\mathcal{H}} = (\boldsymbol{\mathcal{F}} \otimes \mathbf{1}_N) \odot \boldsymbol{\mathcal{A}} \tag{4.36}$$

4.4.6 Implementation details

For a fair comparison with other methods, we use a pre-trained GoogleNet [163] on ImageNet. The embedding size is fixed to 512 for all models. As it is done in common practice, we set the triplet margin $\alpha = 0.1$, the contrastive and the binomial margin $\beta = 0.5$ and the negative sample weight $C = 25$ in the binomial deviance. We follow the same training procedure as state-of-the-art methods [129, 98]: training hyper-parameters (learning rate, batch size, *etc.*) are empirically chosen based on the training loss after a few epochs. Model hyper-parameters (number of layers, *etc.*) are set to comparable values as the ones of models reported in ABE [98]. We use the following data augmentation on the images: multi-resolution where the size is uniformly sampled in $[0.8, 1.8]$ times the crop size as well as random 256×256 crop and horizontal flip during the training. During testing, we re-scale the images to 256×256. We use Adam [99] with the default parameters and a learning rate of 10^{-5}. For all models, the function $\boldsymbol{\psi}$ is shared across the attention maps to reduce the large number of parameters induced. In practice, it still leads to strong experimental results and avoids over-fitting on small datasets, for instance Cub-200-2011 and Cars-196.

4.4.7 Ablation studies

In this section, we compare our approach to the baseline in terms of model complexity and computation. Then, we present ablation studies on pre and post-attention, on the assignment approaches (feature and dimension-wise), on the dictionary strategy and on the dictionary size.

	Cub-200-2011						Cars-196					
R@	1	2	4	8	16	32	1	2	4	8	16	32
RML [146]	52.3	64.5	75.3	84.0	-	-	73.2	82.2	88.6	92.2	-	-
Angular loss [175]	54.7	66.3	76.0	83.9	-	-	71.4	81.4	87.5	92.1	-	-
DAML [50]	52.7	65.4	75.5	84.3	-	-	75.1	83.8	89.7	93.5	-	-
HDML [202]	53.7	65.7	76.7	85.7	-	-	79.1	87.1	92.1	95.5	-	-
DAMLRMM [193]	55.1	66.5	76.8	85.3	-	-	73.5	82.6	89.1	93.5	-	-
HDC [197]	53.6	65.7	77.0	85.6	91.5	95.5	73.7	83.2	89.5	93.8	96.7	98.4
BIER [129]	55.3	67.2	76.9	85.1	91.7	95.5	78.0	85.8	91.1	95.1	97.3	98.7
DVML [111]	52.7	65.1	75.5	84.3	-	-	82.0	88.4	93.3	96.3	-	-
HTG [201]	59.5	71.8	81.3	88.2	-	-	76.5	84.7	90.4	94.0	-	-
HTL [57]	57.1	68.8	78.7	86.5	92.5	95.5	81.4	88.0	92.7	95.7	97.4	99.0
A-BIER [130]	57.5	68.7	78.3	86.2	91.9	95.5	82.0	89.0	93.2	96.1	97.8	98.7
JCF [84]	60.1	72.1	81.7	88.3	-	-	82.6	89.2	93.5	96.0	-	-
HORDE [85]	59.4	71.0	81.0	88.0	93.1	96.5	83.2	89.6	93.6	96.3	98.0	98.8
ABE [98]	60.6	71.5	79.8	87.4	-	-	85.2	90.5	94.0	96.1	-	-
Contrastive (Ours)	58.7	69.7	79.4	87.0	92.6	96.1	78.5	85.9	90.9	94.4	96.7	98.1
Contrastive + *DIABLO*	62.3	73.6	82.6	89.2	94.0	96.9	84.8	90.5	94.3	96.6	98.1	98.9
Triplet (Ours)	55.9	67.0	77.7	86.1	92.0	95.5	73.1	81.7	87.9	92.9	95.9	97.6
Triplet + *DIABLO*	59.6	70.6	80.3	87.7	92.7	96.2	75.0	83.4	89.4	93.5	96.4	98.1
Binomial (Ours)	59.6	70.8	81.0	88.1	93.1	96.2	78.8	86.2	91.5	94.8	97.0	98.4
Binomial + *DIABLO* $N = 8$	62.8	73.9	82.4	89.3	94.0	96.7	85.0	90.8	94.0	96.4	98.0	98.9
Binomial + *DIABLO* $N = 16$	**63.9**	**74.3**	**82.4**	**88.8**	**94.0**	**96.8**	**85.4**	**91.3**	**95.0**	**97.2**	**98.5**	**99.1**

Table 4.5: Comparison with the state-of-the-art on Cub-200-2011 and Cars-196 datasets using GoogleNet as feature extractor. Results are in percents. State-of-the-art results are in bold and results which improve the baseline are underlined.

	Stanford Online Products				In-Shop Clothes Retrieval					
R@	1	10	100	1000	1	10	20	30	40	50
Angular loss [175]	70.9	85.0	93.5	98.0	-	-	-	-	-	-
DAML [50]	68.4	83.5	92.3	-	-	-	-	-	-	-
HDML [202]	68.7	83.2	92.4	-	-	-	-	-	-	-
RML [146]	69.2	83.1	92.7	-	-	-	-	-	-	-
DAMLRMM [193]	69.7	85.2	93.2	-	-	-	-	-	-	-
HDC [197]	69.5	84.4	92.8	97.7	62.1	84.9	89.0	91.2	92.3	93.1
BIER [129]	72.7	86.5	94.0	98.0	76.9	92.8	95.2	96.2	96.7	97.1
DVML [111]	70.2	85.2	93.8	-	-	-	-	-	-	-
HTG [201]	-	-	-	-	80.3	93.9	95.8	96.6	97.1	-
HORDE [85]	72.6	85.9	93.7	97.9	84.4	95.4	96.8	97.4	97.8	98.1
HTL [57]	74.8	88.3	94.8	98.4	80.9	94.3	95.8	97.2	97.4	97.8
A-BIER [130]	74.2	86.9	94.0	97.8	83.1	95.1	96.9	97.5	97.8	98.0
ABE [98]	76.3	88.4	94.8	98.2	87.3	96.7	97.9	98.2	98.5	98.7
JCF [84]	77.4	89.9	95.8	98.6	-	-	-	-	-	-
Contrastive (Ours)	75.0	87.9	94.5	98.1	89.0	97.2	98.0	98.4	98.6	98.7
Contrastive + *DIABLO* $N = 8$	**77.8**	**89.5**	**95.3**	**98.4**	**91.3**	**98.1**	**98.7**	**99.0**	**99.1**	**99.1**
Triplet (Ours)	70.6	85.7	94.0	98.2	85.6	96.5	97.7	98.2	98.4	98.6
Triplet + *DIABLO* $N = 8$	73.5	87.8	95.0	98.5	87.4	97.2	98.1	98.6	98.8	98.9

Table 4.6: Comparison with the state-of-the-art on Stanford Online Products and In-Shop Clothes Retrieval using GoogleNet as feature extractor. Results in percents. State-of-the-art results are in bold and results which improve the baseline are underlined.

4.4.7.1 Model complexity and computation cost

In this section, we give the architecture choices to compute the functions ϕ and ψ for the post-attention and for the pre-attention architectures. Then, we analyze the induced computations and the complexity introduced by the dictionary approach.

In post-attention, the GoogleNet backbone extracts the feature map including and up to the max-pooling between the fourth and the fifth scales. To compute ϕ, we add the two inception blocks named '5a' and '5b' upon these features. The function ψ is also composed by two different inception blocks '5a' and '5b'. The attention maps and the weighted features are computed using one of the two proposed strategies. Then, each branch is pooled using a global average pooling as well

		Feature		Dimension	
	Baseline	Pre-att	Post-att	Pre-att	Post-att
R@1	59.6	57.6	**60.3**	**62.8**	60.8

Table 4.7: Impact of the pre-attention or post-attention on performances for the three proposed attention strategies. Reported Recall@1 (R@1) is on the Cub-200-2011 dataset in percent.

as an embedding layer of size $512/N$ and a ℓ_2-norm are added. The output representation is the concatenation of these N branches which leads to a 512d representation.

In pre-attention, we use the GoogleNet features including and up to the max-pooling between the third and the fourth scales. The non-linear function $\boldsymbol{\phi}$ is composed of the five inception blocks from the fourth scale of GoogleNet pre-trained on ImageNet. The refinement function $\boldsymbol{\psi}$ is composed by the fourth and fifth scales of GoogleNet, they are shared for each map but they are independent from $\boldsymbol{\phi}$. The attention maps and the weighted feature maps are computed using either dimension-wise or feature-wise selection. As it the case with pre-attention, each branch is pooled using a global average pooling, an embedding layer of size $512/N$ and a ℓ_2-norm are added before the concatenation of all branches in order to produce the full 512d representation.

These choices directly follow ABE [98] and we refer the reader to the corresponding paper for more ablations on the architecture, including the multi-head approach, the attention module and the sharing of parameters across the attention modules.

All additional parameters are included in the computation of the function $\boldsymbol{\phi}$. By using the five Inception blocks from the fourth scale of GoogleNet, this leads to 3.5M additional parameters. Note that these parameters are shared across the dictionary entries and this drastically reduces the number of parameters. Also, note that the function $\boldsymbol{\psi}$ is already included in the number of parameters of GoogleNet.

In terms of computation, the most important additional computations come from the function $\boldsymbol{\psi}$ which is computed on each attention map. The computation of the fourth and fifth Inception scales are estimated to 0.7 Gflop (see [98], Table 1). In comparison, the whole GoogleNet requires 1.6 Gflop to produce the image embedding. Note that in the case of the pre-attention, all parameters of $\boldsymbol{\psi}$ are shared across the attention maps, which leads to fewer additional parameters. Overall, these choices lead to higher experimental results for both ABE and *DIABLO* compared to the baseline.

4.4.7.2 Feature selection and attention strategy

First, we evaluate the benefit of the pre and post-attention strategies with respect to the assignment strategy. To that end, we fix the dictionary size to 8 and we use the training procedure from subsection 4.4.6. Results are reported in Table 4.7 for the dataset Cub-200-2011 with binomial loss. We remark that all strategies with the exception of the feature-wise selection with pre-attention improve over the baseline and this confirms the benefit of attention maps. This experiment also shows that feature-wise attention and dimension-wise attention impact differently the model.

In Feature-wise attention, the post-attention leads to stronger representations with +2.7% on Recall@1 over the pre-attention. We argue that selecting features make better sense with post-attention than pre-attention: only related features are aggregated together with post-attention whereas in pre-attention, the refinement part mostly processes sparse feature maps. In the case of dimension-wise attention, both approach provide stronger results with +0.5% and +2.5% over the best feature-wise strategy, even though it is still better to use the pre-attention approach. We argue that pre-attention is better with dimension-wise selection because the refinement part processes denoised features. Indeed, certain dimensions may be useless for a given dictionary entry and the dimension-wise approach can select only the relevant dimensions. Then, the refinement part is trained with sparse vectors which contain only the relevant information. Moreover, feature-wise selection with post-attention leads to aggregate sparse vector together, which might bring to sub-optimal results because some dimensions are rarely used.

	Feat. + Post-att.				Dim. + Pre-att.			
N	2	4	8	16	2	4	8	16
R@1	59.3	58.8	60.3	59.2	61.0	61.9	62.8	63.9

Table 4.8: Impact of the dictionary size for the feature-wise post-attention mapping and for the dimension-wise pre-attention mapping. Reported results are Recall@1 (R@1) on the Cub-200-2011 dataset in percent.

4.4.7.3 Comparison to ABE

We compare our method to ABE [98] in order to evaluate the impact of the structural constraints imposed by the dictionary (Equation 4.30 and Equation 4.34). In ABE, the authors show that a M-head approach already provides strong results on the Cars-196 dataset with 76.1% (+8.9%) Recall@1 for $M = 8$. However, this architecture tends to overfit due to the large number of parameters. Then, they propose an enhanced version named ABE that takes advantage of attention maps. The divergence loss increases the performance from 69.7% without the divergence loss to 85.2% (+15.5%) in Recall@1 (see Table 2 in [98]).

In addition to a new hyper-parameter, the divergence loss also has the drawback that optimizing the loss generates gradients which are in opposition with the metric learning loss ones. Indeed, it is designed to reduce the similarity between different branches even when the images are similar. We solve this issue in *DIABLO* where this optimization constraint is replaced by a structural constraint (softmax in Equation 4.30 and Equation 4.34). The orthogonality is ensured by the design of *DIABLO*, which allows to simply remove the divergence loss at the price of a reduced expressiveness: feature map entries can be chosen independently in ABE whereas in *DIABLO* they are constrained to only one dictionary entry. In Table 4.5 and Table 4.6, *DIABLO* performs well compared to ABE with similar results on the Cars-196 dataset (-0.2% compared to ABE) and higher ones on other datasets such as Cub-200-2011 (+2.2%), Stanford Online Products (+1.5%) and Inshop Clothes Retrieval (+3.4%) for this set of parameters.

4.4.7.4 Dictionary size

In this section, we evaluate the impact of the dictionary size on *DIABLO*. We evaluate both the feature-wise and the dimension-wise with dictionary sizes in $\{2, 4, 8, 16\}$. Recall@1 on the Cub-200-2011 dataset are reported in Table 4.8.

In post-attention with feature-wise selection, extreme combinations (*e.g.*, with 2 branches with 256 dimensions or 16 branches with 32 dimensions) lead to results under the baseline. Thus, to increase performances of such attention strategy, there is a compromise between the representation size of each branch and the number of branches (+0.7% over the baseline). In pre-attention with dimension-wise selection, all parameter combinations for this approach lead to better results than the baseline (+0.4% to +4.3%). The number of branches increases the performance with the log of the dictionary size i the case of pre-attention with dimension-wise selection on the contrary to the post-attention with feature-wise selection.

4.4.8 Comparison to the state-of-the-art

In this section, we compare *DIABLO* to the state-of-the-art on 4 DML datasets, named Cub-200-2011 [174], Cars-196 [102], Stanford Online Products [127] and In-Shop Clothes Retrieval [115]. For Cub-200-2011, Cars-196 and Stanford Online Products, we follow the standard splits from [127] and for In-Shop Clothes Retrieval we follow the one from [115]. We report the Recall@K which evaluates, for a given query, if there is at least one image with the same label in the top-K retrieved images. We use $K \in \{1, 2, 4, 8, 16, 32\}$ for Cub-200-2011 and Cars-196, $K \in \{1, 10, 100, 1000\}$ for Stanford Online Products and $K \in \{1, 10, 20, 30, 40, 50\}$ for In-Shop Clothes Retrieval.

We report the results for Cub-200-2011 and Cars-196 in Table 4.5 and in Table 4.6 for Stanford Online Products and In-Shop Clothes Retrieval. *DIABLO* consistently improves the already strong baseline on the four datasets and for three different loss functions. *E.g.*, using the binomial loss, the baseline is improved from 59.6% to 63.9% (+4.3%) on Cub-200-2011 and from 78.8% to 85.4% (+6.6%) on the Cars-196 dataset. The same observation is made for both the other loss functions on these datasets.

Moreover, *DIABLO* leads to better results when compared to the similar approach ABE. Their best reported approach, ABE-8, is outperformed by *DIABLO* with $N = 8$ and $c = 64$ (total dimension 512) by 2.2% in R@1 on Cub-200-2011, by 1.5% on Stanford Online Products and by 4% on In-Shop Clothes Retrieval. Finally, following the ablation study in Table 4.8, we report results with $N = 16$ and $c = 32$ (total dimension 512): This setup further improves performances on the Cub-200-2011 and Cars-196 dataset by 1.1% and 0.4% respectively in Recall@1. Thus leads to state-of-the-art results on the four deep metric learning datasets.

4.5 Global Framework

In this section, we discuss the link between our three contributions and their applications. First, we discuss the link between the VLAT, the STA and the *JCF* representations. The VLAT representation, without specific dimensionality reduction strategies, is computed as follows:

$$\boldsymbol{T}_k(\boldsymbol{\mathcal{I}}) = \frac{1}{|\boldsymbol{\mathcal{I}}_k|} \sum_{\boldsymbol{x} \in \boldsymbol{\mathcal{I}}_k} (\boldsymbol{x} - \boldsymbol{\mu}_k) \otimes (\boldsymbol{x} - \boldsymbol{\mu}_k) - \widehat{\boldsymbol{T}}_k \tag{4.37}$$

where k stands for the k-th dictionary entry. Note that the VLAT representation can be re-written using the tensor operators as follows:

$$\boldsymbol{\mathcal{T}}(\boldsymbol{\mathcal{I}}) = \sum_{\boldsymbol{x} \in \boldsymbol{\mathcal{I}}} \boldsymbol{h}(\boldsymbol{x}) \otimes \left(\frac{1}{\sum_{\boldsymbol{y} \in \boldsymbol{\mathcal{I}}} \langle \boldsymbol{h}(\boldsymbol{x}) \ ; \ \boldsymbol{h}(\boldsymbol{y}) \rangle} (\boldsymbol{x} - \boldsymbol{D}\boldsymbol{h}(\boldsymbol{x})) \otimes (\boldsymbol{x} - \boldsymbol{D}\boldsymbol{h}(\boldsymbol{x})) \right) - \widehat{\boldsymbol{\mathcal{T}}} \tag{4.38}$$

where $\boldsymbol{D} \in \mathbb{R}^{c \times N}$ is the dictionary written as a matrix. Similarly, the STA representation, after the choice of a neighborhood Ω, is computed as follows:

$$\boldsymbol{\mathcal{T}}(\boldsymbol{\mathcal{I}}) = \sum_{\boldsymbol{x} \in \boldsymbol{\mathcal{I}}} \sum_{\boldsymbol{x}_u \in \Omega(\boldsymbol{x})} \boldsymbol{h}(\boldsymbol{x}) \otimes \boldsymbol{h}(\boldsymbol{x}_u) \otimes \boldsymbol{x} \otimes \boldsymbol{x}_u \tag{4.39}$$

Finally, the *JCF* representations, before the specific dimensionality reduction, is formulated as follows:

$$\boldsymbol{\mathcal{T}}(\boldsymbol{\mathcal{I}}) = \sum_{\boldsymbol{x} \in \boldsymbol{\mathcal{I}}} \boldsymbol{h}(\boldsymbol{x}) \otimes \boldsymbol{h}(\boldsymbol{x}) \otimes \boldsymbol{x} \otimes \boldsymbol{x} \tag{4.40}$$

In a nutshell, the STA and the *JCF* representation are tightly related to each other: *JCF* is a special case of STA where the neighborhood contains only the central feature. Also, the VLAT representation is tightly related to the other ones: by duplicating the assignment branch and by withdrawing the centering and the scaling, we get the *JCF* representation. As a consequence, most of the proposed improvements of the aforementioned representations can also be exploited with a small re-writing to enhance the other ones. Thus, we can consider the factorization in *JCF* and the differentiable dictionary learning for both STA and VLAT to build end-to-end trainable architectures. Moreover, we can take advantage of the VLAD and VLAT improvements [44, 2] to improve our end-to-end trainable representations. Last but not least, the *JCF* representation can be extended to consider the neighborhood of STA: by adding a depth-wise convolution [78] before the combination with the dictionary, this leads to a differentiable of STA with the factorization inherited from *JCF*. Thus, a $M \times N$ depth-wise convolution allows to consider a rectangular neighborhood of size $M \times N$. If we want to consider different neighborhood, a simple solution is to take a larger convolution and to enforce some weights to zero.

Second, we present the links between the aforementioned representations and the framework from *DIABLO*. The first consideration is about the assignment function in the dictionary part. Indeed, in VLAD, VLAT, STA or *JCF*, we directly assign the local features to a dictionary entry. Thus, the clustering involved in these representations is linear. In *DIABLO*, the use of the function $\boldsymbol{\phi}$ improves this assignment by considering non-linear assignment. This formulation can be compared to the kernel method. Indeed, the function $\boldsymbol{\phi}$ can be seen as the non-linear transformation of the kernel. However, this differs from the kernel method by directly exploiting the function instead of the kernel. Thus, the dictionary is directly learned into the non-linear space to improve the dictionary assignment. The second consideration is about the non-linear function $\boldsymbol{\psi}$ in the post-attention setup. Indeed, the VLAT representation considers the second-order pooling of the deep features with a centering and a scaling as transformation. Thus, we can modify the non-linear transformation $\boldsymbol{\psi}$ in *DIABLO* such that:

$$\boldsymbol{\psi}(\boldsymbol{x}) = \frac{1}{\sum_{\boldsymbol{y} \in \boldsymbol{\mathcal{I}}} \langle \boldsymbol{h}(\boldsymbol{x}) \ ; \ \boldsymbol{h}(\boldsymbol{y}) \rangle} \left(\boldsymbol{x} - \boldsymbol{D}\boldsymbol{h}(\boldsymbol{x})\right) \otimes \left(\boldsymbol{x} - \boldsymbol{D}\boldsymbol{h}(\boldsymbol{x})\right) \tag{4.41}$$

By doing so, the VLAT representation is a special case of the *DIABLO* framework. Interestingly, a similar thing can be done on *JCF*: the $\boldsymbol{\psi}$ function integrates the factorization and the merging block computes the low-rank approximation for *JCF*-N-R and the matrix product. Consequently, differentiable version of STA and VLAT but also *JCF* can be improved by following the results from *DIABLO*: non-linear clustering, pre-attention, *etc.*

4.6 Conclusion

In this chapter, we have presented our contributions on tensor-based aggregation methods. First, we have extended the Spatial Tensor Aggregation (STA) representation by considering centering, normalization and dimensionality reduction strategies to build a more compact representation with higher performances. This representation has been evaluated on three image retrieval datasets named Holidays, Paris6k and Oxford5k and leads to higher performances than end-to-end trainable architectures such as NetVLAD and Crow. Second, we have proposed a differentiable factorization scheme that allows to train end-to-end second-order statistics-based representation with a dictionary learning. By taking advantage of the property between the Kronecker product and the dot product, we have designed a factorization scheme that indirectly computes the second-order statistics of the deep local features. This new representation named *JCF* leads to a very compact representation compared to standard second-order representations with state-of-the-art performances. Third, we have modified the ABE and the NetVLAD representations by considering the link between dictionary-based and attention-based approaches. By doing so, the divergence loss in the original ABE model has been replaced by a dictionary. Thus, the structural constraint of the dictionary makes the model easier to train than ABE with optimization constraint from the divergence loss.

Last, we have discussed the links between our proposed methods and other image representation such as VLAT. On the one hand, we have presented the links between the following image representations: VLAT, STA/ISTA ad *JCF*. We have shown that VLAT is a special case of STA and *JCF*: VLAT and STA are is the same representation if we consider an neighborhood with only the central feature and VLAT and *JCF* (without the factorization) are the same if we drop the duplicated assignment. Thus, all of our contributions (normalization, pre-processing, dimensionality reduction, *etc.*) can be exploited for all of these representations. On the other hand, we have stressed that the framework proposed in *DIABLO* generalizes these representations and give insights to improve them. Indeed, by designing specifically the non-linear transformation functions on the deep features, we show that we can mimic the VLAT and the *JCF* representations. Also, by adding a depth-wise convolution, *DIABLO* generalizes the STA representation. This allows to consider a differentiable formulation of STA but also to take advantage in the factorization proposed in *JCF*.

5

HORDE: High-Order Regularizer for Deep Embedding

In this chapter, we present a regularization method for the local deep features to strengthen the global representation and the embedding.

Contents

5.1 Prologue **74**
5.2 Introduction **75**
5.3 Proposed High-Order Regularizer **75**
5.3.1 High-order computation 76
5.3.2 Theoretical analysis 78
5.4 Experiments **80**
5.4.1 Comparison to state-of-the-art 80
5.4.2 Ablation studies 82
5.5 Discussion **84**
5.5.1 Magnet loss and DVML 84
5.5.2 Empirical estimators and the image distribution hypothesis 84
5.5.3 Image representation 85
5.5.4 Random Maclaurin factorization 85
5.6 Conclusion **86**

5.1 Prologue

Research paper detailed:

- **Pierre Jacob**, David Picard, Aymeric Histace, Edouard Klein. *"Metric Learning With HORDE: High-Order Regularizer for Deep Embeddings"*. Proceedings of the IEEE International Conference on Computer Vision (ICCV), Oct. 2019.

Related presentations:

- **Pierre Jacob**, David Picard, Aymeric Histace, Edouard Klein. *"Metric Learning With HORDE: High-Order Regularizer for Deep Embeddings"*. GDR ISIS Workshop "Deep Learning Theory", Oct. 2019.
- **Pierre Jacob**, David Picard, Aymeric Histace, Edouard Klein. *"Metric Learning With HORDE: High-Order Regularizer for Deep Embeddings"*. WILLOW-ENPC-Berkeley Workshop "Vision and Robotics", Nov. 2019.

Context:
We provide in this chapter a regularization method in the context of deep metric learning. Metric learning aims to learn the image representations and a metric together such that similar images have similar representations with respect to the learned metric and dissimilar images have image representation far away. Most of state-of-the-art methods tackle the metric learning problem by considering better loss functions [127, 152, 158, 175, 182], mining strategies [111, 193, 202] or ensemble methods [98, 129, 130, 197]. However, there is few recent work on the representation themselves because the dimensionality imposed by the metric learning framework is very small, typically 512 dimensions. This largely penalizes high-order representations, dictionary-based representations, and so on. Also, there is few work on the image representation training procedure except from the loss function. All methods rely on standard SGD or on Adam [99] to optimize the weights of the deep network, the representations and the metric.

In this work, we also differ from the standard contributions in deep metric learning by providing a regularization method named *HORDE*. This regularizer is designed to solve what we call the *scattering problem*. The scattering problem is the phenomenon observed in Figure 5.1: deep local features extracted from training images are not localized around the representation which is, in this case, the mean of the deep features. Thus, in the case of feature sampling issues (which might be the consequence of occlusion, illumination change, *etc.*), this simple image representation is heavily affected and performances of the deep network are strongly reduced. As a consequence, we propose *HORDE*, a High-Order Regularizer for Deep Embedding which is designed to optimize the local feature to be the same in the case of similar images and to be different in the case of dissimilar images. To do so, we consider an image as a distribution and the deep network which extracts deep local features as a sampler of random vectors that follow the given distribution. Then, the optimization of the features to be similar (resp. dissimilar) is the same problem as the optimization of a divergence between similar (resp. dissimilar) distributions. To this end, we propose to optimize a divergence between the distribution of the deep local features in addition to the distance between the image representations.

In *HORDE*, we propose to optimize embeddings of the high-order moments of the deep features instead of directly tackle the deep feature distributions. This choice is deeply discussed in section 5.5 with respect to: the implication of empirical estimators of divergences between distribution, the strong constraint induced by the divergence and the main hypothesis, the computation of the image representation. We also discuss about related work that that the Magnet loss [143] and the Deep Variational Metric Learning [111].

Contributions:
The contribution of this work is to provide a new way to improve deep metric learning models by strengthening the image representations to be more robust to the *scattering problem*, thanks to a regularization method. This regularization is easy to implement and naturally extends the deep metric learning framework by optimizing embeddings of the high-order moments of the deep

features. We give theoretical guarantees of our methods (upper and lower bounds of divergence between distributions) and we experimentally confirm the benefits of *HORDE*.

5.2 Introduction

In most of recent contributions in deep metric learning, the image representations are obtained by the aggregation of the deep features using a Global Average Pooling [203]. Thus, the deep features are summarized using the sample mean, and the training process makes sure that the sample mean is discriminative enough for the target task. Our insight is that ignoring the characteristics of the deep feature distribution leads to a lack of distinctiveness in the deep features. We illustrate this phenomenon in Figure 5.1. In Figure 5.1a, we train a DML model on MNIST and plot both the deep features (points) and the image representations (stars) from a set of images sampled from the training set. We observe that the representations are perfectly organized while the deep features are in contrast scattered in the entire feature space. As the representations are obtained using the sample mean only, they are sensitive to outliers or sampling problems (occlusions, illumination, background variation, etc.), which we refer to as the *scattering problem*. We illustrate this problem in Figure 5.1b where the representations are computed using the same architecture but by sampling only 1/6-th of the original deep features. As we can see, the resulting representations are no longer correctly organized.

In this work, we propose *HORDE*, a High-Order Regularizer for Deep Embeddings which tackles this scattering problem. By minimizing (resp. maximizing) the distance between high-order moments of the deep feature distributions, this DML regularizer enforces deep feature distributions from similar (resp. dissimilar) images to be nearly identical (resp. to not overlap). As illustrated in Figure 5.1c, our *HORDE* regularizer produces well localized features, leading to robust image representations even if they are computed using only 1/6-th of the original deep features.

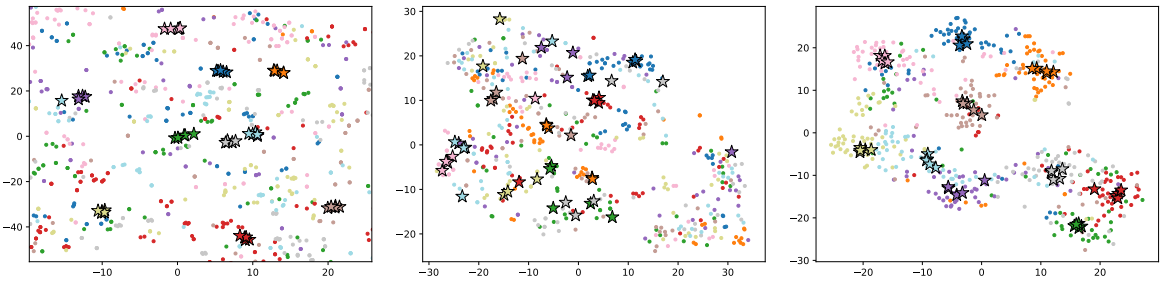
Our contributions are the following: First, we propose a High-Order Regularizer for Deep Embeddings (*HORDE*) that reduces the scattering problem and allows the sample mean to be a robust representation. We provide a theoretical analysis in which we support this claim by showing that *HORDE* is a lower bound of the Wasserstein distance between the deep feature distributions while also being an upper-bound of their Maximum Mean Discrepancy. Second, we show that *HORDE* consistently improves DML with varying loss functions, even when considering ensemble methods. Using *HORDE*, we are able to obtain state of the art results on four standard DML datasets (Cub-200-2011 [174], Cars-196 [102], In-Shop Clothes Retrieval [115] and Stanford Online Products [127]).

The remaining of this chapter is organized as follows: In section 5.3, after an overview of our proposed method, we present the practical implementation of *HORDE* as well as a theoretical analysis. In section 5.4 we compare our proposed architecture with the state-of-the-art on four image retrieval datasets and we conduct extensive experiments to demonstrate the robustness of our regularization and its statistical consistency.

5.3 Proposed High-Order Regularizer

We first give an overview of the proposed method in Figure 5.2. We start by extracting a deep feature map of size $h \times w \times c$ using a CNN where h and w are the height and width of the feature map and c is the deep features dimension. Following standard DML practices, these features are aggregated using a Global Average Pooling to build the image representation and are projected into an embedding space before a similarity-based loss function is computed over these representations (top-right blue box in Figure 5.2).

In *HORDE*, we directly optimize the distribution of the deep features by minimizing (respectively maximizing) a distance between the deep feature distributions of similar images (respectively dissimilar images). We approximate the deep feature distribution distance by computing high-order moments (bottom-right red box in Figure 5.2). We recursively approximate the high-order moments



(a) Representations using all deep features

(b) Representations using 1/6-th of deep features

(c) Representations using 1/6-th of deep features trained with *HORDE*

Figure 5.1: 2D visualizations of representations (stars) and deep features (points) using t-SNE computed from a DML architecture on MNIST dataset with **one color per class**. Representations and features come from the **training set**. Figure 5.1a shows discriminative representations but with scattered deep features (Remark the scale of the axes). Figure 5.1b shows representations computed with 1/6-th of the deep features, leading to a disorganized space. Figure 5.1c shows the same model trained with *HORDE*: the deep features are well concentrated and the representations computed using 1/6-th of the deep features are organized according to the classes (best viewed on a computer screen).

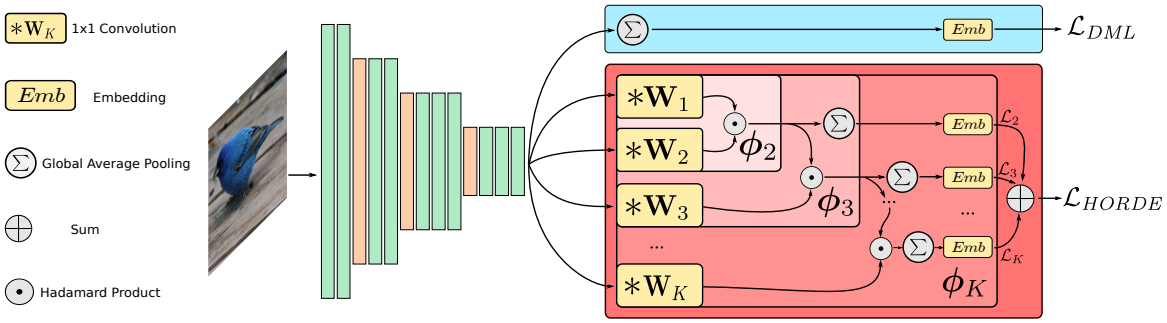


Figure 5.2: Global overview of our *HORDE* architecture. The deep convolutional neural network extracts $h \times w \times c$ deep features. The standard architecture (top blue block) relies on a global average pooling and an embedding before computing the $\mathcal{L}_{\text{DML}}$ loss. The bottom red block is our *HORDE* regularizer, composed by the approximation of all high-order moments $\boldsymbol{\phi}_k$, global average pooling and embeddings before computing the sum of each $\mathcal{L}_k$ loss.

and we compute an embedding after each of these approximations. Then, we apply a DML loss function on each of these embeddings.

5.3.1 High-order computation

In practice, the computation of high-order moments is very intensive due to their high dimension. Furthermore, it has been shown in [87, 129] that an independence assumption over all high-order moment components is unrealistic. Hence, we rely on factorization schemes to approximate their computation, such as Random Maclaurin (RM) [94]. The RM algorithm relies on a set of random projectors to approximate the inner product between two high-order moments. In the case of the second-order, we sample two independent random vectors $\boldsymbol{w}_1, \boldsymbol{w}_2 \sim \mathcal{W}$ where $\mathcal{W}$ is a uniform distribution in $\{-1, +1\}$. For two non random vectors $\boldsymbol{x}$ and $\boldsymbol{y}$, the inner product between their

second-order moments can be approximated as:

$$\begin{aligned}\mathbb{E}_{\boldsymbol{w}_1,\boldsymbol{w}_2\sim\mathcal{W}}[\phi_2(\boldsymbol{x})\phi_2(\boldsymbol{y})] &= \mathbb{E}_{\boldsymbol{w}_1,\boldsymbol{w}_2\sim\mathcal{W}}[\langle \boldsymbol{w}_1 \ ; \ \boldsymbol{x}\rangle \langle \boldsymbol{w}_2 \ ; \ \boldsymbol{x}\rangle \langle \boldsymbol{w}_1 \ ; \ \boldsymbol{y}\rangle \langle \boldsymbol{w}_2 \ ; \ \boldsymbol{y}\rangle] \\ &= \langle \boldsymbol{x} \ ; \ \boldsymbol{y}\rangle^2 \\ &= \langle \boldsymbol{x}\otimes\boldsymbol{x} \ ; \ \boldsymbol{y}\otimes\boldsymbol{y}\rangle \end{aligned} \tag{5.1}$$

where $\otimes$ is the Kronecker product, $\mathbb{E}_{\boldsymbol{w}_1,\boldsymbol{w}_2\sim\mathcal{W}}$ is the expectation over the random vectors $\boldsymbol{w}_1$ and $\boldsymbol{w}_2$ which follow the distribution $\mathcal{W}$ and $\phi_2(\boldsymbol{x}) = \langle \boldsymbol{w}_1 \ ; \ \boldsymbol{x}\rangle \langle \boldsymbol{w}_2 \ ; \ \boldsymbol{x}\rangle$. This approach easily holds to estimate any inner product between K-th moments:

$$\begin{aligned}\mathbb{E}_{\boldsymbol{w}_k\sim\mathcal{W}}[\phi_K(\boldsymbol{x})\phi_K(\boldsymbol{y})] &= \langle \boldsymbol{x} \ ; \ \boldsymbol{y}\rangle^K \\ &= \left\langle \underbrace{\boldsymbol{x}\otimes\cdots\otimes\boldsymbol{x}}_{K \text{ times}} \ ; \underbrace{\boldsymbol{y}\otimes\cdots\otimes\boldsymbol{y}}_{K \text{ times}} \right\rangle \end{aligned} \tag{5.2}$$

where $\phi_K(\boldsymbol{x})$ is computed as:

$$\phi_K(\boldsymbol{x}) = \prod_{k=1}^{K} \langle \boldsymbol{w}_k \ ; \ \boldsymbol{x}\rangle \tag{5.3}$$

In practice, we approximate the expectation of this quantity by using the sample mean over d sets of these random projectors. That is, we sample independent random matrices $\boldsymbol{W}_1, \boldsymbol{W}_2, ..., \boldsymbol{W}_K \in \mathbb{R}^{c\times d}$ and we compute the vector $\boldsymbol{\phi}_K(\boldsymbol{x}) \in \mathbb{R}^d$ that approximates the K-th moments of $\boldsymbol{x}$ with the following equation:

$$\boldsymbol{\phi}_K(\boldsymbol{x}) = \left(\boldsymbol{W}_1^\top \boldsymbol{x}\right) \odot \left(\boldsymbol{W}_2^\top \boldsymbol{x}\right) \odot \cdots \odot \left(\boldsymbol{W}_K^\top \boldsymbol{x}\right) \tag{5.4}$$

where $\odot$ is the Hadamard (element-wise) product. Thus, the inner product between the K-th moments is:

$$\langle \boldsymbol{x} \ ; \ \boldsymbol{y}\rangle^K \approx \frac{1}{d} \langle \boldsymbol{\phi}_K(\boldsymbol{x}) \ ; \ \boldsymbol{\phi}_K(\boldsymbol{y})\rangle \tag{5.5}$$

However, Random Maclaurin produces a consistent estimator independently of the analyzed distributions, and thus also encodes non informative high-order moment components. To ignore these non-informative components, the projectors $\boldsymbol{W}_k$ can be learned from the data. However, the high number of parameters in $\mathcal{O}(K^2cd)$ makes it difficult to learn a consistent estimator, as we empirically show in subsubsection 5.4.2.2. We solve this problem by computing the high-order moment approximation using the following recursion:

$$\boldsymbol{\phi}_k(\boldsymbol{x}) = \boldsymbol{\phi}_{k-1}(\boldsymbol{x}) \odot \left(\boldsymbol{W}_k^\top \boldsymbol{x}\right) \tag{5.6}$$

This last equation leads to the proposed cascaded architecture for *HORDE* summarized in Algorithm 1. We empirically show in subsubsection 5.4.2.2 that this recursive approach produces a consistent estimator of the informative high-order moment components.

Then, the *HORDE* regularizer consists in computing a DML-like loss function on each of the high-order moments, such that similar (respectively dissimilar) images have similar (respectively dissimilar) high-order moments:

$$\mathcal{L}_{HORDE} = \sum_{k=2}^{K} \mathcal{L}_k \left(\mathbb{E}_{\boldsymbol{x}\sim\mathcal{I}}[\boldsymbol{\phi}_k(\boldsymbol{x})], \mathbb{E}_{\boldsymbol{y}\sim\mathcal{J}}[\boldsymbol{\phi}_k(\boldsymbol{y})]\right) \tag{5.7}$$

In practice, we cannot compute the expectation $\mathbb{E}_{\boldsymbol{x}\sim\mathcal{I}}[\boldsymbol{\phi}_k(\boldsymbol{x})]$ since the distribution of $\boldsymbol{x}$ is unknown. We propose to estimate it using the empirical estimator:

$$\mathcal{L}_{HORDE}(\mathcal{I},\mathcal{J}) = \sum_{k=2}^{K} \mathcal{L}_k \left(\frac{1}{|\mathcal{I}|}\sum_{\boldsymbol{x}_i\in\mathcal{I}} \boldsymbol{\phi}_k(\boldsymbol{x}_i), \frac{1}{|\mathcal{J}|}\sum_{\boldsymbol{x}_j\in\mathcal{J}} \boldsymbol{\phi}_k(\boldsymbol{y}_j)\right) \tag{5.8}$$

Algorithm 1 High-order moments computation

Require: $\boldsymbol{W}_1, \ldots, \boldsymbol{W}_K$ sampled from $\{-1; +1\}$
Ensure: K first moments approximations
procedure ApproxMoments($\boldsymbol{x}$)
  $\boldsymbol{\phi}_2(\boldsymbol{x}) \leftarrow \frac{1}{\sqrt{d}} \left(\boldsymbol{W}_1^\top \boldsymbol{x}\right) \odot \left(\boldsymbol{W}_2^\top \boldsymbol{x}\right)$
  $k \leftarrow 3$
  while $k \leq K$ **do**
    $\boldsymbol{\phi}_k(\boldsymbol{x}) = \boldsymbol{\phi}_{k-1}(\boldsymbol{x}) \odot \left(\boldsymbol{W}_k^\top \boldsymbol{x}\right)$
    $k \leftarrow k + 1$
  end while
  return $\boldsymbol{\phi}_2(\boldsymbol{x}), \ldots, \boldsymbol{\phi}_K(\boldsymbol{x})$
end procedure

where $\{\boldsymbol{x}_i \in \mathcal{I}\}$ and $\{\boldsymbol{x}_j \in \mathcal{J}\}$ are the sets of deep features extracted from images $\mathcal{I}$ and $\mathcal{J}$.

Hence, the DML model is trained on a combination of a standard DML loss and the *HORDE* regularizer on pairs of images $\mathcal{I}$ and $\mathcal{J}$:

$$\mathcal{L}(\mathcal{I}, \mathcal{J}) = \mathcal{L}_{DML}(\mathcal{I}, \mathcal{J}) + \mathcal{L}_{HORDE}(\mathcal{I}, \mathcal{J}) \tag{5.9}$$

This can easily be extended to any tuple based loss function. In practice, we use the same DML loss function for *HORDE* ($\forall k, \mathcal{L}_k = \mathcal{L}_{DML}$).

Remark also that at inference time, the image representation $\boldsymbol{\phi}_1(\mathcal{I})$ consists only of the sample mean of the deep features:

$$\boldsymbol{\phi}_1(\mathcal{I}) = \frac{1}{|\mathcal{I}|} \sum_{\boldsymbol{x}_i \in \mathcal{I}} \boldsymbol{x}_i, \tag{5.10}$$

and the *HORDE* part of the model can be discarded.

5.3.2 Theoretical analysis

In this section, we show that optimizing distances between high-order moments is directly related to the Maximum Mean Discrepancy (MMD) [65] and the Wasserstein distance. We consider the Reproducing Kernel Hilbert Space (RKHS) $\mathcal{H}$ of distributions $f : \Omega \mapsto \mathbb{R}^+$ defined on the compact $\Omega \subset \mathbb{R}^c$, endowed with the Gaussian kernel $k(\boldsymbol{x}, \boldsymbol{y}) = e^{-\gamma \|\boldsymbol{x}-\boldsymbol{y}\|^2}$. An image is then represented as a distribution $\mathcal{I} \in \mathcal{H}$ from which we can sample a set of deep features $\{\boldsymbol{x}_i \in \Omega\}_i$. We denote $\mathbb{E}_{\boldsymbol{x}\sim\mathcal{I}}[\boldsymbol{x}] \in \mathbb{R}^c$ the expectation of $\boldsymbol{x}$ sampled from $\mathcal{I}$. The high-order moments are denoted using their vectorized forms, that is $\mathbb{E}_{\boldsymbol{x}\sim\mathcal{I}}[\boldsymbol{x}^{\otimes k}] \in \mathbb{R}^{c^k}$ where $\boldsymbol{x}^{\otimes 2} = \boldsymbol{x} \otimes \boldsymbol{x}, \boldsymbol{x}^{\otimes 3} = \boldsymbol{x} \otimes \boldsymbol{x} \otimes \boldsymbol{x}$, *etc.* By extension, we use $\mathbb{E}_{\boldsymbol{x}\sim\mathcal{I}}[\boldsymbol{x}^{\otimes 1}]$ for the mean. We assume that all moments exist for every distributions in $\mathcal{H}$ and we note, $\forall \mathcal{I} \in \mathcal{H}$:

$$\max_k \parallel \mathbb{E}_{\boldsymbol{x}\sim\mathcal{I}}[\boldsymbol{x}^{\otimes k}] \parallel^2 = K < \infty \tag{5.11}$$

Following [65], the MMD between two distributions $\mathcal{I}$ and $\mathcal{J}$ is expressed as:

$$\text{MMD}(\mathcal{I}, \mathcal{J}) = \sup_T \mathbb{E}_{\boldsymbol{x}\sim\mathcal{I}}[T(\boldsymbol{x})] - \mathbb{E}_{\boldsymbol{y}\sim\mathcal{J}}[T(\boldsymbol{y})] \tag{5.12}$$

The MMD searches for a transform T that maximizes the difference between the expectation of two distributions. Intuitively, a low MMD implies that both distributions are concentrated in the same regions of the feature space.

In the following theorem, we show that the distance over high-order moments is an upper-bound of the squared MMD (the proof mainly follows [65]):

Theorem 1. *There exists $A \in \mathbb{R}^{+*}$ such that, for every distributions $\mathcal{I}, \mathcal{J} \in \mathcal{H}$, the MMD is bounded from above by the p first moments of $\mathcal{I}$ and $\mathcal{J}$ by:*

$$MMD^2(\mathcal{I}, \mathcal{J}) \leq A \sum_{k=1}^{p} \left\| \mathbb{E}_{\boldsymbol{x} \sim \mathcal{I}}[\boldsymbol{x}^{\otimes k}] - \mathbb{E}_{\boldsymbol{y} \sim \mathcal{J}}[\boldsymbol{y}^{\otimes k}] \right\|^2 + 1 + o(\frac{\gamma^p K}{p!}) \tag{5.13}$$

Proof. As the MMD is a distance on the RKHS $\mathcal{H}$ [65], the square of the MMD can be re-written such as:

$$\mathrm{MMD}^2(\mathcal{I}, \mathcal{J}) = \|\mathbb{E}_{\boldsymbol{x} \sim \mathcal{I}}[\boldsymbol{\phi}(\boldsymbol{x})] - \mathbb{E}_{\boldsymbol{y} \sim \mathcal{J}}[\boldsymbol{\phi}(\boldsymbol{y})]\|^2_{\mathcal{H}} \tag{5.14}$$

where $\boldsymbol{\phi}$ is defined using the kernel trick $k(\boldsymbol{x}, \boldsymbol{y}) = \langle \boldsymbol{\phi}(\boldsymbol{x}) \ ; \ \boldsymbol{\phi}(\boldsymbol{y}) \rangle$. Then, we can approximate the Gaussian kernel using its Taylor expansion:

$$\begin{aligned} k(\boldsymbol{x}, \boldsymbol{y}) &= \exp\big(-\gamma \|\boldsymbol{x}\|^2 - \gamma \|\boldsymbol{y}\|^2\big) \exp\big(2\gamma \langle \boldsymbol{x} \ ; \ \boldsymbol{y} \rangle\big) \\ &= \exp\big(-\gamma \|\boldsymbol{x}\|^2 - \gamma \|\boldsymbol{y}\|^2\big) \sum_{k=0}^{+\infty} \frac{(2\gamma)^k}{k!} \langle \boldsymbol{x}; \boldsymbol{y} \rangle^k \\ &\leq 1 + \sum_{k=1}^{+\infty} a_k \left\langle \boldsymbol{x}^{\otimes k}; \boldsymbol{y}^{\otimes k} \right\rangle \end{aligned} \tag{5.15}$$

where $a_k = \frac{(2\gamma)^k}{k!} > 0$. Thus, we can define $\boldsymbol{\phi}$ as the direct sum of all weighted and vectorized moments:

$$\boldsymbol{\phi}(\boldsymbol{x}) = \bigoplus_{k=1}^{+\infty} \sqrt{a_k} \boldsymbol{x}^{\otimes k} \tag{5.16}$$

As all moments exist, we can swap the expectation and the direct sum. Moreover, since the sequence $a_k = \frac{(2\gamma)^k}{k!} \to 0$ when $k \to +\infty$ and the moments are bounded by K, the higher-order moment contributions become negligible compared to the p first moments. Thus, we have:

$$\begin{aligned} \mathrm{MMD}^2(\mathcal{I}, \mathcal{J}) &\leq 1 + \sum_{k=1}^{+\infty} a_k \|\mathbb{E}_{\boldsymbol{x} \sim \mathcal{I}}[\boldsymbol{x}^{\otimes k}] - \mathbb{E}_{\boldsymbol{y} \sim \mathcal{J}}[\boldsymbol{y}^{\otimes k}]\|^2 \\ &\leq A \sum_{k=1}^{p} \|\mathbb{E}_{\boldsymbol{x} \sim \mathcal{I}}[\boldsymbol{x}^{\otimes k}] - \mathbb{E}_{\boldsymbol{x} \sim \mathcal{J}}[\boldsymbol{x}^{\otimes k}]\|^2 + 1 + o(\frac{\gamma^p K}{p!}) \end{aligned}$$

where $A = \max_k a_k$. □

This result implies that regularizing high-order moments to be similar enforces similar images to have deep features sampled from similar distributions. Thus, deep features from similar images have a higher probability of being concentrated in the same regions of the feature space.

Next, we show a converse relation between high-order moments and the Wasserstein distance:

Theorem 2. *There exists $a \in \mathbb{R}^{+*}$ such that, for every distributions $\mathcal{I}, \mathcal{J} \in \mathcal{H}$, the squared Wasserstein distance is bounded from below by the p first moments of $\mathcal{I}$ and $\mathcal{J}$ by:*

$$W_1^2(\mathcal{I}, \mathcal{J}) \geq a \sum_{k=1}^{p} \left\| \mathbb{E}_{\boldsymbol{x} \sim \mathcal{I}}[\boldsymbol{x}^{\otimes k}] - \mathbb{E}_{\boldsymbol{y} \sim \mathcal{J}}[\boldsymbol{y}^{\otimes k}] \right\|^2 - o(\frac{\gamma^p}{p!}) \tag{5.17}$$

Proof. Similarly to the Theorem 1, we can lower-bound the Gaussian kernel using its Taylor expansion:

$$k(\boldsymbol{x}, \boldsymbol{y}) \geq \alpha \sum_{k=1}^{+\infty} a_k \left\langle \boldsymbol{x}^{\otimes k}; \boldsymbol{y}^{\otimes k} \right\rangle$$

where $\alpha = \exp(-2\gamma K)$ and $a_k = \frac{(2\gamma)^k}{k!} > 0$. Then, by using the definition of $\boldsymbol{\phi}$ from Equation 5.16, a lower-bound for the MMD is:

$$\text{MMD}^2(\mathcal{I}, \mathcal{J}) \geq \alpha \; a' \sum_{k=1}^{p} \|\mathbb{E}_{\boldsymbol{x}\sim\mathcal{I}}[\boldsymbol{x}^{\otimes k}] - \mathbb{E}_{\boldsymbol{y}\sim\mathcal{J}}[\boldsymbol{y}^{\otimes k}]\|^2 - o(\frac{\gamma^p K}{p!}) \tag{5.18}$$

where $a' = \min_k a_k$. Finally, the MMD is a lower-bound of the Wasserstein distance [159]:

$$\sqrt{K} W_1(\mathcal{I}, \mathcal{J}) \geq \text{MMD}(\mathcal{I}, \mathcal{J}) \tag{5.19}$$

By combining Equation 5.18 and Equation 5.19, we get the expected lower-bound:

$$W_1^2(\mathcal{I}, \mathcal{J}) \geq a \sum_{k=1}^{p} \|\mathbb{E}_{\boldsymbol{x}\sim\mathcal{I}}[\boldsymbol{x}^{\otimes k}] - \mathbb{E}_{\boldsymbol{y}\sim\mathcal{J}}[\boldsymbol{y}^{\otimes k}]\|^2 - o(\frac{\gamma^p}{p!}) \tag{5.20}$$

where $a = \frac{\alpha \; a'}{K}$. □

Hence, regularizing high-order moments to be dissimilar enforces dissimilar images to have deep features sampled from different distributions. As such, deep features are more distinctive as they are sampled from different regions of the feature space for dissimilar images. This is illustrated in Figure 5.1c ($p = 5$) compared to Figure 5.1a ($p = 1$).

5.4 Experiments

5.4.1 Comparison to state-of-the-art

We present the benefits of our method by comparing our results with the state-of-the-art on four datasets, namely CUB-200-2011 (CUB) [174], Cars-196 (CARS) [102], Stanford Online Products (SOP) [127] and In-Shop Clothes Retrieval (INSHOP) [115]. We report the Recall@K (R@K) on the standard DML splits associated with these datasets. Following standard practices, we use GoogleNet [163] as a backbone network and we add a fully connected layer at the end for the embedding. For CUB and CARS, we train *HORDE* using 5 high-order moments with 5 classes and 8 images per instance per batch. For SOP and INSHOP, we use 4 high-order moments with a batch size of 2 images and 40 different classes as there are classes with only 2 images in these datasets. We use 256×256 crops and the following data augmentation at training time: multi-resolution where the resolution is uniformly sampled in $[80\%, 180\%]$ of the crop size, random crop and horizontal flip. At inference time, we only use the images resized to 256×256. For *HORDE*, we use 8192 dimensions for all high-order moments and we fix all embedding dimensions to 512. Finally, we take advantage of the high-order moments at testing time by concatenating them together. To be fair with other methods, we reduce their dimensionality to 512 using a PCA. These results are annotated with a †.

First, we show in the upper part of Table 5.1 that *HORDE* significantly improves three popular baselines (contrastive loss, triplet loss and binomial deviance). These improvements allow us to claim state of the art results for single model methods on CUB with **58.3**% R@1 (compared to 57.1% R@1 for HTL [57]) and second best for CARS.

We also present ensemble method results in the second part of Table 5.1. We show that *HORDE* is also a benefit to ensemble methods by improving ABE [98] by 2.7% R@1 on CUB and 7.2% R@1 on CARS. To the best of our knowledge, this allows us to outperform the state of the art methods on both datasets with **62.7**% R@1 on CUB and **86.4**% R@1 on CARS, despite our implementation of ABE under-performing compared to the results reported in [98].

Note that both single models and ensemble ones are further improved by using the high-order moments at testing: +1.1% on CUB and +1.7% on CARS for the single models + *HORDE* and +1.2% on CUB and +1.6% on CARS for ABE + *HORDE*.

		Cub-200-2011				Cars-196			
Backbone	R@	1	2	4	8	1	2	4	8
Loss functions or mining strategies									
GoogleNet	Angular loss [175]	54.7	66.3	76.0	83.9	71.4	81.4	87.5	92.1
	HDML [202]	53.7	65.7	76.7	85.7	79.1	87.1	92.1	95.5
	DAMLRMM [193]	55.1	66.5	76.8	85.3	73.5	82.6	89.1	93.5
	DVML [111]	52.7	65.1	75.5	84.3	82.0	88.4	93.3	96.3
	HTL [57]	57.1	68.8	78.7	86.5	81.4	88.0	92.7	95.7
	contrastive loss (Ours)	55.0	67.9	78.5	86.2	72.2	81.3	88.1	92.6
	contrastive loss + *HORDE*	57.1	69.7	79.2	87.4	76.2	85.2	90.8	95.0
	Triplet loss (Ours)	50.5	63.3	74.8	84.6n	65.2	75.8	83.7	89.4
	Triplet loss + *HORDE*	53.6	65.0	76.0	85.2	74.0	82.9	89.4	93.7
	Binomial Deviance (Ours)	55.9	67.6	78.3	86.4	78.2	86.0	91.3	94.6
	Binomial Deviance + *HORDE*	**58.3**	**70.4**	**80.2**	**87.7**	81.5	**88.5**	92.7	95.4
	Binomial Deviance + *HORDE*†	**59.4**	**71.0**	**81.0**	**88.0**	**83.2**	**89.6**	**93.6**	**96.3**
BN-Inception	Multi-similarity loss [183]	65.7	77.0	**86.3**	**91.2**	84.1	90.4	94.0	96.5
	contrastive loss + *HORDE*	**66.3**	76.7	84.7	90.6	94.5	90.3	94.1	96.3
	contrastive loss + *HORDE*†	**66.8**	**77.4**	85.1	91.0	**86.2**	**91.9**	**95.1**	**97.2**
Ensemble Methods									
GoogleNet	HDC [197]	53.6	65.7	77.0	85.6	73.7	83.2	89.5	93.8
	BIER [129]	55.3	67.2	76.9	85.1	78.0	85.8	91.1	95.1
	A-BIER [130]	57.5	68.7	78.3	86.2	82.0	89.0	93.2	96.1
	ABE [98]	60.6	71.5	79.8	87.4	85.2	90.5	94.0	96.1
	ABE (Ours)	60.0	71.8	81.4	88.9	79.2	87.1	92.0	95.2
	ABE + *HORDE*	**62.7**	**74.3**	**83.4**	**90.2**	**86.4**	**92.0**	**95.3**	**97.4**
	ABE + *HORDE*†	**63.9**	**75.7**	**84.4**	**91.2**	**88.0**	**93.2**	**96.0**	**97.9**

Table 5.1: Comparison with the state-of-the-art on Cub-200-2011 and Cars-196 datasets. Results in percents. † means that the test scores are computed using all the high-order moments (concatenation + PCA to the embedding size).

		Stanford Online Products			In-Shop Clothes Retrieval			
Backbone	R@	1	10	100	1	10	20	30
GoogleNet	Angular loss [175]	70.9	85.0	93.5	-	-	-	-
	HDML [202]	68.7	83.2	92.4	-	-	-	-
	DAMLRMM [193]	69.7	85.2	93.2	-	-	-	-
	DVML [111]	70.2	85.2	93.8	-	-	-	-
	HTL [57]	**74.8**	**88.3**	**94.8**	80.9	94.3	95.8	97.2
	Binomial Deviance (Ours)	67.4	81.7	90.2	81.3	94.2	95.9	96.7
	Binomial Deviance + *HORDE*	72.6	85.9	93.7	**84.4**	**95.4**	**96.8**	**97.4**
BN-Inception	Multi-similarity loss [183]	78.2	90.5	96.0	89.7	97.9	98.5	98.8
	contrastive loss + *HORDE*	**80.1**	**91.3**	**96.2**	**90.4**	97.8	98.4	98.7

Table 5.2: Comparison with the state-of-the-art on Stanford Online Products and In-Shop Clothes Retrieval. Results in percents.

Furthermore, we show that *HORDE* generalizes well to large scale datasets by reporting results on SOP and INSHOP in Table 5.2. *HORDE* improves our baseline binomial deviance by 5.2% R@1 on SOP and 3.1% R@1 on INSHOP. This improvement allows us to claim state of the art results for single model methods on INSHOP with **84.2**% R@1 (compared to 80.9% R@1 for HTL) and second best on SOP with 72.6% R@1 (compared to 74.8% R@1 for HTL). Remark also that *HORDE* outperforms HTL on 3 out of 4 datasets.

We also report some results with the BN-Inception [79]. Our model trained with *HORDE* and contrastive loss leads to similar results compared to the recent MS loss with mining [183] on smaller datasets while on larger datasets it outperforms it by 1.9% on SOP and by 0.7% on INSHOP. By using the high-order moments are testing, performances are further increased and outperforms MS loss with mining by 1.1% on CUB and by 2.1% on CARS.

Finally, we show some example queries and their nearest neighbors in Figure 5.3 on the test split of CUB.

k	1	2		3			4				5					6					
n	1	1	2	1	2	3	1	2	3	4	1	2	3	4	5	1	2	3	4	5	6
R@1	55.9	**57.8**	58.6	56.8	58.0	56.9	**57.8**	58.8	57.6	56.1	57.4	57.7	56.8	56.3	53.3	57.4	57.9	57.1	55.6	54.4	50.7
R@2	67.6	69.5	70.4	68.1	69.4	68.7	69.2	70.6	70.0	68.5	68.8	69.9	69.3	68.1	65.4	**69.9**	70.6	70.5	68.9	66.2	63.0
R@4	78.3	79.0	79.8	78.3	78.8	78.1	78.6	79.9	79.2	78.1	78.7	78.8	79.2	78.0	75.9	**79.4**	80.0	79.9	78.7	76.5	74.0
R@8	86.4	86.7	87.2	86.2	86.7	86.6	86.5	87.2	87.0	85.5	**87.0**	87.1	87.1	86.5	84.2	86.9	87.4	87.4	86.7	85.4	82.5

Table 5.3: Impact of the high order moments as regularizers. We report the Recall@K on CUB. k is the number of chosen orders at training time, and n is the order used at testing time to evaluate performances. $k = n = 1$ is the baseline.

k	1	2		3			4				5					6					
n	1	1	2	1	2	3	1	2	3	4	1	2	3	4	5	1	2	3	4	5	6
R@1	55.9	57.0	53.4	57.6	54.7	50.6	57.9	55.4	52.3	47.6	58.1	55.9	53.1	48.4	43.7	**58.4**	55.7	52.9	47.8	43.9	40.5
R@2	67.6	68.3	65.4	69.9	67.0	63.0	69.5	67.1	65.0	60.2	**70.3**	67.7	65.0	60.8	56.0	69.9	67.6	64.9	59.9	56.0	53.0
R@4	78.3	78.3	75.8	79.1	76.8	73.6	79.6	77.5	75.2	71.0	**79.9**	78.2	75.5	72.8	67.2	79.8	78.0	75.6	70.2	67.2	64.7
R@8	86.4	86.2	84.2	87.0	84.7	82.4	87.1	85.8	83.6	80.2	87.1	85.2	83.9	81.7	78.0	**87.3**	85.6	83.8	79.6	77.5	75.2

Table 5.4: Impact of the high order moments when all parameters are trained. We report the Recall@K on CUB. k is the number of chosen orders at training time, and n is the order used at testing time. $k = n = 1$ is the baseline.

5.4.2 Ablation studies

In this section, we provide an ablation study on the different contributions of this work. We perform 3 experiments on the CUB dataset [174]. The first experiment shows the impact of high-order regularization on a standard architecture while the high-order moments are consistently approximated using the Random Maclaurin approximation. The second experiment illustrates the benefit of learning the high-order moments projection matrices. The last experiment confirms the statistical consistency of our cascaded architecture when the parameters are learned.

5.4.2.1 Regularization effect

In this section, we assess the regularization impact of *HORDE*. To that aim, we use the baseline detailed in subsection 5.4.1 and we train the architecture with a number of high-order moments varying from 2 to 6. In this first experiment, the computation of the high-order moments does not rely on the cascade computation approach of Equation 5.6. Instead, the matrices to approximate the high-order moments are untrainable and sampled using the Random Maclaurin method of Equation 5.4. Remark also that the embedding layers on all high-order moments are not added. We use the binomial deviance loss with the standard parameters [173]. The results are shown in Table 5.3.

First, we can see that *HORDE* consistently improves the baseline from 1% to 2% in R@1. These results corroborate the insights of our theoretical analysis in section 5.3 and also provide a quantitative evaluation of the behavior observed in Figure 5.1 on the retrieval ranking. When considering the high-order moments as representations, we observe improved results with respect to the baseline for orders 2 and 3. Note however that the reported high-order results are not comparable to the first order as the similarity measure is computed on the 8192 dimensional representations. While adding orders higher than 2 does not seem interesting in terms of performances, we found that the training process is more stable with 5 or 6 orders than only 2. This is observed in practice by measuring the Recall@K with $K \geq 8$ which tend to vary less between training steps. Moreover on the CUB dataset, while the baseline requires around 6k steps to reach the best results, we usually need 1k steps less to reach higher accuracy with *HORDE*.

5.4.2.2 Statistical consistency

To evaluate the impact of estimating only informative high-order moments, we first train the projection matrices and the embeddings but without the cascade architecture and report the results in Table 5.4.

k	1	2		3			4				5					6					
n	1	1	2	1	2	3	1	2	3	4	1	2	3	4	5	1	2	3	4	5	6
R@1	55.9	57.0	53.4	57.9	56.1	54.2	57.6	55.4	54.3	53.0	**58.3**	56.3	56.0	54.7	52.4	57.9	56.6	55.8	55.0	53.9	51.6
R@2	67.6	68.3	65.4	69.4	67.9	66.2	69.3	67.2	66.0	65.2	**70.4**	68.7	68.1	66.9	64.7	69.5	68.8	68.3	67.7	65.2	64.0
R@4	78.3	78.3	75.8	79.2	77.8	76.4	79.5	77.2	77.0	75.8	**80.2**	78.5	78.3	76.9	75.6	79.6	76.6	77.9	77.9	75.3	74.4
R@8	86.4	86.2	84.2	86.6	85.3	84.4	87.1	85.6	84.4	84.1	**87.7**	86.3	86.0	85.4	84.1	87.0	86.4	85.6	84.8	84.0	83.7

Table 5.5: Impact of the cascaded architecture when all parameters are trained using Algorithm 1. We report the Recall@K on CUB. k is the number of chosen orders at training time, and n is the order used at testing time. $k = n = 1$ is the baseline.

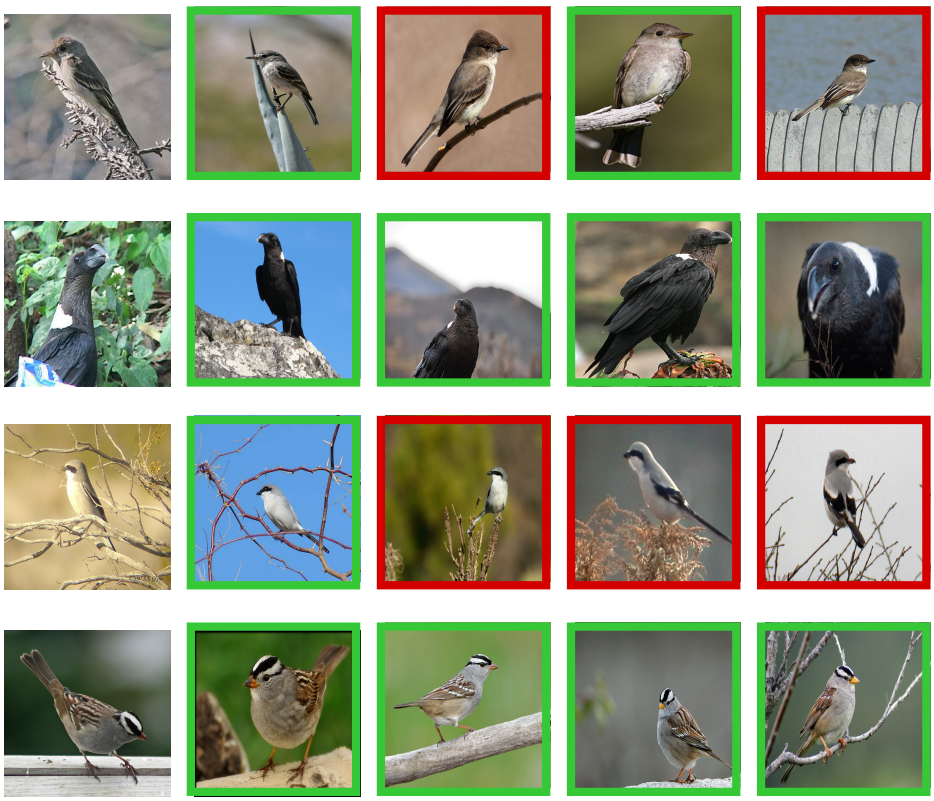


Figure 5.3: Qualitative results on CUB for *HORDE*. Correct results are highlighted green (incorrect in red).

In this second experiment, we empirically show that such scheme also increases the baseline by at least 1% in R@1. Notably, by focusing on the most informative high-order moment components, *HORDE* further improves the performances of the untrainable *HORDE* from 57.8% to 58.4%. However, the retrieval performances of the high-order representations are heavily degraded compared to Table 5.3. We interpret these results as an inconsistent estimations of the high-order moments due to overfitting the model. For example, the 6% loss in R@1 for the third-order moment between the first and the second experiments suggests a reduced interest for higher-order moments.

For the third experiment, we report the results of our cascaded architecture in Table 5.5. Interestingly, the high-order moments computed from the cascaded architecture perform almost identically to those computed from the untrained method Table 5.3 but with a smaller dimension. Moreover, we keep the performance improvement of the second experiments of Table 5.4. This confirms that the proposed cascaded architecture does not overfit its estimations of the high-order moments while still improving the baseline. Finally, this cascaded architecture only produces a small computational overhead during the training compared to the architecture without the cascade.

5.5 Discussion

In this section, we discuss about the proposed regularization method according to the following points:

- Differences with Magnet loss [143] and DVML [111]
- Empirical estimators of divergence between distributions
- The image distribution hypothesis
- The computation of the image representation
- The Random Maclaurin factorization
- The extension of *JCF*to high-order moments

5.5.1 Magnet loss and DVML

Recent approaches also consider a distribution analysis for deep metric learning [143, 111]. Contrarily to us, they only consider the distribution of the representations to design a loss function or a hard negative generator but they do not take into account the distribution of the underlying deep features. Consequently, they do not address the scattering problem. More precisely, Magnet loss [143] proposes to better represent a given class manifold by learning a K-mode distribution instead of the standard uni-mode assumption. To that aim, the per-class distribution is approximated using K-means clustering. The proposed loss tries to minimize the distance between a representation and its nearest class mode and tries to maximize the distance between all modes of all other classes. However, since the magnet loss is directly applied to the sample means of the deep features, it leads to the scattering problem illustrated in Figure 5.1. In DVML [111], the authors assume that the representations follow a per-class Gaussian distribution. They propose to estimate the parameters of these distributions using a variational auto-encoder approach. Then, by sampling from a Gaussian distribution with the learned parameters, they are able to generate artificial hard samples to train the network. However, no assumption is made on the distribution of the deep features, which leads to the scattering problem illustrated in Figure 5.1 (see also [111], Figure 1). In contrast, we show that focusing on the distribution of the deep features reduces the scattering problem and improves performances of DML architectures.

5.5.2 Empirical estimators and the image distribution hypothesis

Instead of optimizing the distance between the high-order moments, we can also exploit the empirical estimator of the divergence between distributions. For example, the MMD can be estimated using the empirical estimator from [65, 18]:

$$\mathrm{MMD}(\mathcal{I},\mathcal{J}) = \frac{1}{|\boldsymbol{\mathcal{I}}|^2}\sum_{\boldsymbol{x}_i,\boldsymbol{x}_j\in\boldsymbol{\mathcal{I}}} k(\boldsymbol{x}_i,\boldsymbol{x}_j) + \frac{1}{|\boldsymbol{\mathcal{J}}|^2}\sum_{\boldsymbol{y}_i,\boldsymbol{y}_j\in\boldsymbol{\mathcal{J}}} k(\boldsymbol{y}_i,\boldsymbol{y}_j) - \frac{2}{|\boldsymbol{\mathcal{I}}||\boldsymbol{\mathcal{J}}|}\sum_{\boldsymbol{x}_i\in\boldsymbol{\mathcal{I}},\boldsymbol{y}_j\in\boldsymbol{\mathcal{J}}} k(\boldsymbol{x}_i,\boldsymbol{y}_j) \quad (5.21)$$

where $k(\cdot,\cdot)$ is the Gaussian kernel. Following the proofs of Theorem 1 and Theorem 2, optimizing the empirical estimator of the MMD is a way to implicitly optimize all high-order moments of the deep local features. However, there are two main advantages of optimizing high-order moments instead of the aforementioned empirical estimator. on the one hand, the explicit optimization of the high-order moments has more impact than the implicit optimization: due to the Gaussian kernel,the weights for the p-th high-order moment are proportional to $\frac{\sigma^p}{p!}$. Thus, except if we use large standard deviation, the third order moment is already negligible compared to the first and the second order moments. Also, we have empirically shown that matching higher-order moments further increases performances of the deep network which experimentally confirm the benefit of the explicit matching. On the other hand, enforcing the distribution to be very similar is a too strong hypothesis on the distributions of the local deep features. Indeed, different images that we suppose to have the same distribution might be two images with the same object but with a different background, occlusion,

etc. As a consequence, we also enforce the features of the background to be discriminant for the class. By considering the explicit match of high-order moments, and by learning the projection matrices to reduce the dimensionality, these features can be discarded during the learning step. Thus, by construction, the explicit matching of high-order moments can only enforce parts of the distributions to be the same instead of the whole distribution which includes the local features from the background.

5.5.3 Image representation

In the case of most recent deep network architectures, the average of the deep features is used as the image representation. Interestingly, *HORDE* naturally extends this image representation by using the higher-order moments as additional representations and by matching them. Moreover, the scattering problem can be observed even on stronger image representations such as ABE [98]. Indeed, we have shown in Table 5.1 that even a stronger image representation such as ABE can be improved from 60.6% in Recall@1 to 62.7% with *HORDE*. This also shows that better image representations are still subjected to the scattering problem and that *HORDE* should benefit for other image representations. Also, this phenomenon still appears with stronger deep local features that uses batch-normalization [79] and *HORDE* still increases performances.

5.5.4 Random Maclaurin factorization

Because of the high dimensionality of high-order moments, we have relied on the Random Maclaurin factorization to get smaller representations that approximate these high-order moments. In this section, we discuss about the choice of the Random Maclaurin algorithm compared to the factorizations presented in section 3.4[1] but also the proposed one in *JCF* (see section 4.3[2]). The first advantage of the Random Maclaurin algorithm is that, in expectation, the representations give the expected quantities. This implies that our experiment in Table 5.3 which considers non-trainable weights leads to consistent approximation. That is, this approximation in expectation gives the desired quantity without bias and that the estimator's variance decreases in $\frac{1}{d}$ where d is the dimensionality of the approximated representation. This ensures that our experimental confirmation of Theorem 1 and Theorem 2 is correct, up to the approximation error. Thus, the optimization of the high-order moments, which is directly related to the optimization of divergences between distributions, has a beneficial impact on the deep local features by improving performances.

The second advantage of Random Maclaurin is that it is very easy to implement, fast to compute but also that it is the factorization that needs the smaller number of parameters when we train the weights of the approximation. Indeed, the additional computation time added by the high-order moments approximation is small compared to the whole network. *E.g.*, it represents only around 30% of the additional training time when the network in trained with the six first high-order moments. Also, the evaluation part is kept the same as these high-order moments can be discarded after the training. Furthermore, the Random Maclaurin algorithm is very easy to implement by using K 1×1 convolutions (one for each order), Hadamard product between these feature maps and by aggregating each feature maps per high-order moment using global average pooling. It is also very efficient with current deep learning framework because it is relies on the aforementioned well implemented linear algebra operations.

Instead of using the random projection in the Random Maclaurin algorithm, we can also consider the *JCF* factorization proposed in section 4.3. Indeed, *JCF* can easily be extended to any K-th order moment by re-writing Equation 4.15:

$$z_i(\boldsymbol{x}) = \prod_{k=1}^{K} \langle \boldsymbol{p}_{k,i} \ ; \ \boldsymbol{h}(\boldsymbol{x}) \otimes \boldsymbol{x} \rangle \tag{5.22}$$

[1] Chapter "Second-Order Pooling"
[2] Chapter "Dictionary learning", section on *JCF*.

where $\boldsymbol{\mathcal{P}} = \{\boldsymbol{p}_{k,i} \in \mathbb{R}^{Nd}\}$ is a tensor in $\mathbb{R}^{K \times D \times Nd}$ composed by the parameters of the model. $\boldsymbol{p}_{k,i}$ can also be factorized following Equation 4.18 which leads to the extension of Equation 4.22 to the K-th order moments factorization:

$$z_i(\boldsymbol{x}) = \prod_{k=1}^{K} \left(\boldsymbol{h}(\boldsymbol{x})^T \boldsymbol{U}_{k,i}^T \boldsymbol{x} \right) \tag{5.23}$$

where $\boldsymbol{U}_{k,i} \in \mathbb{R}^{N \times d}$ is a learnable projection for the i-th dimension of the output representation of the K-th order moment. Using such factorization leads to a very large number of parameters in $O(K^2NdD)$ without any factorization: for a given order K, we need KD matrices $\boldsymbol{U}_{k,i} \in \mathbb{R}^{N \times d}$ for the approximation. The recursive formulation of random Maclaurin used in Equation 5.6 can also be considered to reduce the number of parameters from $O(K^2DNd)$ to $O(KDNd)$. Also, the dictionary entries can be shared across the high-order moments to further reduce the number of parameters and the computation. Compared to the Random Maclaurin algorithm, the number of parameters in *JCF* is still very high and the *JCF* extension discussed in subsection 4.3.9 should solve this issue.

5.6 Conclusion

In this work, we have presented *HORDE*, a new deep metric learning regularization scheme which improves the distinctiveness of the deep features. This regularizer, based on the optimization of the distance between the distributions of the deep features, provides consistent improvements to a wide variety of popular deep metric learning methods. We give theoretical insights that show *HORDE* upper-bound the Maximum Mean Discrepancy and lower-bound the Wasserstein distance. The computation of high-order moments is tackled using a trainable Random Maclaurin factorization scheme which is exploited to produce a cascaded architecture with small computation overhead. We have discussed in depth some concern about the implementation that includes the optimization of high-order moments versus the use of an empirical estimator of a divergence between distribution, the "image as a distribution" hypothesis, the Random Maclaurin factorization and the connection with our previous work such as *JCF*. Finally, *HORDE* achieves very competitive performances on four well known datasets.

6

MIRAGE: Improving Deep Metric Learning with Virtual Classes and Examples Mining

In this chapter, we present an example generation method that leverages virtual classes composed solely of generated examples to tackle the problem of label ambiguity arising from hard negative generation.

Contents

6.1 Prologue **88**
6.2 Introduction **88**
6.3 Method overview **89**
6.3.1 *MIRAGE* overview 89
6.3.2 Deep metric learning 90
6.3.3 Training class example generation 90
6.3.4 Virtual class example generation 91
6.3.5 *MIRAGE* architecture 91
6.4 Experiments **91**
6.4.1 Prototype visualization 91
6.4.2 Example generation ablation 92
6.4.3 Comparison to state-of-the-art 93
6.5 Discussion **96**
6.6 Conclusion **97**

6.1 Prologue

Research paper detailed:

- **Pierre Jacob**, David Picard, Aymeric Histace, Edouard Klein. *"Improving Deep Metric Learning with Virtual Classes and Examples Mining"*. In submission, Nov. 2019.

Context:
We provide in this chapter an example generation method in the context of supervised deep metric learning. Deep metric learning (DML) is an important, yet challenging task in the Computer Vision community, with numerous applications such as multi-modal retrieval [26, 185], face verification [152] or person re-identification [112]. DML methods intend to learn an embedding space, where visually-related images (*e.g.*, two different birds from the same breed) have similar representations, while unrelated images (*e.g.*, two different breeds of crows from North America and Europe) have dissimilar representations. To learn this embedding space, recent contributions focus on three main points: (1) loss functions to improve generalization [183], (2) ensemble methods to tackle the embedding space diversity [129] and (3) hard example mining strategies to resume the training when randomly sampling informative tuples becomes nearly impossible [193]. Also, recent contributions consider example generation methods to replace the mining step. By generating hard negative examples online, the mining step is avoided and the training is not slow to find the informative triplets in the dataset.

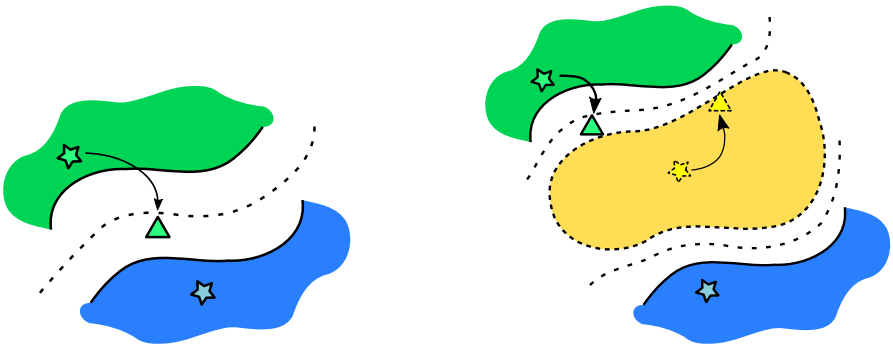
In this work, we propose an example generation strategy that solve what we call *label ambiguity*. Label ambiguity appears in adversarial example generation when the generator is too strong and easily fools the discriminator. Such issue leads to the generation of examples that are outside the class manifold, and maybe inside another class manifold, but with the label of the first class. As a consequence, training a network for metric learning with these examples tend to destroy the embedding space because most of the computed gradients come from these examples. To solve this problem, we propose *MIRAGE*, an example generation method that leverages virtual classes composed solely of generated examples to tackle the problem of label ambiguity arising from hard negative generation. By training virtual classes to be placed between real ones, we create buffer areas. By doing so, examples that lie inside these buffer areas are generated without any label ambiguity because they are simply sampled within these virtual class manifolds. Empirically, we show that *MIRAGE* improves strong baseline for a variety of loss functions and backbone network.

In *MIRAGE*, the generation procedure is both inspired by variational approaches and by adversarial approaches which leads to both benefits from these methods. Thus, we discuss the related work in example generation such as HTG [201], DAML [50], DVML [111] and HDML [202]. Additionally, we show that the proposed training of virtual classes has a good behavior in practice (classes are well placed between training classes, both in low and high dimension) but also good performances.

Contributions:
The contribution of this work is an example generation method that leverages virtual classes composed solely of generated examples to tackle the problem of label ambiguity arising from hard negative generation. This method is easy to implement with a good behavior in practice and leads to competitive performances on most of the standard metric learning datasets.

6.2 Introduction

Example generation has recently been proposed as a hard negative mining strategy. In this case, a generator and the metric learning network are trained together to provide informative tuples using either VAEs [111] or GANs [50, 201, 202]. In the case of VAEs, a large amount of examples is generated by sampling with respect to the training sample distribution estimated from the data. Usually, this leads to sampling inside the class manifolds and rarely produces hard negative examples. Such variational approaches are interesting in the case of few training samples per class but they are not well suited for mining informative examples at later training stages. On the opposite, GAN-based approaches generate discriminative examples. However, adversarial generators are difficult to tune



(a) Standard hard negative generation

(b) Virtual class (dashed line manifold) hard negative generation

Figure 6.1: Hard negative generation. The standard hard negative generation on Figure 6.1a can lead to incorrect label if the generated example is sampled beyond the boundary of the class manifold. By adding a virtual class between the training classes on Figure 6.1b, hard negative examples generated beyond the boundary of the class manifold are still within the correct classes with respect to training classes.

due to the contrary objectives of the DML network and the adversarial learning of the generator. On the one hand, if the adversarial loss is much lower than the DML loss, the generated examples tend to be at the center of the class manifold and the method faces the same problems as VAE generators. On the other hand, if the adversarial loss is much higher than the DML loss, some examples can be generated beyond the boundary of the class manifolds and lead to label ambiguity as illustrated on Figure 6.1a. The mining strategy then produces examples with incorrect labels with respect to the training classes.

As the main contribution of this paper, we propose *MIRAGE*, a method that leverages virtual classes composed solely of generated examples to tackle the problem of label ambiguity arising from hard negative generation. Virtual classes play the role of buffer areas as shown on Figure 6.1b. Hard negative examples that lie between the training class manifolds are generated inside these buffer areas, without any label ambiguity, by sampling the virtual classes. In addition to solving the problem of label ambiguity, virtual classes example generation leads to better generalization capabilities: The metric learning network has better results on unseen classes than other adversarial approaches [50, 201, 202].

The chapter is organized as follow: in section 6.3, we expose the core aspects of *MIRAGE* and its simple implementation. We experimentally show that *MIRAGE* indeed produces buffer areas between the training classes (see subsection 6.4.1) and we perform an ablation study of the different aspects of the method (see subsection 6.4.2). Finally, in subsection 6.4.3, we show that our method improves over other sample generation methods on four DML datasets (Cub-200-2011, Cars-196, Stanford Online Products and In-Shop Clothes Retrieval), and obtains results comparable to the state-of-the-art.

6.3 Method overview

6.3.1 *MIRAGE* overview

MIRAGE is designed to improve deep metric learning by using the following core aspects:

DML training. Like any other DML method, *MIRAGE* uses a deep neural network to embed feature vectors into a latent representation space where visually-related images have similar representations and where unrelated images have dissimilar representations. We use the standard metric learning approach which extracts deep local features using a backbone network (*e.g.*, GoogleNet [163]

or BN-Inception [79]), computes a feature vector (*e.g.*, using an average pooling) and projects it into an embedding space in order to learn the metric.

Training class sample generation. As it is done in variational approaches, *MIRAGE* generates artificial examples from the training classes in order to provide a better sampling of each training class manifold. By doing so, class manifolds are filled with synthetic examples. These generated examples are added to the mini-batch along real examples in order to have larger batches from which informative tuples can be sampled. These additional tuples are then used to train the DML model.

Virtual class hard negative sample generation. Similarly to adversarial sample generation, *MIRAGE* also generates hard negative examples. However, current hard negative generation are prone to label ambiguity (see Figure 6.1a). To tackle this issue, we add virtual classes between training classes that play the role of buffer areas (see Figure 6.1b). Hard negative examples are consequently generated inside these buffer areas by sampling within these virtual class manifolds. Similarly to training class generation, these examples are added to the mini-batch. We experimentally show that it leads to better performances than other generation-based methods.

6.3.2 Deep metric learning

The first part of a DML network is to extract a global feature vector $\boldsymbol{f}_t$. *E.g.*, we use GoogleNet [163] followed by a global average pooling to compute $\boldsymbol{f}_t$. The feature vectors are then projected into the embedding space with $\boldsymbol{W}$ where their corresponding examples are denoted $\boldsymbol{x}_t$. In practice, all examples $\boldsymbol{x}_t$ are ℓ_2-normalized to ease the optimization. We note E the function that transforms a feature vector $\boldsymbol{f}_t$ into an example $\boldsymbol{x}_t$ using the following equation:

$$\boldsymbol{x}_t = E(\boldsymbol{f}_t) = \frac{\boldsymbol{W}^\top \boldsymbol{f}_t}{\|\boldsymbol{W}^\top \boldsymbol{f}_t\|_2} \tag{6.1}$$

To train the network, we rely on standard metric learning loss functions such as the contrastive loss [33], the triplet loss [152] or the binomial loss [173]. As proposed by [122], we use a class representation prototype $\boldsymbol{p}_t$ to accelerate the training. E and $\boldsymbol{p}_t$ are trained together using a DML loss (triplet, contrastive, *etc.*) denoted $\mathcal{L}_{\text{DML}}$.

6.3.3 Training class example generation

To generate examples from the training classes, *MIRAGE* relies on a conditional generator G that is designed to produce an artificial example $\widetilde{\boldsymbol{x}}_t$ from the prototype $\boldsymbol{p}_t$ of class t and a Gaussian noise $\boldsymbol{\epsilon} \sim \mathcal{N}(\boldsymbol{0}; \boldsymbol{\Sigma})$, as follows:

$$\widetilde{\boldsymbol{x}}_t = E\left(G\left(\frac{\boldsymbol{p}_t + \boldsymbol{\epsilon}}{\|\boldsymbol{p}_t + \boldsymbol{\epsilon}\|_2}\right)\right) \tag{6.2}$$

$\widetilde{\boldsymbol{x}}_t$ is then used as a training example to optimize the loss function described in the previous section.

To train the generator G, we use a reconstruction loss by computing the ElasticNet loss between a feature vector $\boldsymbol{f}_t$ extracted from a real image and a feature vector generated by feeding the generator G with $E(\boldsymbol{f}_t)$, as follows:

$$\mathcal{L}_{\text{rec}} = \|\boldsymbol{f}_t - G(E(\boldsymbol{f}_t))\|_1 + \|\boldsymbol{f}_t - G(E(\boldsymbol{f}_t))\|_2^2 \tag{6.3}$$

6.3.4 Virtual class example generation

To generate hard negative examples, we consider a set of virtual classes associated with prototypes $\boldsymbol{p}_v$. Examples are generated from these prototypes exactly like if they are from training classes. To produce hard negative samples, we encourage the prototypes and the generator to output realistic samples between the training classes. To that end, we use a discriminative classifier D. D is trained using binary cross-entropy to distinguish between real and generated samples (with output D_g). D is also trained using categorical cross-entropy to predict classes (with output D_c). The combined loss $\mathcal{L}_{\text{adv}}$ for training D is a two head classification loss based on cross-entropy:

$$\mathcal{L}_{\text{adv}} = \underbrace{-\log(D_g(E(\boldsymbol{f}_t))) - \log\left(1 - D_g\left(E\left(G\left(\frac{\boldsymbol{p}_v + \boldsymbol{\epsilon}}{\|\boldsymbol{p}_v + \boldsymbol{\epsilon}\|_2}\right)\right)\right)\right)}_{\text{binary cross-entropy on real/generated samples}}$$
$$\underbrace{-y_t \log(D_c(E(\boldsymbol{f}_t))) - y_v \log\left(D_c\left(E\left(G\left(\frac{\boldsymbol{p}_v + \boldsymbol{\epsilon}}{\|\boldsymbol{p}_v + \boldsymbol{\epsilon}\|_2}\right)\right)\right)\right)}_{\text{categorical cross-entropy on the classes}}$$

where y_t and y_v are the class labels of the prototypes $\boldsymbol{p}_t$ and $\boldsymbol{p}_v$ respectively.

To encourage the generator to output realistic samples that are between the training classes, G is trained to maximize $\mathcal{L}_{\text{adv}}$, with D being fixed. By optimizing over D_g, the generator is encouraged to output generated samples that are indistinguishable from real samples. By optimizing over D_c, the generator is encouraged to output samples at the boundaries of the classes (*i.e.*, in the buffer area described in Figure 6.1b). Just like the training class sample generation, virtual class sample generation is used to populate the mini-batches used for training using $\mathcal{L}_{\text{DML}}$.

6.3.5 *MIRAGE* architecture

We describe the implementation of the *MIRAGE* architecture in Figure 6.2. A set of deep local features is first extracted from the image using a backbone network such as GoogleNet [163] or BN-Inception [79]. These local features are then aggregated into feature vectors using an average pooling. They are followed by the encoder E which is composed of a single fully-connected layer without bias followed by a ℓ_2 normalization. A prototype $\boldsymbol{p}$ is used for each class and is represented by a star, either in plain lines for training classes or in dashed lines for virtual classes. The generator G is composed of two fully-connected layers with ReLU activation. The discriminator D is composed of a fully-connected layer with ReLU activation which is followed by two fully-connected layers: One with sigmoid activation for the binary classification of real or virtual feature vectors and one with softmax activation for the class prediction.

To train *MIRAGE*, we generate mini-batches composed of training examples $\boldsymbol{x}_t$, generated examples from the training class $\widetilde{\boldsymbol{x}}_t$ and generated examples from virtual classes $\widetilde{\boldsymbol{x}}_v$. The ratio of training examples and generated examples in the mini-batch corresponds to how much each aspect of *MIRAGE* is used. This ratio is investigated in the ablation studies. The backbone network, the encoder E and the prototypes are trained together using the entire mini-batch minimizing the metric learning loss function $\mathcal{L}_{\text{DML}}$ from subsection 6.3.2. The generator G is trained on $\widetilde{\boldsymbol{x}}_t$ minimizing $\mathcal{L}_{\text{rec}}$ from subsection 6.3.3 and on $\widetilde{\boldsymbol{x}}_v$ maximizing $\mathcal{L}_{\text{adv}}$ from subsection 6.3.4. Finally, the discriminator D is trained on the entire batch minimizing $\mathcal{L}_{\text{adv}}$ from subsection 6.3.4.

6.4 Experiments

6.4.1 Prototype visualization

The first ablation considers an empirical analysis of the learned embedding space with the virtual classes. The objective is to verify that our architecture encourages the virtual classes to settle between

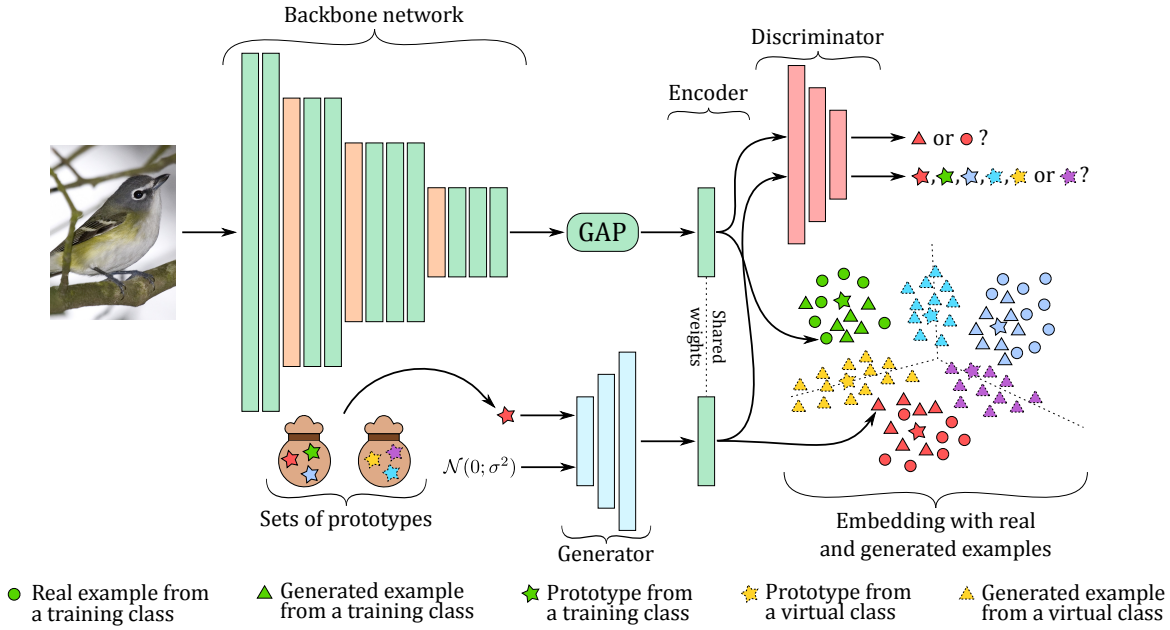


Figure 6.2: Overview of our proposed architecture. We extract a feature vector from the image using the backbone network and a global average pooling (GAP). Then, this feature vector is projected into the embedding space where we learn the metric. The framework relies on a set of prototypes (stars) which belong either to the training classes (plain lines) or to the virtual classes (dashed lines). We send through a generator a sampled prototype and a Gaussian noise to generate a new feature vector that is then projected into the same embedding space as the training images. In order to train this generator, we use a conditional discriminator to determine whether the sample is real or generated but also to determine the class to which it belongs.

	Training class examples ratio						Virtual class ratio					
ratio	0%	10%	50%	100%	200%	400%	0%	10%	50%	100%	200%	400%
Recall@1	57.0	57.8	**58.8**	58.5	58.7	58.7	57.0	58.2	59.0	**59.3**	58.6	58.9

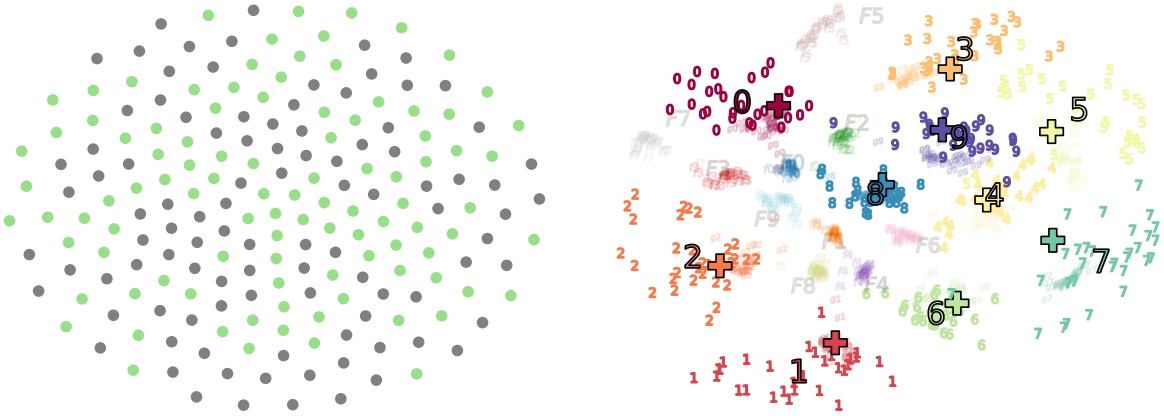
Table 6.1: Impact of the number of training class generated examples and of the number of virtual class in the mini-batches. A ratio of 0% means that no examples are generated.

the training classes. To that end, we show a t-SNE visualization of the training and virtual prototypes of the model trained on Cub-200-2011 on Figure 6.3a. As we can see, the virtual prototypes (in gray) are indeed in the middle of the training class prototypes. Quantitatively, we found that 80% of the training class prototypes have a virtual class prototype as nearest neighbor in the 512 dimensional latent space. This numerically shows that our architecture is able to produce virtual class as buffer areas between training classes.

To avoid the bias introduced by the 2D embedding performed by t-SNE, we also train a model on the popular MNIST dataset with a latent space of dimension 2. We plot the resulting prototypes as well as examples generated from these prototypes on Figure 6.3b. As we can see, the virtual prototypes (denoted F and in pale colors) are indeed acting as buffer between the training classes, even with the high constraints of having such a low dimensional latent space.

6.4.2 Example generation ablation

First, we evaluate the impact of the number of generated training class examples. For that purpose, we do not use virtual class prototypes. We vary the size of the generated example set $\widetilde{\mathcal{B}}$ with respect to a ratio r of the real example set $\mathcal{B}$, such that: $|\widetilde{\mathcal{B}}| = r\,|\mathcal{B}|$. We report Recall@1 on the Cub-200-2011 dataset in Table 6.1 for $r \in \{0, 10\%, 50\%, 100\%, 200\%, 400\%\}$.



(a) Cub-200-2011 (b) MNIST

Figure 6.3: Visualization of the training and virtual prototypes on MNIST and Cub-200-2011 datasets. For the Cub-200-2011 dataset (left), a t-SNE on the prototypes is run to visualize the high-dimensional embedding space. Virtual classes are in gray while training classes are in green. For the MNIST dataset (right), a 2D embedding space is learned without the ℓ_2-norm. Training examples are represented by the numbers, and the prototypes are represented by the crosses. Virtual examples are in pale color.

The reported value for $r = 0\%$ means that no examples have been generated and obtains a strong *Baseline* of 57.0% Recall@1. One can note that even a small amount of generated example, *e.g.*, 10%, increases performances by 0.8% in Recall@1 on the Cub-200-2011 dataset. Hence, this confirms the benefit of a generation-based mining strategy to improve DML. With a further increase of the size of the generated example set, we improve performances of the *Baseline* from 57.0% to 58.8% Recall@1, a significant increase of nearly 2%.

Next, we evaluate the impact of the number of virtual classes. We fix the size of the generated examples set $\widetilde{\mathcal{B}}$ to the size of the training examples batch $\mathcal{B}$, that is: $|\widetilde{\mathcal{B}}| = |\mathcal{B}| = 40$. Then, we vary the number of the virtual class prototype N_v as a ratio of the number of the training class N_t, such that $N_v = r\ N_t$. We only generate examples from these virtual classes and not from the training classes. We report Recall@1 on the Cub-200-2011 dataset in Table 6.1 for $r \in \{0, 10\%, 50\%, 100\%, 200\%, 400\%\}$. Interestingly, even a small number of additional classes, *e.g.* 10%, already improves the *Baseline* by a significant increase of more than 1.0% in Recall@1, from 57.0% to 58.2%. Increasing the number of virtual classes improves even more performances, and leads to the best results for Recall@1 with 59.3% - a significant increase of more than 2% over the *Baseline*.

Finally, we evaluate the merging of both the training class example generation and the virtual class example generation. Results are reported in Table 6.2. Following the two previous ablations, we set $|\widetilde{\mathcal{B}}| = |\mathcal{B}| = 40$ and $N_v = N_t$. To avoid any bias in the selection of these parameters, we report results on the Cars-196 dataset for three different DML losses; namely the contrastive loss [33], the triplet loss [152] and the binomial loss [173]. We also compare three different approaches, namely: the *Baseline*, the training class example generation only (denoted as TCG) and *MIRAGE*. For the three DML losses, both the training class example generation and *MIRAGE* lead to significant improvements over the *Baseline*. *E.g.*, with the contrastive loss, the *Baseline* is improved from 74.0% to 78.8% which is nearly a 5% improvement in Recall@1. Besides, performances of the binomial loss and the triplet loss are improved from 71.2% to 77.8% and from 70.9% to 73.6% respectively, which is an absolute improvement of +6.6% and +1.7% in Recall@1. This improvements is achieved without tuning the parameters for the Cars-196 dataset and for all evaluated DML loss functions.

6.4.3 Comparison to state-of-the-art

In this section, we present the benefits of *MIRAGE* on four deep metric learning datasets named Cub-200-2011 [174], Cars-196 [102], Stanford Online Products [127] and In-Shop Clothes Retrieval [115].

Method	R@1	R@2	R@4	R@8
Contrastive	74.0	83.1	89.4	93.8
Contrastive + TCG	76.3	85.2	90.8	94.6
Contrastive + *MIRAGE*	**78.8**	**86.4**	**91.7**	**95.4**
Triplet	70.9	80.5	87.6	92.8
Triplet + TCG	72.0	81.4	88.1	**93.2**
Triplet + *MIRAGE*	**73.6**	**82.2**	**88.5**	**93.2**
Binomial	71.2	80.8	87.7	93.0
Binomial + TCG	74.6	83.8	89.7	93.9
Binomial + *MIRAGE*	**77.8**	**86.1**	**91.3**	**94.7**

Table 6.2: Recall at K for three loss functions on the Cars-196 dataset with GoogleNet backbone network. We compare the baseline, the training class example generation only (denoted TCG) and *MIRAGE*. Results that improve over the baseline are underlined and best results are in bold for each loss function.

Backbone	Method	R@1	R@2	R@4	R@8
GoogleNet	DAMLRMM [193]	55.1	66.5	76.8	85.3
	DAML [50]	52.7	65.4	75.5	84.3
	DVML [111]	52.7	65.1	75.5	84.3
	HDML [202]	53.7	65.7	76.7	85.7
	***MIRAGE* (Ours)**	**59.7**	**71.1**	**80.4**	**88.1**
BN-Inception	MS loss [183]	65.7	77.0	**86.3**	**91.2**
	SoftTriplet [141]	65.4	76.4	84.5	90.4
	HORDE [85]	**66.8**	77.4	85.1	91.0
	***MIRAGE* (Ours)**	66.4	**78.9**	84.6	90.3

Table 6.3: Comparison to the state-of-the-art on the Cub-200-2011 dataset. Results are reported using GoogleNet as backbone network for fair comparison with generation-based methods. Results are also reported with BN-Inception backbone for comparison with other recent methods.

We follow the standard splits from [130] and Recall@K are reported for each dataset respectively in Table 6.3, Table 6.4, Table 6.5 and Table 6.6.

We first compare our architecture with recent sample generation approaches from the literature using the now standard GoogleNet backbone to ensure all results are fairly comparable. Also, the batch of real examples is now composed of 80 images and we keep the ratio from the ablation studies As we can see, *MIRAGE* obtains very strong results on all datasets. We achieve best performances on Cub-200-2011 and Cars-196, and second best on Stanford Online Products. This shows the importance of combining in class sample generation, like in [111] with hard sample generation like [202], which *MIRAGE* achieves with a simple architecture.

In order to compare *MIRAGE* with recent methods, we also report Recall@K using BN-Inception [79] with the same hyper-parameters as the ones used for GoogleNet. *MIRAGE* obtains strong performances when compared to very recent state-of-the-art methods. On Cub-200-2011, we obtain second best performances, being only 0.4% behind HORDE [85]. On Cars-196 and Stanford Online Products, we obtain performances comparable to that of Multi-Similarity loss [183] and SoftTriplet [141]. On In-Shop Clothes Retrieval, we obtain results comparable to MS loss [183] and better than recently proposed D&C [149] and MIC [144] that use the stronger ResNet50 backbone network. We want to emphasize that the reported results were obtained using the contrastive loss function, and yet bring improvements to the baseline comparable to that of using a much more advance loss function such as [183], [141] or [85]. We believe this demonstrates the soundness of our approach.

Backbone	Method	R@1	R@2	R@4	R@8
GoogleNet	DAMLRMM [193]	73.5	82.6	89.1	93.5
	DAML [50]	75.1	83.8	89.7	93.5
	DVML [111]	82.0	88.4	93.3	**96.3**
	HDML [202]	79.1	87.1	92.1	95.5
	***MIRAGE* (Ours)**	**82.1**	**89.1**	**93.6**	96.2
BN-Inception	MS loss [183]	84.1	90.4	94.0	96.5
	SoftTriplet [141]	84.5	90.7	94.5	96.9
	HORDE [85]	**86.2**	**91.9**	**95.1**	**97.2**
	***MIRAGE* (Ours)**	83.9	90.3	94.4	96.9

Table 6.4: Comparison to the state-of-the-art on the Cars-196 dataset. Results are reported using GoogleNet as backbone network for fair comparison with generation-based methods. Results are also reported with BN-Inception backbone for comparison with other recent methods.

Backbone	Method	R@1	R@10	R@100
GoogleNet	DAMLRMM [193]	69.7	**85.2**	93.2
	DAML [50]	68.4	83.5	92.3
	DVML [111]	**70.2**	**85.2**	**93.8**
	HDML [202]	68.7	83.2	92.4
	***MIRAGE* (Ours)**	69.9	84.3	92.6
ResNet50	D&C [149]	75.9	88.4	94.9
	MIC [144]	77.2	89.4	95.6
BN-Inception	MS loss [183]	78.2	90.5	96.0
	SoftTriplet [141]	78.3	90.3	95.9
	HORDE [85]	**80.1**	**91.3**	96.2
	***MIRAGE* (Ours)**	76.0	88.3	94.8

Table 6.5: Comparison to the state-of-the-art on the Stanford Online Products dataset. Results are reported using GoogleNet as backbone network for fair comparison with generation-based methods. Results are also reported with BN-Inception backbone for comparison with other recent methods.

Backbone	Method	R@1	R@10	R@20	R@30
ResNet50	D&C [149]	85.7	95.5	96.9	97.5
	MIC [144]	88.2	97.0	-	98.0
BN-Inception	MS loss [183]	89.7	**97.9**	**98.5**	**98.8**
	HORDE [85]	**90.4**	97.8	98.4	98.7
	***MIRAGE* (Ours)**	89.1	97.0	97.9	98.3

Table 6.6: Comparison to the state-of-the-art on the In-Shop Clothes Retrieval dataset.

6.5 Discussion

In this section, we discuss about the proposed example generation method according to the following points:

- Difference with HTG [201], DAML [50], DVML [111] and HDML [202]
- Difference with Magnet loss [143] and Proxy loss [122]

MIRAGE differs from other hard negative example generation methods, such as DAML [50], HTG [201], HDML [202] and DVML [111]. Both DAML and HTG rely on sampling a triplet, by feeding this triplet to a generator trained in an adversarial manner, and then producing a hard negative example to replace the original negative one. However, both methods suffer from the problem of label ambiguity illustrated in Figure 6.1 in that the generator can output an example inside another class manifold. Zheng *et al.* [202] face the same issue with HDML. HDML tries to alleviate this effect by generating first an intermediate example that may be outside its class manifold; then a generator projects this example into the class manifold. In case of a failure, they use the DML loss over the real examples to weight the generator loss: if the triplet is an easy one, the generator only slightly modifies the example to avoid the generation of an intermediate example that would be too far from its class manifold. Moreover, the metric learning loss that is computed on the generated triplets is also weighted by the reconstruction loss: the worse the reconstruction is, the less they take into account the new triplet. In other words, to mitigate the effect of label ambiguity, HDML tends to discard really hard negative examples which limits its hard negative generation capabilities. At the same time, DVML gets rid of the triplet constraints for the generator by considering the class manifold as a Gaussian distribution. By estimating the parameters of the distribution, the sampling of new examples is performed using a variational approach. Because there is no adversarial training, examples tend to be mostly sampled at the center of the Gaussian distribution. As such, they only slightly contribute to the DML loss.

To solve the problem of label ambiguity while generating hard negative, we propose to insert buffer areas between the training classes. To that end, we introduce virtual classes that we encourage to migrate between the training classes. Sampling hard negatives from the virtual classes allows us to use a generative sampling process similar to DVML [111] which is simpler than the triplet based adversarial methods. At the same time, it also removes the need to take into account the possible incorrectness of the labels since these generated examples do not correspond to existing classes.

MIRAGE also differs from proxy-based approaches such as Magnet loss [143] and Proxy loss [122]. Both methods consider proxies, which are similar to our prototypes, to model training classes. In Proxy loss [122], proxies does not belong to a given class. In fact, they can be seen as a quantification of the embedding space in order to ease the training. Examples are assigned to their nearest proxy and are trained to be moved closer to this proxy and to be far away from the others. Even if two examples from two different classes can be moved close to the same proxy, in practice, it leads to good results and speed-up the convergence compared to a naive batch construction in metric learning. In Magnet loss [143], many proxies are also trained to quantify the embedding space. However, the number of proxy is fixed to K proxies per training class in order to condition the quantification to the training classes. In a similar way to Proxy loss, the examples are trained to me moved to its nearest proxy with respect to its training class and to be moved far away from proxies of other training classes.

In *MIRAGE*, we consider prototypes (or proxies) as a way to generate examples from a given training class. The approach is similar to Magnet loss by considering only one prototype per class but we follow the Proxy loss approach by training the prototypes online instead of offline as it is done in Magnet loss using K-means algorithm. In addition, these prototypes are used to generate examples from training classes and are trained in an adversarial manner in order to generate hard negative examples. Furthermore, we consider virtual prototypes during the training which is comparable to Proxy loss when more proxies are used than training classes. However, we still consider these additional prototypes to belong to virtual classes, which means that every examples from training classes are expected to be moved far away from their contrary to Proxy loss. Furthermore, we generate examples from these virtual prototypes and train them in an adversarial manner in order to

generate hard examples but also to move the virtual prototypes between training classes. By doing so, the quantification is performed only within the data manifold, *i.e.* the manifold of all training examples. In Proxy loss, some proxies can be moved far away from the data manifold which make them useless during the training.

6.6 Conclusion

In this work, we introduce *MIRAGE*, a generation-based strategy that naturally solves the generation of hard examples. *MIRAGE* naturally solves the problem of generating incorrectly labeled hard negative examples by relying on a set of virtual class prototypes solely composed of generated examples. Even when the generator produces examples beyond their class manifolds, the presence of virtual classes ensures that the examples are still generated with the correct labels regarding the training classes. We empirically show that *MIRAGE* outperforms the state-of-the-art mining strategies and leads to competitive results when compared to complementary approaches. This is validated on four deep metric learning datasets named Cub-200-2011, Cars-196, Stanford Online Products and In-Shop Clothes Retrieval.

Part III

Conclusion and Perspectives

7

Global Conclusion

This thesis has been mainly focused on building richer and compact image representations using tensor-based aggregation methods, and on improving the training procedure of deep metric learning networks. In particular, three aggregation methods and two training strategies have been proposed. We have addressed the lack of expressivity of image representations by considering high-order statistics of local features and dictionary strategies. The dimensionality of these representations has been then tackled by proposing factorizations of projection matrices in order to reduce the number of parameters and the computation cost. First, we have extended the off-the-shelf method Spatial Tensor Aggregation (STA) with centering, normalization and two steps dimensionality reduction which leads to a compact representation with higher performances than end-to-end trainable representations. We have modified this representation in order to make it end-to-end trainable. This representation *JCF* has been shown to outperform most of state-of-the-art methods in deep metric learning, while being as compact as the original raw features, but with higher performances than representations that only consider first-order statistics. Finally, we have modified the ABE and NetVLAD representations by considering the links between the dictionary-based and attention-based approaches. As such, the divergence loss in ABE has been removed for the benefit of a dictionary that structurally encourages the attention maps to be complementary.

The training procedure in metric learning has been improved through the two methods *HORDE* and *MIRAGE*. The high-order regularization in *HORDE* has been shown to greatly improve the robustness of image representations by encouraging local features from the same class to have the same distribution, and local features from different classes to have different distributions. We have given theoretical guarantees of our methods through upper and lower bounds of well-known divergence between distributions. We also have given an efficient implementation which leads to few computation overhead during the training. The example generation method *MIRAGE* has been designed to solve an inherent problem of example-based named *label ambiguity*. This label ambiguity comes from the hard examples generation process that might generate examples outside their class manifold, or even worse, within another class manifold with an incorrect label. Such examples might destroy the embedding space when we use them to train the metric and the image representations. *MIRAGE* has addressed this issue by considering virtual classes that are trained to be placed between the training classes. By doing so, examples are generated between a training class and a virtual class (or within a virtual class), and the virtual class plays the role of buffer area to absorb local changes into the embedding that might destroy it.

8

Perspectives

In this chapter, we extend the discussion about our current work and we give perspectives about future research.

Contents

8.1 Image Representation 104
8.1.1 Extension of Joint Codebook Factorization 104
8.1.2 Factorization of Covariance Matrices 104
8.1.3 High-Order Moments for Image Representation 105
8.1.4 Invariance and Prior in Image Representation 105
8.2 Extension of *HORDE* 105
8.3 Rethinking Metric Learning 106

8.1 Image Representation

8.1.1 Extension of Joint Codebook Factorization

Following the discussion in subsection 4.3.9, *JCF* could be modified by using a fixed set of projections that are shared with every dictionary entries instead of a single set for each dictionary. As such, Equation 4.22 can be modified by considering $\tilde{\boldsymbol{U}}$ instead of $\tilde{\boldsymbol{U}}_i$ and $\tilde{\boldsymbol{V}}_i$, and by learning many $\boldsymbol{A}_i$ and $\boldsymbol{B}_i$ instead of single $\boldsymbol{A}$ and $\boldsymbol{B}$:

$$z_i(\boldsymbol{x}) = \left(\boldsymbol{h}(\boldsymbol{x})^\top \boldsymbol{A}_i^\top \boldsymbol{x}\right) \left(\boldsymbol{h}(\boldsymbol{x})^\top \boldsymbol{B}_i^\top \boldsymbol{x}\right) \tag{8.1}$$

Thus, we approximate our projections using a shared pool of projections with a growth in $\mathcal{O}(Rd)$, and we learn to combine them for each dictionary entry and for each output dimension, which means that the combination of them require $\mathcal{O}(RND)$ parameters. Compared to *JCF*, we enhance the parameter gain from $\frac{N}{R}$ to $\frac{d}{R}$.

From another point of view, the low-rank approximation in *JCF* makes the assumption that the projection matrices are low-rank, and reduces the complexity from $\mathcal{O}(NdD)$ to $\mathcal{O}(RdD)$. The sharing approach supposes that the features are low-rank, and reduces the complexity from $\mathcal{O}(NdD)$ to $\mathcal{O}(NRD)$. In a similar way, one can consider that the output is low-rank, and reduces the complexity from $\mathcal{O}(NdD)$ to $\mathcal{O}(NdR)$ by considering a bottleneck at the output. Contrary to the previous factorizations, this one is not really interesting as it is, because it adds parameters and increases the dimensionality of the output representation without potential gain in performances compared to simply using an output representation of size R. Finally, these three factorizations only act on a single property: either the dictionary in *JCF*, the projections in the sharing method or on the output dimension with the bottleneck. One enhancement could be to exploit multiple properties factorization using *e.g.*, Tucker decomposition or CP decomposition.

8.1.2 Factorization of Covariance Matrices

In the case of covariance matrices, designing better factorization methods would make these representations more attractive for a variety of tasks such as VQA or image retrieval. A research direction could be to consider dimensionality reduction methods that respect the geometry of the manifold. For instance, it could be interesting to investigate how the eigenvectors of covariance matrices are modified between two close covariance matrices in terms of geodesic distance: Are only the eigenvalues close to each other and the eigenvectors "random" ? Are only the eigenvectors close to each other ? Are both the eigenvectors and eigenvalues close to each other ? With this information, non-linear dimensionality reduction could be designed to keep the advantage of using a geodesic distance. Also, with a well-designed factorization, one might simplify the computation of the geodesic distance to make it tractable for image retrieval applications.

Another direction could be to rethink this covariance pooling in the case of deep learning. Indeed, for most of deep networks, these matrices are necessary low-rank because there are not enough local features in the feature maps. This leads to covariance matrices that are semi-definite, and all hypotheses about the SPD manifold do not hold in practice. Thus, one should exploit the properties of the symmetric positive semi-definite (SPSD) matrix manifold in order to build geodesic distances, or dimensionality reduction methods. A first approach could be to "augment" the local feature map. For example, one could model their distribution and sample additional local features with respect to this distribution is order to compute their full-rank covariance matrix. A second approach could be to consider the geometry of the SPSD manifold. For example, in the case of a fixed low-rank k, the SPSD manifold is Riemannian, geodesically complete (all "straight lines" stay on the manifold, can be indefinitely followed and in all directions), and a geodesic distance invariant to all transformations that preserve angles (orthogonal transformations, scalings and pseudoinversion) can be defined [16]. Consequently, such properties could be exploited in order to design dimensionality reduction methods that preserves the geometry of the SPSD manifold.

8.1.3 High-Order Moments for Image Representation

Higher-order moments than the second-order could benefit for other computer vision tasks that are inherently multimodal. For instance, one can extend the visual question answering from images to videos. Using the modalities of images and sound from the videos and the text modality for the question naturally leads to a third-order tensor to encode the input information. Still, high-order moments in computer vision are not well understood, especially when it comes to the geometry of their manifold. Also, most of the properties of second-order matrices do not extend to high-order moments. Even worse, standard approaches well-known on covariance matrices are NP-Hard in the case of high-order moments [77], or ill-posed (*e.g.*, the low-rank approximation of tensors is an ill-posed problem). Consequently, studying these tensors could naturally address many problems in computer vision.

8.1.4 Invariance and Prior in Image Representation

In this section, we discuss the construction of richer representations in light of the statistical learning theory. While the question of why should we build richer representations is easily understood and has direct consequences in computer vision tasks (better results), the question of how to build them is still an open topic. Also, one way to argue against research on the construction of richer representations is directly related to the statistical learning theory: with enough samples, any deep learning model should be able to solve any task simply by minimizing the empirical risk, even using the mean of the local features as representations. In most of the current research in computer vision, and especially in metric learning, this empirical risk minimization is the mainly addressed research direction: training procedure is improved with loss functions with better generalization properties, example mining strategies to accelerate the learning, regularization methods, *etc.* These methods tend to give good results but require lots of samples to be trained in order to avoid over-fitting. As such, these approaches are not suited for learning with few examples, even if metric learning is the de facto approach for this task.

Structural risk minimization is far less studied than empirical risk minimization but should provide answers to these tasks. In structural risk minimization, the main idea is to select the class of functions using prior knowledge of the domain, and to evaluate many of them by increasing their complexity. For example, we might select the class of residual network architectures [74] and evaluate many of this architecture by increasing the number of neurons per layers, the number of layers, and so on. Consequently, one research direction is to find better image representations that encode within the architecture some priors or invariance. By doing so, we minimize the structural risk because we limit the expressivity of the original architecture (certain functions cannot be approximated any better), but this could help to minimize the empirical risk when there are only few available samples. For instances, replacing the fully connected layers by convolutional layers in the case of image analysis limits the expressivity of the network. However, this prior is interesting because it exploits domain knowledge: images are 2D signals, and filtering in signal processing (*i.e.*, convolution) is a good way to deal with it. Also, it introduces invariance to translation, which is a very interesting prior in images: translating objects in an images should not change the output prediction. Similarly, building image representations that lie on a manifold (*e.g.*, a Grassmannian manifold) limits the expressivity of the network but encode priors and invariance (invariance to rotation in the case of a Grassmannian manifold and priors about the detection of patterns - the eigenvectors, but not to their numbers - the corresponding eigenvalues).

8.2 Extension of *HORDE*

As discussed in section 5.5, *HORDE* could be enhanced by considering better high-order moment approximations, such as the one proposed in *JCF*. Extending the *JCF* method to approximate the high-order moments could improve the regularization in *HORDE* by betting approximating them. From another point of view, one could design other methods in order to optimize the distribution of deep features. For instance, one could argue that the local features are inherently multimodal:

for birds, one feature represents the beak, another the feathers, *etc.* in the case of birds recognition. Thus, modelizing the distribution of the local features using, *e.g.*, a Gaussian mixture model (GMM) and optimize a divergence between these distributions for each mode of the GMM could lead to better performances.

8.3 Rethinking Metric Learning

As raised by recent pre-print articles [145, 123], current supervised metric learning evaluation protocol is flawed: too small datasets subjected to over-fitting, no validation set, marginal improvements due to recent contributions and high change in performances from hyper-parameters tuning (batch size, image size, optimizers, frozen batch-normalization or not, *etc.*) and so on. Thus, the question of rethinking the evaluation protocol has been discussed in these two articles and most of the above remarks have been addressed. From another point of view, the current evaluation protocol is directly related to the classification task with a k-NN classifier. In this way, recent work have shown that classification with softmax and "proxies per class" (*i.e.*, linear classifiers) works better than most of tuple-based loss functions [199]. These considerations are mainly a consequence of the datasets themselves. Actually, they are fine-grained classification datasets that have been split into different training and testing sets. This comes against the original motivation of image retrieval which was to give a measure of similarity between pairs of images in a more general setting than just the notion of same class or not.

Finally, if we look at other tasks such as zero-shot classification that are built upon metric learning-based methods, the "generalized zero-shot" protocol shows that generalization to unseen does not appear [17] and is only a consequence of an "egg box phenomenon" in the embedding space. In generalized zero-shot learning, the training and testing sets are usually built as follows: the training set is composed of a set of images from a fixed number of classes usually referred to "seen classes". The testing set is composed of two subsets: the first one is composed of images from the "seen classes" and the second one is composed of images from "unseen classes", that is, classes that have not been seen during the training. The evaluation protocol is divided into four evaluations:

- $s \rightarrow s$: Images from the **seen** classes are used as queries to perform the retrieval of images from the **seen** classes
- $u \rightarrow u$: Images from the **unseen** classes are used as queries to perform the retrieval of images from the **unseen** classes
- $s \rightarrow a$: Images from the **seen** classes are used as queries to perform the retrieval of images from all classes (**seen and unseen**)
- $u \rightarrow a$: Images from the **unseen** classes are used as queries to perform the retrieval of images from all classes (**seen and unseen**)

The $u \rightarrow u$ is the standard protocol in metric learning, but some datasets also use the $s \rightarrow s$ (*e.g.*, in person re-id). In these two setups, performances are usually good, thanks to this "egg box phenomenon": there is the same number of seen classes and unseen classes, so each unseen class falls into an "egg hole" of a seen class. However, the two other protocols are much more different: the testing set is composed of both seen and unseen classes, which means that there are more classes during testing than training. As such, there are not enough "egg holes" for each class, and we can observe two phenomenons: On the $s \rightarrow a$ protocol, performances have only slightly decreased because the "egg holes" have been learned for these seen classes. On the $u \rightarrow a$: performances are extremely degraded: because the unseen classes fall into "egg holes" of seen classes, and because these "egg holes" have been specifically learned for these seen classes, performances are very bad (typically, performances are halved). As a consequence, the question of generalization to unseen classes in image retrieval is questionable, and might be easily shown using, for example, the training images as distractors during the retrieval. Investigating the causes of such phenomenon would benefit the computer vision community for a wide range of tasks, and might help to understand generalization to "out-of-distribution" classes.

One way to cope with this phenomenon could be to consider multi-label datasets to get the measure of similarity, or even better with only annotations of pairs of images with (or without) partial labelling similarly to multi-label classification with partial labels [51]. By doing so, learning a single embedding to encode every relationship between labels should be challenging but with a great interests for image search engine applications (similarity-based product recommendation or comparison, hashtag assignment from visual similarity, *etc*).

Thanks to the multi-label annotation, the similarity can be refined by counting the number of matching labels between pairs. For example, a given image A annotated similar to two other images B and C, can be very similar to B with 6 matched labels while being slightly similar to C with only 1 shared label. At the same time, an image D can be similar to A with 3 matched labels but might not be annotated. To evaluate such models, a fully annotated test set can be used to measure different properties of the embedding space such as: ranking similarity (ranking of images that have matching labels), label encoding (ranking of images that match or not with respect to a single label, averaged by all images per label), *etc.* As a consequence, these modifications in the evaluation protocol would make the supervised metric learning setup more challenging while still having the possibility to add unseen labels during testing.

Bibliography

[1] Relja Arandjelovic, Petr Gronat, Akihiko Torii, Tomas Pajdla, and Josef Sivic. Netvlad: Cnn architecture for weakly supervised place recognition. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1437 – 1451, June 2016. 4, 9, 50, 51, 55, 56, 58

[2] Relja Arandjelovic and Andrew Zisserman. All about vlad. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1578 – 1585, June 2013. 33, 52, 70

[3] Vincent Arsigny, Pierre Fillard, Xavier Pennec, and Nicholas Ayache. Log-euclidean metrics for fast and simple calculus on diffusion tensors. *Magnetic Resonance in Medicine: An Official Journal of the International Society for Magnetic Resonance in Medicine*, 56(2):411–421, 2006. 34

[4] Vincent Arsigny, Pierre Fillard, Xavier Pennec, and Nicholas Ayache. Geometric means in a novel vector space structure on symmetric positive-definite matrices. *SIAM journal on matrix analysis and applications*, 29(1):328–347, 2007. 30

[5] Adrien Auclair, Nicole Vincent, and Laurent D. Cohen. Hash functions for near duplicate image retrieval. In *The IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 1–6, Dec 2009. 4, 10

[6] Nicolas Aziere and Sinisa Todorovic. Ensemble deep manifold similarity learning using hard proxies. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019. 22

[7] Ricardo Baeza-Yates and Berthier Ribeiro-Neto. *Modern Information Retrieval*. Addison Wesley, 1999. 4

[8] Ankan Bansal, Karan Sikka, Gaurav Sharma, Rama Chellappa, and Ajay Divakaran. Zero-shot object detection. In *The European Conference on Computer Vision (ECCV)*, September 2018. 14

[9] R. H. Bartels and G. W. Stewart. Solution of the matrix equation ax + xb = c [f4]. *Commun. ACM*, 15(9):820–826, Sept. 1972. 37

[10] Herbert Bay, Andreas Ess, Tinne Tuytelaars, and Luc Van Gool. Speeded-up robust features (surf). *Computer vision and image understanding*, 110(3):346–359, 2008. 5

[11] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. Surf: Speeded up robust features. In *The European Conference on Computer Vision (ECCV)*, pages 404–417. Springer, 2006. 5

[12] Jeffrey S Beis and David G Lowe. Shape indexing using approximate nearest-neighbour search in high-dimensional spaces. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1000–1006, 1997. 6

[13] Hedi Ben-Younes, Rémi Cadene, Matthieu Cord, and Nicolas Thome. Mutan: Multimodal tucker fusion for visual question answering. In *The IEEE International Conference on Computer Vision (ICCV)*, pages 2612–2620, 2017. 30, 32, 42

[14] Hedi Ben-Younes, Remi Cadene, Nicolas Thome, and Matthieu Cord. Block: Bilinear superdiagonal fusion for visual question answering and visual relationship detection. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 8102–8109, 2019. 30, 42

[15] Alina Beygelzimer, Elad Hazan, Satyen Kale, and Haipeng Luo. Online gradient boosting. In *Advances in Neural Information Processing Systems (NIPS)*, pages 2458–2466, 2015. 22

[16] Silvere Bonnabel and Rodolphe Sepulchre. Riemannian metric and geometric mean for positive semidefinite matrices of fixed rank. *SIAM Journal on Matrix Analysis and Applications*, 31(3):1055–1070, 2010. 104

[17] Maxime Bucher, Stéphane Herbin, and Frédéric Jurie. Generating visual representations for zero-shot classification. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, pages 2666–2673, 2017. 106

[18] Maxime Bucher, Tuan-Hung VU, Matthieu Cord, and Patrick Pérez. Zero-shot semantic segmentation. In *Advances in Neural Information Processing Systems (NIPS)*, pages 466–477, 2019. 84

[19] Sijia Cai, Wangmeng Zuo, and Lei Zhang. Higher-order integration of hierarchical convolutional activations for fine-grained visual categorization. In *The IEEE International Conference on Computer Vision (ICCV)*, Oct 2017. 40, 44

[20] Fatih Cakir, Kun He, Xide Xia, Brian Kulis, and Stan Sclaroff. Deep metric learning to rank. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1861 – 1870, June 2019. 17, 19, 20

[21] Michael Calonder, Vincent Lepetit, Mustafa Ozuysal, Tomasz Trzcinski, Christoph Strecha, and Pascal Fua. Brief: Computing a local binary descriptor very fast. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 34(7):1281–1298, 2011. 5

[22] Michael Calonder, Vincent Lepetit, Christoph Strecha, and Pascal Fua. Brief: Binary robust independent elementary features. In *The European Conference on Computer Vision (ECCV)*, pages 778–792. Springer, 2010. 5

[23] Yue Cao, Mingsheng Long, Bin Liu, and Jianmin Wang. Deep cauchy hashing for hamming space retrieval. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1229–1237, 2018. 10

[24] Zhe Cao, Tao Qin, Tie-Yan Liu, Ming-Feng Tsai, and Hang Li. Learning to rank: from pairwise approach to listwise approach. In *International Conference on Machine Learning (ICML)*, pages 129–136, 2007. 19

[25] João Carreira, Rui Caseiro, Jorge Batista, and Cristian Sminchisescu. Semantic segmentation with second-order pooling. In *The European Conference on Computer Vision (ECCV)*, pages 430–443, 2012. 30

[26] Micael Carvalho, Rémi Cadène, David Picard, Laure Soulier, Nicolas Thome, and Matthieu Cord. Cross-modal retrieval in the cooking context: Learning semantic text-image embeddings. In *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*, pages 35 – 44, 2018. 88

[27] Ken Chatfield, Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Return of the devil in the details: Delving deep into convolutional nets. In *Proceedings of the British Machine Vision Conference (BMVC)*, 2014. 38

[28] Ratthachat Chatpatanasiri, Teesid Korsrilabutr, Pasakorn Tangchanachaianan, and Boonserm Kijsirikul. A new kernelization framework for mahalanobis distance learning algorithms. *Neurocomputing*, 73:1570–1579, 2010. 14

[29] Long Chen, Hanwang Zhang, Jun Xiao, Wei Liu, and Shih-Fu Chang. Zero-shot visual recognition using semantics-preserving adversarial embedding networks. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018. 14

[30] Weihua Chen, Xiaotang Chen, Jianguo Zhang, and Kaiqi Huang. Beyond triplet loss: A deep quadruplet network for person re-identification. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1320 – 1329, July 2017. 17, 18

[31] Anoop Cherian, Suvrit Sra, Arindam Banerjee, and Nikolaos Papanikolopoulos. Jensen-bregman logdet divergence with application to efficient similarity search for covariance matrices. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 35(9):2161–2174, Sep. 2013. 33, 34

[32] Kyunghyun Cho, Bart van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder–decoder approaches. In *Proceedings of SSST-8, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation, EMNLP*, pages 103–111, Oct. 2014. 32

[33] Sumit Chopra, Raia Hadsell, and Yann LeCun. Learning a similarity metric discriminatively, with application to face verification. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 539 – 546, 2005. 90, 93

[34] Aruni Roy Chowdhury, Tsung-Yu Lin, Subhransu Maji, and Erik Learned-Miller. One-to-many face recognition with bilinear cnns. In *2016 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 1–9, 2016. 30

[35] Matthieu Cord, Sylvie Philipp-Foliguet, Philippe-Henri Gosselin, and Jérôme Fournier. Interactive exploration for image retrieval. *EURASIP Journal on Advances in Signal Processing*, 2005(14):649689, 2005. 4

[36] Gabriella Csurka, Christopher Dance, Lixin Fan, Jutta Willamowski, and Cédric Bray. Visual categorization with bags of keypoints. In *The European Conference on Computer Vision (ECCV) Workshops*, pages 1–2, 2004. 4, 5, 6, 50

[37] Yin Cui, Feng Zhou, Jiang Wang, Xiao Liu, Yuanqing Lin, and Serge Belongie. Kernel pooling for convolutional neural networks. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017. 44

[38] Tao Dai, Jianrui Cai, Yongbing Zhang, Shu-Tao Xia, and Lei Zhang. Second-order attention network for single image super-resolution. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 11065–11074, 2019. 30

[39] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 1, pages 886–893, June 2005. 4, 5

[40] Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab S Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *Proceedings of the twentieth annual symposium on Computational geometry*, pages 253–262, 2004. 6

[41] Jason V. Davis, Brian Kulis, Prateek Jain, Suvrit Sra, and Inderjit S. Dhillon. Information-theoretic metric learning. In *International Conference on Machine Learning (ICML)*, page 209–216, 2007. 14

[42] Lieven De Lathauwer, Bart De Moor, and Joos Vandewalle. A multilinear singular value decomposition. *SIAM journal on Matrix Analysis and Applications*, 2000. 42

[43] Vin De Silva and Lek-Heng Lim. Tensor rank and the ill-posedness of the best low-rank approximation problem. *SIAM Journal on Matrix Analysis and Applications*, 30(3):1084–1127, 2008. 42

[44] Jonathan Delhumeau, Philippe-Henri Gosselin, Hervé Jégou, and Patrick Perez. Revisiting the vlad image representation. In *ACM Multimedia*, pages 653–656, Barcelona, Spain, Oct. 2013. 52, 59, 70

[45] J. Deng, W. Dong, R. Socher, L. Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 248–255, June 2009. 5, 30, 36, 55

[46] Daniel DeTone, Tomasz Malisiewicz, and Andrew Rabinovich. Superpoint: Self-supervised interest point detection and description. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, June 2018. 6

[47] Mandar D Dixit and Nuno Vasconcelos. Object based scene representations using fisher scores of local subspace projections. In *Advances in Neural Information Processing Systems (NIPS)*, pages 2811–2819, 2016. 50

[48] Thanh-Toan Do, Quang D. Tran, and Ngai-Man Cheung. Faemb: A function approximation-based embedding method for image retrieval. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015. 52

[49] Ian L. Dryden, Alexey Koloydenko, and Diwei Zhou. Non-euclidean statistics for covariance matrices, with applications to diffusion tensor imaging. *The Annals of Applied Statistics*, 3(3):1102–1123, 09 2009. 35

[50] Yueqi Duan, Wenzhao Zheng, Xudong Lin, Jiwen Lu, and Jie Zhou. Deep adversarial metric learning. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2780 – 2789, June 2018. 17, 20, 21, 67, 88, 89, 94, 95, 96

[51] Thibaut Durand, Nazanin Mehrasa, and Greg Mori. Learning a deep convnet for multi-label classification with partial labels. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019. 107

[52] Thibaut Durand, Nicolas Thome, and Matthieu Cord. Weldon: Weakly supervised learning of deep convolutional neural networks. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016. 9

[53] Martin A. Fischler and Robert C. Bolles. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, June 1981. 6

[54] Yoav Freund and Robert E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. In *Computational Learning Theory*, pages 23–37, 1995. 22

[55] Akira Fukui, Dong Huk Park, Daylen Yang, Anna Rohrbach, Trevor Darrell, and Marcus Rohrbach. Multimodal compact bilinear pooling for visual question answering and visual grounding. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 457–468. Association for Computational Linguistics, 2016. 40

[56] Yang Gao, Oscar Beijbom, Ning Zhang, and Trevor Darrell. Compact bilinear pooling. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 317 – 326, June 2016. 30, 33, 38, 40, 50

[57] Weifeng Ge, Weilin Huang, Dengke Dong, and Matthew R. Scott. Deep metric learning with hierarchical triplet loss. In *The European Conference on Computer Vision (ECCV)*, pages 272 – 288, September 2018. 17, 19, 61, 62, 67, 80, 81

[58] Hugo Germain, Guillaume Bourmaud, and Vincent Lepetit. Sparse-to-dense hypercolumn matching for long-term visual localization. In *2019 International Conference on 3D Vision (3DV)*, pages 513–523, Sep. 2019. 9

[59] Jacob Goldberger, Geoffrey E Hinton, Sam T. Roweis, and Russ R Salakhutdinov. Neighbourhood components analysis. In *Advances in Neural Information Processing Systems (NIPS)*, pages 513–520, 2005. 14

[60] Yunchao Gong, Svetlana Lazebnik, Albert Gordo, and Florent Perronnin. Iterative quantization: A procrustean approach to learning binary codes for large-scale image retrieval. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 35(12):2916–2929, 2013. 51

[61] Albert Gordo, Jon Almazán, Jérome Revaud, and Diane Larlus. Deep image retrieval: Learning global representations for image search. In *The European Conference on Computer Vision (ECCV)*, 2016. 9

[62] Albert Gordo, Jon Almazán, Jerome Revaud, and Diane Larlus. End-to-end learning of deep visual representations for image retrieval. *International Journal of Computer Vision (IJCV)*, 124(2):237–254, Sept. 2017. 9

[63] Albert Gordo and Diane Larlus. Beyond instance-level image retrieval: Leveraging captions to learn a global visual representation for semantic retrieval. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017. 4

[64] Mengran Gou, Fei Xiong, Octavia Camps, and Mario Sznaier. Monet: Moments embedding network. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018. 40, 43

[65] Arthur Gretton, Karsten M Borgwardt, Malte Rasch, Bernhard Schölkopf, and Alex J Smola. A kernel method for the two-sample-problem. In *Advances in Neural Information Processing Systems (NIPS)*, pages 513–520, 2007. 78, 79, 84

[66] Raia Hadsell, Sumit Chopra, and Yann LeCun. Dimensionality reduction by learning an invariant mapping. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2006. 15

[67] Jihun Hamm and Daniel D Lee. Grassmann discriminant analysis: a unifying view on subspace-based learning. In *International Conference on Machine Learning (ICML)*, pages 376–383, 2008. 44

[68] Mehrtash Harandi, Mathieu Salzmann, and Richard Hartley. Dimensionality reduction on spd manifolds: The emergence of geometry-aware methods. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 40(1):48–62, 2017. 34, 41

[69] Mehrtash T. Harandi, Mathieu Salzmann, and Richard Hartley. From manifold to manifold: Geometry-aware dimensionality reduction for spd matrices. In *The European Conference on Computer Vision (ECCV)*, pages 17–32, 2014. 34, 41

[70] Bharath Hariharan and Ross Girshick. Low-shot visual recognition by shrinking and hallucinating features. In *The IEEE International Conference on Computer Vision (ICCV)*, Oct 2017. 14

[71] Chris Harris and Mike Stephens. A combined corner and edge detector. In *Alvey Vision Conference*, pages 147–151, 1988. 5

[72] B. Harwood and T. Drummond. Fanng: Fast approximate nearest neighbour graphs. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5713–5722, June 2016. 20

[73] Ben Harwood, Vijay Kumar B G, Gustavo Carneiro, Ian Reid, and Tom Drummond. Smart mining for deep metric learning. In *The IEEE International Conference on Computer Vision (ICCV)*, pages 2840 – 2848, Oct 2017. 17, 20

[74] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770 – 778, June 2016. 32, 33, 52, 105

[75] Nicholas J Higham. Stable iterations for the matrix square root. *Numerical Algorithms*, 15(2):227–242, 1997. 37

[76] Nicholas J Higham. *Functions of matrices: theory and computation*, volume 104. Siam, 2008. 37

[77] Christopher J. Hillar and Lek-Heng Lim. Most tensor problems are np-hard. *Journal of the ACM*, 60(6), Nov. 2013. 42, 44, 105

[78] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *CoRR*, abs/1704.04861, 2017. 55, 70

[79] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning (ICML)*, pages 448 – 456, Jul 2015. 81, 85, 90, 91, 94

[80] Catalin Ionescu, Orestis Vantzos, and Cristian Sminchisescu. Matrix backpropagation for deep networks with structured layers. In *The IEEE International Conference on Computer Vision (ICCV)*, December 2015. 34, 35, 37

[81] Catalin Ionescu, Orestis Vantzos, and Cristian Sminchisescu. Training deep networks with structured layers by matrix backpropagation. *arXiv preprint arXiv:1509.07838*, 2015. 35

[82] Ahmet Iscen, Giorgos Tolias, Yannis Avrithis, and Ondřej Chum. Mining on manifolds: Metric learning without labels. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018. 20, 22, 23

[83] Tommi Jaakkola and David Haussler. Exploiting generative models in discriminative classifiers. In *Advances in Neural Information Processing Systems (NIPS)*, pages 487–493, 1999. 7

[84] Pierre Jacob, David Picard, Aymeric Histace, and Edouard Klein. Efficient codebook and factorization for second order representation learning. In *The IEEE International Conference on Image Processing (ICIP)*, pages 849 – 853, Sep. 2019. 67

[85] Pierre Jacob, David Picard, Aymeric Histace, and Edouard Klein. Metric learning with horde: High-order regularizer for deep embeddings. In *The IEEE International Conference on Computer Vision (ICCV)*, pages 6539 – 6548, October 2019. 67, 94, 95

[86] Sadeep Jayasumana, Richard Hartley, Mathieu Salzmann, Hongdong Li, and Mehrtash Harandi. Kernel methods on riemannian manifolds with gaussian rbf kernels. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 37(12):2464–2477, 2015. 44

[87] Hervé Jégou and Ondrej Chum. Negative evidences and co-occurrences in image retrieval: the benefit of PCA and whitening. In *The European Conference on Computer Vision (ECCV)*, Oct. 2012. 55, 76

[88] Herve Jegou, Matthijs Douze, and Cordelia Schmid. Hamming embedding and weak geometric consistency for large scale image search. In *The European Conference on Computer Vision (ECCV)*, page 304–317, 2008. 4, 7, 8, 10, 23, 51, 55

[89] Herve Jegou, Matthijs Douze, and Cordelia Schmid. Product quantization for nearest neighbor search. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 33(1):117–128, 2010. 6

[90] Ke Jiang, Qichao Que, and Brian Kulis. Revisiting kernelized locality-sensitive hashing for improved large-scale image retrieval. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015. 51

[91] H. Jégou, M. Douze, C. Schmid, and P. Pérez. Aggregating local descriptors into a compact image representation. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3304–3311, June 2010. 4, 5, 6, 45, 50, 51

[92] Yannis Kalantidis, Clayton Mellina, and Simon Osindero. Cross-dimensional weighting for aggregated deep convolutional features. In *The European Conference on Computer Vision (ECCV) Workshops*, pages 685–701, 2016. 55

[93] Purushottam Kar and Harish Karnick. Random feature maps for dot product kernels. In *Artificial Intelligence and Statistics*, pages 583–591, 2012. 38

[94] Purushottam Kar and Harish Karnick. Random feature maps for dot product kernels. In *Proceedings of the Fifteenth International Conference on Artificial Intelligence and Statistics*, Apr 2012. 76

[95] Leonid Karlinsky, Joseph Shtok, Sivan Harary, Eli Schwartz, Amit Aides, Rogerio Feris, Raja Giryes, and Alex M. Bronstein. Repmet: Representative-based metric learning for classification and few-shot object detection. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019. 14

[96] Dor Kedem, Stephen Tyree, Fei Sha, Gert R Lanckriet, and Kilian Q Weinberger. Non-linear metric learning. In *Advances in Neural Information Processing Systems (NIPS)*, pages 2573–2581, 2012. 14

[97] Jin-Hwa Kim, Kyoung Woon On, Woosang Lim, Jeonghee Kim, Jung-Woo Ha, and Byoung-Tak Zhang. Hadamard product for low-rank bilinear pooling. In *International Conference on Learning Representations (ICLR)*, 2017. 30, 41, 50

[98] Wonsik Kim, Bhavya Goyal, Kunal Chawla, Jungmin Lee, and Keunjoo Kwon. Attention-based ensemble for deep metric learning. In *The European Conference on Computer Vision (ECCV)*, pages 760 – 777, September 2018. 50, 51, 62, 66, 67, 68, 69, 74, 80, 81, 85

[99] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*, May 2015. 66, 74

[100] Shu Kong and Charless Fowlkes. Low-rank bilinear pooling for fine-grained classification. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 7025 – 7034, July 2017. 40, 41, 50

[101] Andreas Krause and Daniel Golovin. Submodular function maximization. In *Tractability*, pages 71–104. Cambridge University Press, 2014. 19

[102] Jonathan Krause, Michael Stark, Jia Deng, and Li Fei-Fei. 3d object representations for fine-grained categorization. In *4th International IEEE Workshop on 3D Representation and Recognition (3dRR-13)*, pages 554 – 561, Sydney, Australia, 2013. 24, 61, 69, 75, 80, 93

[103] Kyoung Mu Lee. Mode-seeking on graphs via random walks. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, page 606–613, 2012. 20

[104] Peihua Li, Jiangtao Xie, Qilong Wang, and Zilin Gao. Towards faster training of global covariance pooling networks by iterative matrix square root normalization. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 947–955, 2018. 30, 37

[105] Peihua Li, Jiangtao Xie, Qilong Wang, and Wangmeng Zuo. Is second-order information helpful for large-scale visual recognition? In *The IEEE International Conference on Computer Vision (ICCV)*, Oct 2017. 34, 35, 36

[106] Xinchao Li, Martha Larson, and Alan Hanjalic. Pairwise geometric matching for large-scale object retrieval. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015. 51

[107] Yunsheng Li, Mandar Dixit, and Nuno Vasconcelos. Deep scene image classification with the mfafvnet. In *The IEEE International Conference on Computer Vision (ICCV)*, pages 5757 – 5765, Oct 2017. 50, 51

[108] Yanghao Li, Naiyan Wang, Jiaying Liu, and Xiaodi Hou. Factorized bilinear models for image recognition. In *The IEEE International Conference on Computer Vision (ICCV)*, Oct 2017. 40

[109] Tsung-Yu Lin and Subhransu Maji. Improved bilinear pooling with cnns. In *Proceedings of the British Machine Vision Conference (BMVC)*, Sep. 2017. 36, 37

[110] Tsung-Yu Lin, Aruni RoyChowdhury, and Subhransu Maji. Bilinear cnn models for fine-grained visual recognition. In *The IEEE International Conference on Computer Vision (ICCV)*, December 2015. 9, 30, 33, 50

[111] Xudong Lin, Yueqi Duan, Qiyuan Dong, Jiwen Lu, and Jie Zhou. Deep variational metric learning. In *The European Conference on Computer Vision (ECCV)*, pages 714 – 729, September 2018. 17, 20, 21, 67, 74, 81, 84, 88, 94, 95, 96

[112] Hongye Liu, Yonghong Tian, Yaowei Yang, Lu Pang, and Tiejun Huang. Deep relative distance learning: Tell the difference between similar vehicles. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2167 – 2175, June 2016. 4, 88

[113] Tie-Yan Liu. Learning to rank for information retrieval. *Foundations and Trends in Information Retrieval*, 3(3):225–331, 2009. 19

[114] Yang Liu, Qingchao Chen, Wei Chen, and Ian Wassell. Dictionary learning inspired deep network for scene recognition. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2018. 50

[115] Ziwei Liu, Ping Luo, Shi Qiu, Xiaogang Wang, and Xiaoou Tang. Deepfashion: Powering robust clothes recognition and retrieval with rich annotations. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1096 – 1104, June 2016. 4, 24, 69, 75, 80, 93

[116] David G Lowe. Object recognition from local scale-invariant features. In *The IEEE International Conference on Computer Vision (ICCV)*, volume 2, pages 1150–1157. Ieee, 1999. 5

[117] David G Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision (IJCV)*, 60(2):91–110, 2004. 4, 5, 6, 50

[118] Bangalore S Manjunath and Wei-Ying Ma. Texture features for browsing and retrieval of image data. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 18(8):837–842, 1996. 4, 5

[119] Chengzhi Mao, Ziyuan Zhong, Junfeng Yang, Carl Vondrick, and Baishakhi Ray. Metric learning for adversarial robustness. In *Advances in Neural Information Processing Systems (NIPS)*, pages 478–489, 2019. 14

[120] Robert K McConnell. Method of and apparatus for pattern recognition. In *US Patent 4,567,610*, Jan 1986. 4, 5

[121] Brian McFee and Gert R Lanckriet. Metric learning to rank. In *International Conference on Machine Learning (ICML)*, pages 775–782, 2010. 19

[122] Yair Movshovitz-Attias, Alexander Toshev, Thomas K. Leung, Sergey Ioffe, and Saurabh Singh. No fuss distance metric learning using proxies. In *The IEEE International Conference on Computer Vision (ICCV)*, pages 360 – 368, Oct 2017. 20, 21, 22, 61, 90, 96

[123] Kevin Musgrave, Serge Belongie, and Ser-Nam Lim. A metric learning reality check. *arXiv preprint arXiv:2003.08505*, 2020. 106

[124] Yuji Nakatsukasa and Nicholas J Higham. Stable and efficient spectral divide and conquer algorithms for the symmetric eigenvalue decomposition and the svd. *SIAM Journal on Scientific Computing*, 35(3):A1325–A1349, 2013. 37

[125] Hyeonwoo Noh, Andre Araujo, Jack Sim, Tobias Weyand, and Bohyung Han. Large-scale image retrieval with attentive deep local features. In *The IEEE International Conference on Computer Vision (ICCV)*, Oct 2017. 6

[126] Hyun Oh Song, Stefanie Jegelka, Vivek Rathod, and Kevin Murphy. Deep metric learning via facility location. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2206 – 2214, July 2017. 17, 19

[127] Hyun Oh Song, Yu Xiang, Stefanie Jegelka, and Silvio Savarese. Deep metric learning via lifted structured feature embedding. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4004 – 4012, June 2016. 4, 17, 18, 24, 57, 61, 69, 74, 75, 80, 93

[128] Dusan Omercevic, Ondrej Drbohlav, and Ales Leonardis. High-dimensional feature matching: employing the concept of meaningful nearest neighbors. In *The IEEE International Conference on Computer Vision (ICCV)*, pages 1–8. IEEE, 2007. 4, 6

[129] Michael Opitz, Georg Waltner, Horst Possegger, and Horst Bischof. Bier - boosting independent embeddings robustly. In *The IEEE International Conference on Computer Vision (ICCV)*, pages 5199 – 5208, Oct 2017. 22, 61, 62, 66, 67, 74, 76, 81, 88

[130] M. Opitz, G. Waltner, H. Possegger, and H. Bischof. Deep metric learning with bier: Boosting independent embeddings robustly. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 42(2):276–290, Feb 2020. 22, 67, 74, 81, 94

[131] Nikolaos Passalis and Anastasios Tefas. Learning bag-of-features pooling for deep convolutional neural networks. In *The IEEE International Conference on Computer Vision (ICCV)*, Oct 2017. 4, 9, 50

[132] Xavier Pennec, Pierre Fillard, and Nicholas Ayache. A riemannian framework for tensor computing. *International Journal of Computer Vision (IJCV)*, 66(1):41–66, 2006. 30, 33, 34

[133] F. Perronnin, Y. Liu, J. Sánchez, and H. Poirier. Large-scale image retrieval with compressed fisher vectors. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3384–3391, June 2010. 33

[134] Florent Perronnin, Jorge Sánchez, and Thomas Mensink. Improving the fisher kernel for large-scale image classification. In *The European Conference on Computer Vision (ECCV)*, pages 143–156, 2010. 4, 6, 7, 50

[135] Ninh Pham and Rasmus Pagh. Fast and scalable polynomial kernels via explicit feature maps. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 239–247, 2013. 38, 39

[136] J. Philbin, O. Chum, M. Isard, J. Sivic, and A. Zisserman. Object retrieval with large vocabularies and fast spatial matching. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–8, June 2007. 6, 23, 55

[137] James Philbin, Ondrej Chum, Michael Isard, Josef Sivic, and Andrew Zisserman. Lost in quantization: Improving particular object retrieval in large scale image databases. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–8. IEEE, 2008. 23, 55

[138] David Picard. Preserving local spatial information in image similarity using tensor aggregation of local features. In *The IEEE International Conference on Image Processing (ICIP)*, September 2016. 6, 8, 45, 50, 51, 52, 55

[139] David Picard and Philippe-Henri Gosselin. Improving Image Similarity With Vectors of Locally Aggregated Tensors. In *The IEEE International Conference on Image Processing (ICIP)*, 2011. 6, 7, 45, 51, 52, 56, 57

[140] David Picard and Philippe-Henri Gosselin. Efficient image signatures and similarities using tensor products of local descriptors. *Computer Vision and Image Understanding*, 2013. 6, 7, 45, 51, 52, 57

[141] Qi Qian, Lei Shang, Baigui Sun, Juhua Hu, Hao Li, and Rong Jin. Softtriple loss: Deep metric learning without triplet sampling. In *The IEEE International Conference on Computer Vision (ICCV)*, October 2019. 94, 95

[142] Rafael S. Rezende, Joaquin Zepeda, Jean Ponce, Francis Bach, and Patrick Perez. Kernel square-loss exemplar machines for image retrieval. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017. 55

[143] Oren Rippel, Manohar Paluri, Piotr Dollar, and Lubomir Bourdev. Metric learning with adaptive density discrimination. In *International Conference on Learning Representations (ICLR)*, May 2016. 17, 74, 84, 96

[144] Karsten Roth, Biagio Brattoli, and Bjorn Ommer. Mic: Mining interclass characteristics for improved metric learning. In *The IEEE International Conference on Computer Vision (ICCV)*, October 2019. 94, 95

[145] Karsten Roth, Timo Milbich, Samarth Sinha, Prateek Gupta, Bjoern Ommer, and Joseph Paul Cohen. Revisiting training strategies and generalization performance in deep metric learning. In *International Conference on Machine Learning (ICML)*, 2020. 106

[146] Soumava Kumar Roy, Mehrtash Harandi, Richard Nock, and Richard Hartley. Siamese networks: The tale of two manifolds. In *The IEEE International Conference on Computer Vision (ICCV)*, pages 3046 – 3055, October 2019. 26, 67

[147] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. Orb: An efficient alternative to sift or surf. In *The IEEE International Conference on Computer Vision (ICCV)*, pages 2564–2571, 2011. 4, 5

[148] Yossi Rubner, Carlo Tomasi, and Leonidas J Guibas. The earth mover's distance as a metric for image retrieval. *International Journal of Computer Vision (IJCV)*, 40(2):99–121, 2000. 4

[149] Artsiom Sanakoyeu, Vadim Tschernezki, Uta Buchler, and Bjorn Ommer. Divide and conquer the embedding space for metric learning. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 471 – 480, June 2019. 22, 23, 94, 95

[150] Jorge Sánchez, Florent Perronnin, Thomas Mensink, and Jakob Verbeek. Image classification with the fisher vector: Theory and practice. *International Journal of Computer Vision (IJCV)*, 105(3):222–245, Dec. 2013. 4, 6, 7, 50, 51

[151] Stefan Schneider, Graham W Taylor, Stefan Linquist, and Stefan C Kremer. Past, present and future approaches using computer vision for animal re-identification from camera trap data. *Methods in Ecology and Evolution*, 10(4):461–470, 2019. 4

[152] Florian Schroff, Dmitry Kalenichenko, and James Philbin. Facenet: A unified embedding for face recognition and clustering. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 815 – 823, June 2015. 4, 16, 74, 88, 90, 93

[153] Ali Sharif Razavian, Hossein Azizpour, Josephine Sullivan, and Stefan Carlsson. Cnn features off-the-shelf: An astounding baseline for recognition. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, June 2014. 4, 5, 6

[154] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations (ICLR)*, 2015. 33, 55

[155] Josef Sivic and Andrew Zisserman. Video google: A text retrieval approach to object matching in videos. In *The IEEE International Conference on Computer Vision (ICCV)*, pages 1470–1477, October 2003. 4

[156] Lene Theil Skovgaard. A riemannian geometry of the multivariate normal model. *Scandinavian journal of statistics*, pages 211–223, 1984. 43

[157] Jake Snell, Kevin Swersky, and Richard Zemel. Prototypical networks for few-shot learning. In *Advances in Neural Information Processing Systems (NIPS)*, pages 4077–4087, 2017. 14

[158] Kihyuk Sohn. Improved deep metric learning with multi-class n-pair loss objective. In *Advances in Neural Information Processing Systems (NIPS)*, pages 1857–1865, 2016. 17, 18, 61, 62, 74

[159] Bharath K. Sriperumbudur, Arthur Gretton, Kenji Fukumizu, Bernhard Schölkopf, and Gert R.G. Lanckriet. Hilbert space embeddings and metrics on probability measures. *J. Mach. Learn. Res.*, Aug 2010. 80

[160] Yumin Suh, Bohyung Han, Wonsik Kim, and Kyoung Mu Lee. Stochastic class-based hard example mining for deep metric learning. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 7251 – 7259, June 2019. 17, 20, 21, 26

[161] Yumin Suh, Jingdong Wang, Siyu Tang, Tao Mei, and Kyoung Mu Lee. Part-aligned bilinear representations for person re-identification. In *The European Conference on Computer Vision (ECCV)*, September 2018. 40

[162] Michael J Swain and Dana H Ballard. Color indexing. *International Journal of Computer Vision (IJCV)*, 7(1):11–32, 1991. 4

[163] C. Szegedy, Wei Liu, Yangqing Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1 – 9, June 2015. 22, 23, 66, 80, 89, 90, 91

[164] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2818 – 2826, June 2016. 33

[165] P. Tang, X. Wang, B. Shi, X. Bai, W. Liu, and Z. Tu. Deep fishernet for image classification. *IEEE Transactions on Neural Networks and Learning Systems*, 30(7):2244–2250, July 2019. 4, 9, 50

[166] Michael Taylor, John Guiver, Stephen Robertson, and Tom Minka. Softrank: optimizing non-smooth rank metrics. In *Proceedings of the 2008 International Conference on Web Search and Data Mining*, pages 77–86, 2008. 19

[167] Joshua B. Tenenbaum and William T. Freeman. Separating style and content. In *Advances in Neural Information Processing Systems (NIPS)*, pages 662–668, 1997. 30

[168] Engin Tola, Vincent Lepetit, and Pascal Fua. Daisy: An efficient dense descriptor applied to wide-baseline stereo. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 32(5):815–830, 2009. 5

[169] Giorgos Tolias, Ronan Sicre, and Hervé Jégou. Particular object retrieval with integral max-pooling of cnn activations. In *International Conference on Learning Representations (ICLR)*, 2016. 9

[170] Pavan Turaga, Ashok Veeraraghavan, and Rama Chellappa. Statistical analysis on stiefel and grassmann manifolds with applications in computer vision. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–8. IEEE, 2008. 44

[171] Oncel Tuzel, Fatih Porikli, and Peter Meer. Pedestrian detection via classification on riemannian manifolds. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 30(10):1713–1727, 2008. 30, 33

[172] Evgeniya Ustinova, Yaroslav Ganin, and Victor Lempitsky. Multi-region bilinear convolutional neural networks for person re-identification. In *2017 14th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS)*, pages 1–6, 2017. 30

[173] Evgeniya Ustinova and Victor Lempitsky. Learning deep embeddings with histogram loss. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems (NIPS)*, pages 4170–4178, 2016. 17, 18, 61, 62, 82, 90, 93

[174] C. Wah, S. Branson, P. Welinder, P. Perona, and S. Belongie. The caltech-ucsd birds-200-2011 dataset. Technical Report CNS-TR-2011-001, California Institute of Technology, 2011. 24, 61, 69, 75, 80, 82, 93

[175] Jian Wang, Feng Zhou, Shilei Wen, Xiao Liu, and Yuanqing Lin. Deep metric learning with angular loss. In *The IEEE International Conference on Computer Vision (ICCV)*, pages 2612 – 2620, Oct 2017. 17, 19, 67, 74, 81

[176] Liwei Wang, Yin Li, and Svetlana Lazebnik. Learning deep structure-preserving image-text embeddings. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5005–5013, 2016. 4

[177] Qilong Wang, Zilin Gao, Jiangtao Xie, Wangmeng Zuo, and Peihua Li. Global gated mixture of second-order pooling for improving deep convolutional neural networks. In *Advances in Neural Information Processing Systems (NIPS)*, pages 1277–1286, 2018. 43

[178] Qilong Wang, Peihua Li, Qinghua Hu, Pengfei Zhu, and Wangmeng Zuo. Deep global generalized gaussian networks. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019. 43

[179] Qilong Wang, Peihua Li, and Lei Zhang. G2denet: Global gaussian distribution embedding network and its application to visual recognition. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017. 43

[180] Qilong Wang, Peihua Li, Wangmeng Zuo, and Lei Zhang. Raid-g: Robust estimation of approximate infinite dimensional gaussian with application to material recognition. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016. 35

[181] Wei Wang, Zheng Dang, Yinlin Hu, Pascal Fua, and Mathieu Salzmann. Backpropagation-friendly eigendecomposition. In *Advances in Neural Information Processing Systems (NIPS)*, pages 3156–3164, 2019. 37

[182] Xun Wang, Xintong Han, Weilin Huang, Dengke Dong, and Matthew R. Scott. Multi-similarity loss with general pair weighting for deep metric learning. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5022 – 5030, June 2019. 17, 18, 74

[183] Xinshao Wang, Yang Hua, Elyor Kodirov, Guosheng Hu, Romain Garnier, and Neil M. Robertson. Ranked list loss for deep metric learning. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5207 – 5216, June 2019. 17, 81, 88, 94, 95

[184] Yu-Xiong Wang, Ross Girshick, Martial Hebert, and Bharath Hariharan. Low-shot learning from imaginary data. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018. 14

[185] Jônatas Wehrmann and Rodrigo C. Barros. Bidirectional retrieval made simple. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 7718 – 7726, June 2018. 88

[186] Xing Wei, Yue Zhang, Yihong Gong, Jiawei Zhang, and Nanning Zheng. Grassmann pooling as compact homogeneous bilinear pooling for fine-grained visual classification. In *The European Conference on Computer Vision (ECCV)*, September 2018. 30, 40, 44

[187] Kilian Q Weinberger, John Blitzer, and Lawrence K Saul. Distance metric learning for large margin nearest neighbor classification. In *Advances in Neural Information Processing Systems (NIPS)*, pages 1473–1480, 2006. 14

[188] Kilian Q Weinberger and Lawrence K Saul. Distance metric learning for large margin nearest neighbor classification. *Journal of Machine Learning Research*, 10(Feb):207–244, 2009. 14

[189] Yung-Chow Wong. Differential geometry of grassmann manifolds. *Proceedings of the National Academy of Sciences of the United States of America*, 57(3):589, 1967. 44

[190] Chao-Yuan Wu, R Manmatha, Alexander J Smola, and Philipp Krähenbühl. Sampling matters in deep embedding learning. In *The IEEE International Conference on Computer Vision (ICCV)*, 2017. 61

[191] Chang Xiao, Peilin Zhong, and Changxi Zheng. Bourgan: Generative networks with metric embeddings. In *Advances in Neural Information Processing Systems (NIPS)*, pages 2269–2280, 2018. 14

[192] Eric P. Xing, Michael I. Jordan, Stuart J Russell, and Andrew Y. Ng. Distance metric learning with application to clustering with side-information. In *Advances in Neural Information Processing Systems (NIPS)*, pages 521–528, 2003. 14

[193] Xinyi Xu, Yanhua Yang, Cheng Deng, and Feng Zheng. Deep asymmetric metric learning via rich relationship mining. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4076 – 4085, June 2019. 17, 20, 67, 74, 81, 88, 94, 95

[194] Hong Xuan, Richard Souvenir, and Robert Pless. Deep randomized ensembles for metric learning. In *The European Conference on Computer Vision (ECCV)*, pages 751 – 762, September 2018. 22

[195] Chaojian Yu, Xinyi Zhao, Qi Zheng, Peng Zhang, and Xinge You. Hierarchical bilinear pooling for fine-grained visual recognition. In *The European Conference on Computer Vision (ECCV)*, September 2018. 43

[196] Kaicheng Yu and Mathieu Salzmann. Statistically-motivated second-order pooling. In *The European Conference on Computer Vision (ECCV)*, September 2018. 40

[197] Yuhui Yuan, Kuiyuan Yang, and Chao Zhang. Hard-aware deeply cascaded embedding. In *The IEEE International Conference on Computer Vision (ICCV)*, pages 814 – 823, Oct 2017. 22, 61, 62, 67, 74, 81

[198] Yisong Yue, Thomas Finley, Filip Radlinski, and Thorsten Joachims. A support vector method for optimizing average precision. In *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 271–278, 2007. 19

[199] Andrew Zhai and Hao-Yu Wu. Classification is a strong baseline for deep metric learning. In *Proceedings of the British Machine Vision Conference (BMVC)*, 2019. 106

[200] Yan Zhang, Siyu Tang, Krikamol Muandet, Christian Jarvers, and Heiko Neumann. Local temporal bilinear pooling for fine-grained action parsing. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 12005–12015, 2019. 30

[201] Yiru Zhao, Zhongming Jin, Guo-jun Qi, Hongtao Lu, and Xian-sheng Hua. An adversarial approach to hard triplet generation. In *The European Conference on Computer Vision (ECCV)*, pages 508 – 524, September 2018. 20, 21, 67, 88, 89, 96

[202] Wenzhao Zheng, Zhaodong Chen, Jiwen Lu, and Jie Zhou. Hardness-aware deep metric learning. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 72 – 81, June 2019. 20, 21, 67, 74, 81, 88, 89, 94, 95, 96

[203] Bolei Zhou, Aditya Khosla, Agata Lapedriza, Aude Oliva, and Antonio Torralba. Learning deep features for discriminative localization. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2921 – 2929, June 2016. 75

[204] B. Zhou, A. Lapedriza, A. Khosla, A. Oliva, and A. Torralba. Places: A 10 million image database for scene recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 40(6):1452–1464, June 2018. 24, 55

[205] Dengyong Zhou, Jason Weston, Arthur Gretton, Olivier Bousquet, and Bernhard Schölkopf. Ranking on data manifolds. In *Advances in Neural Information Processing Systems (NIPS)*, pages 169–176, 2004. 20, 22, 23

[206] Jiahuan Zhou, Pei Yu, Wei Tang, and Ying Wu. Efficient online local metric adaptation via negative samples for person re-identification. In *The IEEE International Conference on Computer Vision (ICCV)*, pages 2439 – 2447, Oct 2017. 4