LEARNING DEEP EMBEDDINGS BY LEARNING TO RANK

by

KUN HE

B.Eng., Zhejiang University, 2010
M.Sc., Boston University, 2014

Submitted in partial fulfillment of the requirements for the degree of
Doctor of Philosophy

2018
Approved by

First Reader
______________________________
Stanley Sclaroff, PhD
Professor of Computer Science

Second Reader
______________________________
Kate Saenko, PhD
Associate Professor of Computer Science

Third Reader
______________________________
Brian Kulis, PhD
Associate Professor of Computer and Electrical Engineering
Acknowledgments

First, I must express my deepest gratitude to my advisor, Prof. Stan Sclaroff, for his mentoring and patience over the years. Being his student, I was the fortunate recipient of a tremendous amount of academic freedom, as well as rigorous training. His never-waning enthusiasm and high standards trickle down into our everyday interactions, and from him I have learned a great deal about conducting academic research, writing, and critical thinking in general. I am certain that these will continue to reward me for many years to come.

I am also very grateful to my thesis committee members, Prof. Margrit Betke, Prof. Kate Saenko, Prof. Brian Kulis, and Prof. Leonid Sigal. With each of them, I have had many research-oriented interactions, and their feedback is very helpful for improving this thesis. I also thank Dr. Hao Jiang, Prof. Octavia Camps, Prof. Pedro F. Felzenszwalb, and Dr. Yan Lu, for mentoring me at various points in my research career so far.

During my PhD study, I had the privilege of interacting with, and learning from, numerous past and current members of the IVC group. In particular, I would like to thank my main collaborators, Dr. Fatih Çakir and Sarah Adel Bargal, for many of the formulations presented here came out of our long discussions, without which this thesis would be very different. I have also benefited greatly from collaborations and interactions with Dr. Qinxun Bai, Dr. Shugao Ma, Dr. Jianming Zhang, Prof. Danna Gurari, Dr. Zheng Wu, Dr. Tai-Peng Tian, Dr. Ashwin Thangali, Dr. Vitaly Ablavsky, Prof. Liliana Lo Presti, Huijuan Xu, and Xide Xia. In addition, I thank the following IVC members and alumni: Dr. Rui Li, Dr. Diane H. Theriault, Dr. Mehrnoosh Sameki, Dr. Mikhail Breslav, Dr. Andrea Zunino, Dr. Antonio Hernández-Vela, Wenxin Feng, Ajjen Joshi, Vasili Ramanishka, Ben Usman, Xingchao Peng, Donghyun Kim, and Fred Qi Feng.
I owe very much to my family, who made my PhD study far easier than it might have been otherwise. First, I am more than fortunate to be accompanied by my amazing wife Shanshan Dai, whose support, encouragements, and positivity have been there all along this journey. My parents also made selfless sacrifices in their unconditional support for me. Lastly, I thank my son Yunfan, for showing me many of the greatest simple joys in life.

The research reported in this thesis was supported in part by National Science Foundation grant 1029430. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author and do not necessarily reflect the views of the National Science Foundation. Also, we gratefully acknowledge the support from a BU IGNITION award, and gifts from NVIDIA.
ABSTRACT

We study the problem of embedding high-dimensional visual data into low-dimensional vector representations. This is an important component in many computer vision applications involving nearest neighbor retrieval, as embedding techniques not only perform dimensionality reduction, but can also capture task-specific semantic similarities. In this thesis, we use deep neural networks to learn vector embeddings, and develop a gradient-based optimization framework that is capable of optimizing ranking-based retrieval performance metrics, such as the widely used Average Precision (AP) and Normalized Discounted Cumulative Gain (NDCG). Our framework is applied in three applications.

First, we study Supervised Hashing, which is concerned with learning compact binary vector embeddings for fast retrieval, and propose two novel solutions. The first solution optimizes Mutual Information as a surrogate ranking objective, while the other directly optimizes AP and NDCG, based on the discovery of their closed-form expressions for discrete Hamming distances. These optimization problems are NP-hard, therefore we derive their continuous relaxations to enable gradient-based optimization with neural networks. Our solutions establish the state-of-the-art on
several image retrieval benchmarks.

Next, we learn deep neural networks to extract **Local Feature Descriptors** from image patches. Local features are used universally in low-level computer vision tasks that involve sparse feature matching, such as image registration and 3D reconstruction, and their matching is a nearest neighbor retrieval problem. We leverage our AP optimization technique to learn both binary and real-valued descriptors for local image patches. Compared to competing approaches, our solution eliminates complex heuristics, and performs more accurately in the tasks of patch verification, patch retrieval, and image matching.

Lastly, we tackle **Deep Metric Learning**, the general problem of learning real-valued vector embeddings using deep neural networks. We propose a learning to rank solution through optimizing a novel quantization-based approximation of AP. For downstream tasks such as retrieval and clustering, we demonstrate promising results on standard benchmarks, especially in the few-shot learning scenario, where the number of labeled examples per class is limited.
Contents

1 Introduction ................................................. 1
   1.1 Learning Embeddings for Nearest Neighbor Retrieval .............. 1
       1.1.1 Supervised Hashing ..................................... 3
       1.1.2 Learning Local Image Descriptors ......................... 5
       1.1.3 Deep Metric Learning .................................... 6
   1.2 Contributions ............................................ 8
   1.3 Organization of Thesis ..................................... 10
   1.4 List of Related Papers ..................................... 12

2 Mathematical Background .................................. 13
   2.1 Learning Deep Embeddings ..................................... 13
       2.1.1 Deep Supervised Hashing .................................. 14
       2.1.2 Deep Metric Learning .................................... 15
   2.2 Ranking-based Evaluation .................................... 15

3 Related Work .............................................. 19
   3.1 Vector Embedding using Neural Networks ............................ 19
   3.2 Learning to Rank .......................................... 20
   3.3 Hashing for Nearest Neighbor Retrieval .......................... 23
   3.4 Learning Local Image Descriptors ................................ 26

4 Hashing with Mutual Information ............................ 30
   4.1 Optimizing Mutual Information .................................. 34
4.1.1 Continuous Relaxation ........................................ 35
4.1.2 End-to-End Optimization .................................. 36
4.1.3 Efficient Minibatch Backpropagation ..................... 38

4.2 Experiments ...................................................... 40
  4.2.1 Datasets and Setup ....................................... 40
  4.2.2 Results ...................................................... 43
  4.2.3 Discussions ................................................ 46

4.3 Summary .......................................................... 49

5 Hashing as Tie-Aware Learning to Rank ......................... 51
  5.1 Tie-Awareness in Hashing .................................... 53
  5.2 Optimizing Tie-Aware Ranking Metrics ....................... 55
    5.2.1 Continuous Relaxation .................................. 56
    5.2.2 End-to-end Optimization ................................. 56
  5.3 Implementation Details ...................................... 58
    5.3.1 A Simplified Version of Tie-Aware AP .................. 59
    5.3.2 A Lower Bound for Tie-Aware DCG ..................... 60
  5.4 Experiments .................................................... 61
    5.4.1 Datasets and Setup ..................................... 61
    5.4.2 Binary Relevance: AP ................................... 62
    5.4.3 Graded Relevance: NDCG ............................... 65
    5.4.4 Tightening the Continuous Relaxation .................. 66
    5.4.5 Effects of Tie-Braking Strategies ....................... 68
  5.5 Summary ....................................................... 69

6 Local Descriptors Optimized for Average Precision .......... 71
  6.1 Feature Matching is Nearest Neighbor Retrieval .......... 73
  6.2 Optimizing Binary and Real-valued Descriptors .......... 74
List of Tables

4.1 Results on CIFAR-10 and NUS-WIDE with VGG-F . . . . . . . . . . 44
4.2 mAP@1K values on ImageNet100 using AlexNet . . . . . . . . . . . 45
4.3 22K LabelMe results with GIST features . . . . . . . . . . . . . . . 45
5.1 AP comparison on CIFAR-10 and NUS-WIDE datasets . . . . . . . 64
5.2 AP@1000 comparison on the ImageNet100 dataset . . . . . . . . . . 65
5.3 NDCG comparison on the NUS-WIDE and LabelMe datasets . . . . 67
5.4 Comparison between continuous relaxation strategies . . . . . . . 68
6.1 Patch verification performance on the UBC Phototour dataset . . . 82
6.2 Patch retrieval performance on the RomePatches dataset . . . . . . 87
7.1 Performance comparison on the Stanford Online Products dataset . 104
7.2 Ablation study on the batch size $M$ . . . . . . . . . . . . . . . . . 106
## List of Figures

1.1 Supervised hashing for nearest neighbor retrieval .......................... 4
1.2 Detection and matching of local image features ......................... 5
1.3 Metric learning with deep vector embeddings .......................... 7

4.1 Overview of the proposed MIHash method ............................. 32
4.2 The distribution separating effect of MIHash .......................... 46
4.3 Correlation between MI and AP ..................................... 47
4.4 Visualization of learned hash codes .................................. 48

5.1 Tie-aware ranking metrics for hashing ................................. 52
5.2 The flow of computation in the proposed TALR method ............... 58
5.3 Effects of tie-breaking in evaluating hashing algorithms ............. 69

6.1 An example local feature-based image matching pipeline ............. 72
6.2 Patch alignment using the Spatial Transformer ........................ 77
6.3 Label mining in the HPatches dataset ................................ 79
6.4 Examples from patch-based datasets .................................. 80
6.5 Influence of training batch size ..................................... 83
6.6 Results on the HPatches dataset ..................................... 85
6.7 Image matching performance on the Oxford dataset .................. 88

7.1 AP as a general objective for metric learning ......................... 94
7.2 Quantization of the precision-recall curve ............................ 97
7.3 Example images from the Stanford Online Products dataset ........ 100
<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AP</td>
<td>Average Precision</td>
</tr>
<tr>
<td>CDF</td>
<td>Cumulative Distribution Function</td>
</tr>
<tr>
<td>CNN</td>
<td>Convolutional Neural Network</td>
</tr>
<tr>
<td>DCG</td>
<td>Discounted Cumulative Gain</td>
</tr>
<tr>
<td>DNN</td>
<td>Deep Neural Network</td>
</tr>
<tr>
<td>DOAP</td>
<td>Descriptors Optimized for Average Precision</td>
</tr>
<tr>
<td>DOF</td>
<td>Degrees of Freedom</td>
</tr>
<tr>
<td>FPR</td>
<td>False Positive Rate</td>
</tr>
<tr>
<td>IR</td>
<td>Information Retrieval</td>
</tr>
<tr>
<td>KNN</td>
<td>K-Nearest Neighbor</td>
</tr>
<tr>
<td>LSH</td>
<td>Locality Sensitive Hashing</td>
</tr>
<tr>
<td>LSTM</td>
<td>Long Short-Term Memory</td>
</tr>
<tr>
<td>LTR</td>
<td>Learning to Rank</td>
</tr>
<tr>
<td>MI</td>
<td>Mutual Information</td>
</tr>
<tr>
<td>MIHash</td>
<td>Hashing with Mutual Information</td>
</tr>
<tr>
<td>mAP</td>
<td>mean Average Precision</td>
</tr>
<tr>
<td>NDCG</td>
<td>Normalized Discounted Cumulative Gain</td>
</tr>
<tr>
<td>NMI</td>
<td>Normalized Mutual Information</td>
</tr>
<tr>
<td>PDF</td>
<td>Probability Distribution Function</td>
</tr>
</tbody>
</table>
RANSAC ............ Random Sample Consensus
RPN ............... Region Proposal Network
SGD ............... Stochastic Gradient Descent
SIFT ............... Scale Invariant Feature Transform
SLAM ............... Simultaneous Localization and Mapping
SVM ............... Support Vector Machine
TALR ............... Tie-Aware Learning to Rank
Chapter 1

Introduction

Computer vision systems often face the tasks of storing, processing, and searching through high-dimensional visual data, such as mega-pixel images and high-definition videos. Examples include content-based image and video retrieval [4, 107], person and object recognition in photo collections [78, 100], and action detection and classification in surveillance video [68, 76]. To make such systems large-scale, it is desirable to map the high-dimensional data objects to low-dimensional vector representations, both to reduce storage costs and to allow for efficient processing. High-dimensional data also pose challenges for semantic-level understanding: owing to the curse of dimensionality [13], standard similarity metrics in high-dimensional spaces do not necessarily reflect the true perceived similarities between visual objects. Therefore, another requirement for the mapping process is that it should encode desired task-specific similarities, such that objects that are semantically similar are mapped to close neighbors in a low-dimensional vector space.

1.1 Learning Embeddings for Nearest Neighbor Retrieval

In this thesis, we consider the problem of learning low-dimensional vector embeddings of high-dimensional visual data. First and foremost, this task has direct connections to dimensionality reduction [12, 40, 132], which is a well-studied problem in its own right. However, considering that vector embeddings find applications in many different scenarios, each having different goals and forms of supervision, we argue that it is
likely suboptimal to take a purely unsupervised, “one-size-fits-all” dimensionality reduction approach. Instead, we aim to optimize the vector embeddings in a task-driven manner, by learning from data and their accompanying supervision signals.

The primary application scenario that we consider in this thesis is nearest neighbor retrieval, which is a core component in many computer vision applications. Given a query object (e.g. image search query) and a database of indexed objects (e.g. a database of images), a standard retrieval procedure is to first encode the query into a vector representation. Then, distances between the query vector and database vectors are evaluated, either exhaustively over the entire database, or on a subset of the database. The result is a ranking of database items by increasing distance to the query, and finally, objects that have the lowest distances to the query are returned as nearest neighbors. In computer vision, this paradigm is naturally instantiated in problems like image and video retrieval.

It is crucial to have a high-quality distance metric for nearest neighbor retrieval, and when distances are evaluated between vector embeddings, it becomes necessary to optimize the embeddings so as to achieve high quality. Inspired by the recent success of deep neural networks (DNNs) in important computer vision problems such as image classification [85], in this thesis we focus on using DNNs to optimize vector embeddings, as DNNs provide excellent function approximation abilities when coupled with appropriate hardware acceleration. Consequently, our main optimization machinery would be Stochastic Gradient Descent (SGD) [15, 130], which is the standard tool for training DNNs.

In supervised learning of vector embeddings, supervision is commonly given in the form of pairwise affinities. Pairwise affinity labels are easy to obtain (e.g. from agreement of class labels), yet they enable learning powerful representations. In its simplest form, we are given pairs of objects, with binary labels indicating their pairwise rela-
tionships as either “similar” or “dissimilar”. Alternatively, the similarity labels could have multiple levels, or “grades”, indicating different strengths. Intuitively, in formu-
ating the learning problem, we would like the learned vector embeddings to evaluate to small distances between similar pairs, and large distances between dissimilar pairs. In this thesis, based on the “learning to rank” formalism \[105\], we design a principled learning framework to capture such intuitions. This framework is applied to learn deep vector embeddings in several problem domains, which we discuss next.

1.1.1 Supervised Hashing

We start by considering the problem of supervised hashing, where the goal is to learn binary vector embeddings, or hash codes, of data. Please see Figure 1·1 for an illustration. A natural use case for hashing is in content-based retrieval in large databases, including for images, video, audio, or even between different modalities. The Hamming distance between binary vectors can be evaluated efficiently using bitwise operations on modern CPUs, which enables fast approximate nearest neighbor search; the approximate search can also be used as a filtering step to return a shortlist of candidate retrievals, before more expensive and time-consuming distance metrics are applied. In addition, with compact codes, the memory and storage footprint for the entire database can be significantly reduced.

Although hashing can be viewed as a binarization or vector quantization technique, this view implies that there is some original distance metric that is being preserved. In fact, this is true for most unsupervised hashing methods. But as we have argued, distance metrics between high-dimensional raw data representations may not be semantically meaningful. Therefore, we consider supervised hashing, and learn hash codes (and in turn, a Hamming distance metric) so as to best agree with supervision signals. For content-based image retrieval, a common case is that images with the
In supervised hashing, our goal is to learn a hash mapping $\Phi$ from the original high-dimensional feature space to some low-dimensional Hamming space. The resulting binary vector embeddings, or hash codes, enable fast approximate nearest neighbor retrieval in large databases. In this thesis, we parameterize the hash mapping $\Phi$ using deep neural networks.

It is worth noting that hashing is a discrete problem, as the hash codes are binary by nature. For any objective defined over the hash code assignments in a finite training set, in order to optimize it, the worst case is to try all possible bit combinations, except in cases where special structures exist. Therefore, the optimization problems in supervised hashing are often NP-hard. Unfortunately, this creates a mismatch when learning using DNNs, which are continuously parameterized and rely on gradient-based optimization. To tackle this challenge, in our proposed supervised hashing solutions, we follow a standard recipe: first derive continuously differentiable relaxations to the discrete objectives, and then apply gradient methods. At test time, the learned representations, which are continuous, are thresholded to give binary codes.
1.1.2 Learning Local Image Descriptors

The next application scenario that we consider is image matching in low-level computer vision. Traditionally, this is done by identifying localized “interest points” (such as corners, blobs, etc.) in images, and finding their correspondences across images. When local matches between images are established, global geometric correspondences or transformations can be estimated using robust estimation techniques, such as RANSAC [49]. Please see Figure 1·2 for an example.

The local feature matching pipeline plays a fundamental role in many low-level computer vision tasks, such as image registration, camera localization, structure from motion, 3D reconstruction, and Simultaneous Localization and Mapping (SLAM).
SIFT \cite{108} is perhaps the most well-known such pipeline, which consists of carefully handcrafted solutions for 1) detection of interest points, 2) extraction of local feature descriptors from image patches around interest points, and 3) matching between the descriptors. Many learning-based alternatives have been proposed, and they often show improvements on certain benchmark datasets. However, according to recent studies \cite{139}, when the entire pipeline is considered, handcrafted solutions like SIFT still perform competitively in complicated low-level vision tasks. This has motivated more research efforts in optimizing low-level vision pipelines using deep learning.

Among the various stages in feature-based low-level vision pipelines, we are particularly interested in a key stage: extraction of local feature descriptors. This is exactly a vector embedding problem: the input is a local image patch, and the output is a fixed-length vector representation. Differently from the image/video retrieval problem where the desired distance metric is usually based on content, a good distance metric for local feature descriptors should be geared towards the robustness of feature matching, especially under viewpoint and illumination variations in the scene. Many other solutions for extracting local feature descriptors have also been developed over the years, including handcrafted ones such as LBP \cite{1}, RootSIFT \cite{6}, and SURF \cite{11}, and ones that are learned with shallow models, \textit{e.g.} \cite{143}. Instead, our focus in this thesis is on learning local feature descriptors using DNNs.

### 1.1.3 Deep Metric Learning

Finally, we focus on the deep metric learning problem, namely, learning real-valued vector embeddings using deep neural networks.

Metric learning \cite{86} is a general umbrella term for learning distance metrics from data, and the vast majority of existing methods work by learning a transformation to the input data, and then applying standard distance metrics such as the Euclidean
Figure 1.3: Metric learning with deep neural networks is essentially a vector embedding problem: inputs $x$ and $y$ are embedded into vector representations $\phi(x)$ and $\phi(y)$, which are then plugged into standard distance metrics, e.g. the Euclidean distance. In this formulation, the embedding function $\phi$ is modeled as a deep neural network, which contains all the learnable parameters.

Distance. Depending on the type of transformation, metric learning approaches can be grouped into linear and nonlinear methods. Traditionally, linear methods, especially those that learn Mahalanobis distances [109], are extensively studied, partly due to the ease in formulating tractable optimization problems. For nonlinear metric learning, kernel methods have been studied to some extent, and recently deep metric learning methods using DNNs have received much attention. While kernel-based metric learning does not necessarily involve vector embeddings, deep metric learning, on the other hand, is essentially a vector embedding problem, since the input data is transformed by a neural network to a vector representation, as shown in Figure 1.3.

In fact, the two previously discussed problems, supervised hashing and learning local feature descriptors, can both be regarded as special cases of deep metric learning. Despite the generality in its formulations, metric learning is still a task-driven problem: the learned distance metric will ultimately be used to perform some specific task. And indeed, it is often motivated by nearest neighbor retrieval or nearest neighbor classification, where the distance metric is of central importance. Metric learning is also useful for producing distance metrics for data clustering. To stay
within the scope of this thesis, we again focus on the nearest neighbor retrieval aspect, and study learning to rank formulations for deep metric learning. Nevertheless, we will show that the distance metrics learned by this approach can also perform well in other related tasks.

1.2 Contributions

In this thesis, we provide novel solutions to all three problems described above. To start with, we propose two novel solutions to the supervised hashing problem. In the first solution, we use an information-theoretic quantity, mutual information, as a quality measure and ranking surrogate for supervised hashing. We then optimize mutual information to learn high-quality binary embeddings. In the second solution, we instead directly optimize commonly used ranking metrics, such as Average Precision (AP) and Normalized Discounted Cumulative Gain (NDCG). For this purpose, we first identify the problem of tied rankings with the discrete Hamming distance, and propose to optimize tie-aware versions of AP and NDCG. Due to the constraint that the learned hash codes must be binary, the optimization problems in both solutions are combinatorial and NP-hard. We employ continuous relaxations and differentiable approximations to enable end-to-end optimization using DNNs. In learning to rank terminology \cite{105}, both of our solutions use loss functions belonging to the listwise ranking loss category, which are superior to a range of simpler losses in the current literature. We establish the current state-of-the-art for supervised hashing on standard image retrieval benchmarks.

Next, we apply our techniques to the problem of learning local feature descriptors. We first discover an equivalence between the matching of local features and nearest neighbor retrieval, which allows us to apply learning to rank formulations again. Specifically, the problem of learning binary local feature descriptors can be treated as
an instance of supervised hashing, and is solved by directly reusing the formulation that we developed. Additionally, to learn real-valued descriptors, we also devise a real-valued extension to our formulation. Lastly, we augment our models with task-specific components, so that the nature of local feature matching is better taken into account. Experimental results on standard benchmarks suggest the superiority of our approach compared to other learned and handcrafted ones. Notably, on two of the benchmarks, our learned descriptors outperform SIFT for the first time.

Finally, we focus on learning real-valued vector embeddings using deep neural networks, which is a special case of metric learning [86]. Building on our gradient-based optimization framework, we propose a new quantization-based formulation for approximately optimizing AP, which is an excellent surrogate objective for metric learning, especially in the few-shot learning scenario. Experiments on standard image benchmarks for nearest neighbor retrieval and classification show the promise of this method.

To summarize, the major contributions of this thesis are as follows:

- A novel supervised hashing method based on optimizing the information-theoretic quantity of mutual information (published in [62] and [64]),
- A novel supervised hashing method based on directly optimizing ranking-based retrieval evaluation metrics (published in [63]),
- A learning to rank approach for learning robust local feature descriptors with deep neural networks (published in [65]),
- A deep metric learning approach based on a novel quantization-based approximation of AP.
1.3 Organization of Thesis

The remainder of thesis is organized as follows.

Chapter 2: Mathematical Background

This chapter defines notation that will be used throughout this thesis, and reviews necessary mathematical background in the following areas: learning vector embeddings, nearest neighbor retrieval, and learning to rank.

Chapter 3: Related Work

This chapter reviews related work and prior art in the following categories: distance metric learning, learning to rank, hashing for nearest neighbor retrieval, and the learning of local features in low-level computer vision.

Chapter 4: Hashing with Mutual Information

This chapter describes our first contribution to the supervised hashing problem. We propose an information-theoretic solution based on optimizing Mutual Information, which as we demonstrate, is a good surrogate ranking objective. We relax the NP-hard optimization problem into a continuous version, and then perform gradient-based optimization with deep neural networks. On standard image retrieval benchmarks, our solution achieves convincing improvements over competing methods.

Chapter 5: Hashing as Tie-Aware Learning to Rank

This chapter describes our second contribution to supervised hashing, where we develop gradient-based methods to directly optimize AP and NDCG. We first observe that the integer-valued Hamming distance results in tied rankings, which induces ambiguity in evaluation of supervised hashing. We then leverage an existing result in the information retrieval literature to derive tie-aware ranking metrics. By deriving
continuous relaxations to the closed-form expressions of tie-aware metrics, we enable their end-to-end optimization. This solution further improves the state-of-the-art on standard image retrieval benchmarks.

Chapter 6: Local Feature Descriptors Optimized for Average Precision

This chapter describes our proposed solution for learning local feature descriptors. We reuse our AP optimization technique for the supervised hashing problem to learn binary descriptors, and develop an extension to also learn real-valued descriptors. Two improvements are introduced, which are designed for the image descriptor matching task: robustness to geometric distortions using spatial transformer networks [73], and mining additional supervision using clustering. The learned descriptors achieve state-of-the-art performance in three benchmark tasks: patch verification, patch retrieval, and image matching.

Chapter 7: Deep Metric Learning via Learning to Rank

This chapter describes our learning to rank solution for the general problem of deep metric learning, or learning real-valued vector embeddings using deep neural networks. We propose a novel formulation, named FastAP, to approximately optimize AP for real-valued embeddings. This formulation obtains competitive results, especially when the few-shot learning assumption holds, i.e. the number of labeled examples available for each class is limited.

Chapter 8: Conclusions

This chapter concludes this thesis. Specifically, it summarizes our major contributions, discusses the limitations of our current work, and proposes directions for future research.
1.4 List of Related Papers

This thesis is based in part on the following publications:


Chapter 2

Mathematical Background

We review necessary mathematical background related to the vector embedding problem and nearest neighbor retrieval, and define notation that will be used in later chapters.

2.1 Learning Deep Embeddings

Let $\mathcal{X} \subset \mathbb{R}^N$ be the feature space. The goal of learning embeddings is to learn a mapping from $\mathcal{X}$ to some embedding space, where the induced distance metric in the learned embedding space has desirable properties. To enforce the desired properties, we consider a standard setup where supervision is given in the form of pairwise relevance. Formally, we assume an relevance oracle $A$, where $A(x_i, x_j) > 0$ if $x_i, x_j \in \mathcal{X}$ are said to be “similar”, with the value indicating the strength of similarity. Otherwise, $x_i$ and $x_j$ are said to be not similar when $A(x_i, x_j) = 0$. Roughly speaking, the goal of embedding learning is to make the induced distance between the embedding vectors of $x_i$ and $x_j$ small if $A(x_i, x_j) > 0$, and large otherwise.

In this thesis, we restrict $A$ to take values from a finite set $V$, which includes two important special cases. First, the case where $V = \{0, 1\}$ is called binary relevance, which is extensively studied in the hashing and metric learning literature. In practice, binary relevance values can often be derived from agreement of class labels, or from thresholding some distance metric in $\mathcal{X}$. The second case is when $V$ consists of non-negative integers, referred to as multi-level relevance or graded relevance. Graded
relevance is a more fine-grained model, and is frequently considered in information retrieval tasks, including in web search engines \[38\].

In this thesis, we parameterize the functional mappings by deep neural networks, as they have recently shown to have superior function approximation capabilities when coupled with appropriate hardware acceleration, and have achieved significant advances in problems such as image classification \[85\]. Consequently, the optimization approaches that we develop would be gradient-based, in order to take advantage of end-to-end training by backpropagation, as well as recent advances in stochastic optimization.

2.1.1 Deep Supervised Hashing

In the supervised hashing problem, we wish to learn a hash mapping \( \Phi : \mathcal{X} \rightarrow \mathcal{H}^b \), where the embedding space \( \mathcal{H}^b \) is taken to be the \( b \)-dimensional Hamming space. This induces the Hamming distance \( d_\Phi : \mathcal{X} \times \mathcal{X} \rightarrow \{0, 1, \ldots, b\} \). Practically, the Hamming distance can be efficiently evaluated by counting bit differences between binary vectors:

\[
d_\Phi(x, x') = \sum_{i=0}^{b} 1[\phi_i(x) \neq \phi(x')]. \tag{2.1}
\]

Since we focus on gradient-based optimization, we adopt an equivalent formulation of the Hamming distance that is more amenable to differentiation:

\[
d_\Phi(x, x') = \frac{1}{2} (b - \Phi(x)\top \Phi(x')), \tag{2.2}
\]

\[
\Phi(x) = (\phi_1(x), \ldots, \phi_b(x)), \quad \phi_i(x) = \text{sgn}(f_i(x; w)) \in \{-1, +1\}, \quad \forall i, \tag{2.3}
\]

where \( f_i \) are the “logits” for each bit, produced by a neural network with parameters \( w \). Usually, they are the activations from the neural network’s output layer.
2.1.2 Deep Metric Learning

In deep metric learning, we wish to learn a real-valued embedding $\Psi : \mathcal{X} \rightarrow \mathbb{R}^d$, where typically $d < N$. As the embedding space now is an Euclidean space, the induced distance metric is naturally the Euclidean distance:

$$d_\Psi(x, x')^2 = \|\Psi(x) - \Psi(x')\|^2$$

$$= \|\Psi(x)\|^2 + \|\Psi(x')\|^2 - 2\Psi(x)^\top\Psi(x').$$

(2.4)

(2.5)

Also, note that if the vectors are $L_2$-normalized, then the distance is simplified as

$$d_\Psi(x, x')^2 = 2 - 2\Psi(x)^\top\Psi(x').$$

(2.6)

Similarly, we parameterize the embedding with neural networks:

$$\Psi(x) = (f_1(x; w), \ldots, f_d(x; w)), f_i(x; w) \in \mathbb{R},$$

(2.7)

where $f_i$ are the activations from the output layer of a neural network.

2.2 Ranking-based Evaluation

There are many ways to evaluate learned embeddings. For example, metric learning is often motivated by nearest neighbor classification or clustering, and therefore can be evaluated in terms of kNN classification or K-means clustering performance. In this thesis, we are instead interested in the nearest neighbor retrieval aspect, which finds applications in a wide range of retrieval tasks.

Specifically, we assume the setup where a query $x_q \in \mathcal{X}$ is retrieved against some database $S \subseteq \mathcal{X}$. Retrieval is performed by ranking the instances in $S$ (referred to as database items or documents in this thesis) by increasing distance to $x_q$, where the distance is induced from the learned embedding. The ranking can be represented
by an index vector $R$, whose elements form a permutation of $\{1, \ldots, |S|\}$.\footnote{For simplicity, we consider the ranking to be over the entire database here. More sophisticated strategies could be used to limit the ranking to be performed on a subset of the database.} Unless otherwise noted, we implicitly assume dependency on $x_q$ and $S$ in our notation, and omit them for brevity when applicable. We will also use the shorthand $A_q(i) = A(x_q, x_i)$.

Ranking-based metrics usually measure some form of agreement between the ranking and ground truth relevance values. Intuitively, given query $x_q$, ranking metrics are optimized when instances having high relevance to $x_q$ are placed high in the ranking, and low otherwise.

**Precision** is applied in the case of binary relevance, and measures the proportion of retrieved neighbors ($x_i$ for which $A_q(i) = 1$) at a certain cutoff $k$ in the ranking:

\[
\text{Prec}@k(R) = \frac{1}{k} \sum_{j=1}^{k} A_q(R_j). \tag{2.8}
\]

**Recall** is another widely used metric for binary relevance. Like Precision, it is also evaluated at a cutoff $k$, but measures the proportion of retrieved neighbors out of all the neighbors in the database:

\[
\text{Rec}@k(R) = \frac{1}{N^+} \sum_{j=1}^{k} A_q(R_j), \tag{2.9}
\]

where $N^+ = |\{x_i \in S | A_q(i) = 1\}|$ is the total count of neighbors.

**Recall Rate** is closely related to Recall, but involves a set of queries $Q$. It is defined as the ratio of queries for which there is at least one neighbor in the top $k$ retrievals. Mathematically, for each query $x_q \in Q$, let its corresponding ranking be $R_q^{(q)}$, and
Recall Rate can be written as
\[
R@k(\{R^{(q)}|x_q \in Q\}) = \frac{1}{|Q|} \mathbf{1}\left[ \sum_{j=1}^{k} A_q(R_j^{(q)}) > 0 \right],
\]
(2.10)
where \( \mathbf{1}[] \) denotes the binary indicator.

**Average Precision (AP)** takes the average of Precision values over all positions where there is a neighbor:

\[
AP(R) = \frac{1}{N^+} \sum_{k=1}^{|S|} A_q(R_k) \text{Prec@k}(R).
\]
(2.11)
An alternative definition of AP, that is also useful for our later analysis, is the area under the Precision-Recall curve. Specifically, if we view Precision and Recall as continuous values, and view Precision as a function of Recall, then AP corresponds to the following integral:

\[
AP = \int_0^1 \text{Prec}(\text{Rec}) \, d\text{Rec}.
\]
(2.12)

**Discounted Cumulative Gain (DCG)** is used for graded relevance. The idea is to sum up the contributions (gain) from the relevance values in the ranked list, while weighting them (discount) according to the rank:

\[
\text{DCG}(R) = \sum_{k=1}^{|S|} G(A_q(R_k)) D(k),
\]
(2.13)
where
\[
G(a) = 2^a - 1, \quad D(k) = \frac{1}{\log_2(k + 1)},
\]
(2.14)
are called the gain and discount functions, respectively. Other gain and discount functions can also be used (e.g. linear discount instead of logarithmic), but the com-
The combination shown above is used the most.

**Normalized Discounted Cumulative Gain (NDCG)** is the normalized variant of DCG. It divides DCG by its maximum possible value, ensuring a range of \([0, 1]\):

\[
\text{NDCG}(R) = \frac{\text{DCG}(R)}{\max_{R'} \text{DCG}(R')}. \tag{2.15}
\]

Note that, to achieve maximum DCG, retrieved items need to be sorted in descending order according to relevance to the query. Therefore given a query and a database, the maximum DCG is a constant and can be precomputed if necessary. Also, this means that the *optimization* of NDCG is independent of the normalizing factor, which boils down to optimizing DCG.
Chapter 3

Related Work

In this chapter, we review related work and prior art.

3.1 Vector Embedding using Neural Networks

Vector embeddings play a central role in many application areas of pattern recognition, since they allow encoding data to fixed-length representations. In addition, performing the embedding with a small number of dimensions can lead to savings in computation and space. One classical example, the Bag-of-Words (BOW) model [110], is first studied in information retrieval and natural language processing, which represents documents by histograms of word counts. BOW models were also widely adopted in computer vision [39, 93, 159]; in fact, the second-best entry in the ImageNet Large-Scale Visual Recognition Challenge (ILSVRC) in 2012, behind the now-famous AlexNet [85], was based on BOW-type features [133].

With the advent of deep learning, the vector embedding problem has been increasingly studied with deep neural networks. The classical example of vector embedding using neural networks is Siamese networks [18], originally designed for the computer vision problem of handwritten signature verification, and later used in face verification [33]. After the success of AlexNet [85], it was found that off-the-shelf activation features from pretrained deep convolutional neural networks can directly give competitive baselines for image retrieval [8] and general visual recognition [140]. However, it is often necessary to learn the embeddings so as to optimize for certain tasks. For
example, in the complicated image captioning task, vector embeddings can be used to aid nearest neighbor search \[79\]. Another example is learning local image descriptors for sparse feature matching \[48, 112\].

Outside of computer vision, vector embeddings also find important applications in areas such as natural language processing, where Word2Vec \[114\] and Skip-Thought Vectors \[83\] have become widely used tools for encoding words and sentences. Shared vector embeddings are often learned across different modalities to enable cross-modal retrieval and recognition, examples include Cross-Modal Transfer \[144\], DeViSE \[51\], Cross-Modal Scene Networks \[28\], and Audio-Visual Embedding \[7\].

**Connection to Metric Learning**

Broadly speaking, the vector embedding problem is an instance of metric learning \[86\], which is concerned with learning distance functions from data. The distance functions can be parameterized in various ways, for example, a large body of the metric learning literature focuses on learning Mahalanobis distances \[98, 111, 168, 174\], which essentially learn a linear transformation to the input data. On the other hand, recently much attention has been paid to learning vector embeddings with deep neural networks, \textit{e.g.} \[94, 123, 146, 147, 155, 171\]. When used with standard distance metrics, \textit{e.g.} Euclidean distance, the learned vector embeddings effectively induce new distance metrics. In this light, the specific problems of supervised hashing and learning local feature descriptors using neural networks, which we study in this thesis, are instances of metric learning.

### 3.2 Learning to Rank

Nearest neighbor retrieval has often been studied in the Information Retrieval (IR) literature, motivated by practical applications such as web search engines \[38\] or
recommendation systems [14]. For consistency with the IR literature, here we use a terminology inspired by search engines, and assume a set of queries (e.g. text input from the user) and a set of documents (e.g. web pages) in the database. An IR system generally includes a scoring function, such that given a query, candidate documents in the database are assigned scores to indicate the relevance to the query. By sorting the scores, a ranked list of documents is produced and returned to the user. The quality of such a ranked list is commonly evaluated using ranking-based metrics [110], such as Precision, Recall, Average Precision (AP), and Normalized Discounted Cumulative Gain (NDCG). In fact, the optimization of such metrics is also studied extensively in the “Learning to Rank” literature [105], which is a subfield of IR.

Learning to Rank (LTR) is concerned with learning the parameters of the scoring function used in IR systems, such that the overall retrieval performance is optimized. Although the ultimate goal usually is to optimize entire ranked lists, in practice a wide spectrum of approaches exist. In [105], they are categorized into pointwise, pairwise, and listwise approaches.

**Pointwise and Pairwise Methods**

The simplest approach to LTR is *pointwise*, where the input is the feature representation of a document, and the desired output is its relevance score to the query. Documents with low relevance to the query should receive low scores, and vice versa. An example of pointwise LTR is Ordinal Regression [35, 67].

In *pairwise* approaches, the input is a pair of documents in some feature representation, and the model is tasked with deciding which document is more relevant to the query. In this case, the model being learned can still be in the form of a scoring function, except that the absolute relevance scores are not as important as their relative comparison. Notable examples of pairwise LTR approaches include RankNet
Listwise Methods

In listwise LTR approaches, the input is a list of documents, and the model needs to optimize the total ordering of such documents according to relevance. Two of the most important listwise evaluation metrics are AP and NDCG, which are found to be well-correlated with user satisfaction in various IR applications. Therefore, many listwise LTR methods are motivated from optimizing such metrics.

The main difficulty in optimizing listwise IR metrics is that the sorting operation, required in producing ranked lists, is non-differentiable. Therefore, gradient-based methods cannot be directly applied, and this gives rise to a plethora of alternative strategies. For instance, surrogates of AP \cite{179} and NDCG \cite{29} can be optimized in the structural SVM framework \cite{154}, and bound optimization algorithms exist for NDCG \cite{156}. Mohapatra et al. \cite{116, 117} tackle the optimization of ranking-based loss functions using a quicksort-flavored algorithm. Additionally, there are gradient-based methods based on smoothing or approximating IR metrics \cite{20, 30, 90, 150}.

Among gradient-based methods, the direct loss minimization framework proposed by Hazan et al. \cite{58} is notable. Although not directly motivated by learning to rank, it gives a principled way to asymptotically derive gradients for a wide family of loss functions, including the ranking-based AP and NDCG. The framework is extended to deep neural networks by Song et al. \cite{148}, and is recently used by Triantafillou et al. \cite{152} to tackle the problem of few-shot learning.

Connection to Metric Learning

The scoring function in learning to rank can also be induced from a distance metric: a low distance between the query and document indicates high relevance, and vice versa. Therefore, learning to rank can be transformed to a metric learning problem.
or put differently, metric learning can use loss functions from learning to rank.

For learning Mahalanobis distances with shallow models, learning to rank formulations are discussed in [32, 98, 111], to name a few. In the context of learning deep embeddings, Siamese networks [18] can be viewed as a pointwise approach, in the sense that an instance pair can be viewed as consisting of a query and a document, and the loss function encourages the distances between the query and relevant documents to be low, and high otherwise. Similarly, triplet-based loss functions used in recent deep metric learning approaches [89, 147, 171, 178] can be viewed as pairwise ranking losses, in that the query is constrained to have lower distance to a relevant document than the other irrelevant document.

We take inspiration from the large body of work in Information Retrieval and Learning to Rank, and propose novel ranking-based formulations to tackle the vector embedding problem for nearest neighbor retrieval. Our primary application context is computer vision, and the loss functions that we use belong to the listwise category. In IR terminology, our ranking models are \textit{query-dependent}, since each time a new query is encountered, the document vectors are compared to a different vector, and in general a different ranking is produced. On the other hand, the document embeddings are \textit{query-agnostic}, since queries and documents are independently embedded to vector representations using neural networks. In fact, in the computer vision problems that we study, the notions of query and document are often interchangeable; for example, in image retrieval, an image could either be a query or a document in the database.

### 3.3 Hashing for Nearest Neighbor Retrieval

Hashing is a widely used approach for practical nearest neighbor retrieval [162], thanks to the efficiency of evaluating Hamming distances using bitwise operations, as well as
the low memory and storage footprint. Hashing methods can be grouped into data-independent and data-dependent techniques. Data-independent techniques, such as the well-known Locality Sensitive Hashing (LSH) [54, 88], can preserve certain distance metrics with theoretical guarantees. On the other hand, it is widely observed empirically, and theoretically demonstrated [3], that data-dependent hashing methods can outperform data-independent ones. Below, we review two different approaches to data-dependent hashing.

**Quantization Methods**

Quantization methods generally optimize an objective involving a reconstruction error, and do not assume the availability of supervision. Noteworthy work includes PCA-inspired methods such as Iterative Quantization [55] and K-means Hashing [59], where principal components are taken as the hash functions. If “groups” that are suitable for clustering exist within the data, then further refining the principal components has proven to be beneficial.

Unsupervised quantization can also be approached as a special case of generative modeling. Semantic Hashing [137] is one early example of such algorithms based on the autoencoding principle, using stacked Restricted Boltzmann Machines (RBMs). Binary Autoencoders [27] construct autoencoders with a binary latent layer, so that relaxation errors could be avoided. More recently, Stochastic Generative Hashing [41] learns a generative hashing model based on the minimum description length principle, which optimizes a variational objective.

**Supervised Hashing**

The supervised hashing problem is also known as affinity-based hashing [87, 106, 127], where supervision is often given in the form of pairwise affinity/similarity labels.

We first discuss shallow supervised hashing methods that are not based on deep
learning. In the pioneering work Binary Reconstructive Embeddings \[87\], Kulis and Darell propose a kernel-based solution where the goal is to construct hash functions by minimizing an empirical loss between the input and Hamming space distances via a coordinate descent algorithm. Supervised Hashing with Kernels \[106\] also proposes a kernel-based solution, and learns the kernel function weights by minimizing an objective function based on binary code inner products. Minimal Loss Hashing \[120\] and Hamming Distance Metric Learning \[121\] minimize a hinge loss function in online fashion, motivated by structural SVMs \[154\]. Another notable line of work includes Spectral Hashing \[169\] and Self-Taught Hashing \[181\], where the similarity of the instances is preserved by solving a graph Laplacian problem.

Recently, following the success of deep neural networks in related tasks such as image classification, deep supervised hashing methods have gained significant prominence. Xia \textit{et al.} \[173\] and Lai \textit{et al.} \[91\] are among the first to consider jointly learning the image features and the hash mapping. Lin \textit{et al.} \[104\] finetune a network originally trained for classification with a softmax cross-entropy loss, and then threshold the activations in the penultimate layer as the hash codes. In VDSH \[186\], a deep neural network consisting only of fully-connected layers is proposed, and similar to \[27\], auxiliary variables are used to circumvent the vanishing gradient problem during training. Differently, DPSH \[97\] and DTSH \[166\] directly optimize likelihood functions similar to the cross-entropy loss for classification, defined pairs or triplets of training instances. In HashNet \[25\], the mismatch between the discrete nature of hashing and the use of gradient-based continuous optimization is mitigated by using the continuation method \[2\].

Finally, a notable family of approaches is \textit{two-step} methods, which decompose the learning into two steps: binary code inference and hash function learning. The binary code inference step yields binary hash codes for training instances according
to supervision, and the hash function learning step uses them as target vectors. Examples include Fast Hashing with Decision Trees [102, 103], Supervised Hashing with Error-Correcting Codes [23], Fast Training of Triplet Embedding Networks [188], and Binary Matrix Pursuit [22].

In terms of optimization, the discrete nature of hashing usually results in NP-hard problems. Relaxation-based methods relax the discrete constraints and entirely rely on continuous optimization, e.g. [87, 106] and nearly all the deep hashing methods. Other methods alternate between optimizing the binary codes and continuous model parameters [101, 121, 127], including a deep hashing method [45]. In two-step methods, the binary code inference step typically solves a binary quadratic program, and the hash mapping learning step is usually continuous. Our proposed supervised hashing solutions [62, 63, 64] belong to relaxation methods.

**Learning to Rank in Hashing**

Supervised hashing can be cast as a distance metric learning problem [121], in that the learned metric space is a Hamming space. As we discussed earlier, metric learning can be solved using learning to rank formulations. In the hashing literature, different strategies have been proposed to handle the difficulties in optimizing listwise ranking metrics, mostly using shallow models. For example, [164] decomposes listwise supervision into local triplets, [101, 177] use structural SVMs to optimize surrogate losses, [145] maximizes precision at the top, and [165, 187] optimize NDCG surrogates.

### 3.4 Learning Local Image Descriptors

Computer vision can be modeled as a hierarchy consisting of low-level, mid-level, and high-level tasks. Of these, low-level computer vision has the longest history, and is concerned with analyzing intrinsic and primitive properties of images and videos,
such as finding edges or corners, establishing correspondences between images, and estimating optical flow. A very common low-level computer vision task is image matching, which serves as the foundation for more complicated tasks such as camera localization, 3D reconstruction, and SLAM. An image matching pipeline typically works by identifying sparse “features” or “interest points” in each image, and finding correspondences between them. This involves the following stages: interest point detection, feature extraction, and feature matching. Afterwards, the correspondences are often filtered by robust estimation algorithms such as RANSAC [49] to remove outliers and improve the accuracy of estimated geometric transformations.

The most prominent example in low-level computer vision is the SIFT pipeline [108], which is widely used for image matching. It includes carefully handcrafted solutions for interest point detection, feature extraction, and feature matching. Almost 20 years after its initial introduction, SIFT is still being used successfully in complicated vision tasks such as 3D reconstruction [139].

**Learning the Low-Level Computer Vision Pipeline**

Parallel with the long history of handcrafted computer vision pipelines, numerous researchers have attempted to replace handcrafted components with learned counterparts. There exist many formulations for learning different components in local feature based pipelines. For example, interest point detectors are learned in [96, 138, 161, 183], Georgakis *et al.* [53] learn keypoint detectors and feature descriptors jointly for 3D depth maps, Yi *et al.* learn three components separately in a feature matching pipeline named LIFT [176], and Brachmann *et al.* [16, 17] approximately learn a camera localization pipeline end-to-end. In this thesis, we instead focus on learning a key component in the pipeline: local feature descriptors.
Evaluating Local Feature Descriptors

A question that needs to be answered prior to the learning of local feature descriptors is how they should be evaluated, as evaluation metrics often drive the design choices in learning. Ideally, local features should be evaluated in terms of final task performance. For example, Mikolajczyk and Schmid [113] use precision and recall derived from image matching as evaluation metrics, and Schonberger et al. [139] use a large-scale benchmark where the task is 3D reconstruction. However, in complex vision pipelines, final task performance can be affected by individual components. For example, Balntas and Lenc et al. [9] observe that without controlling for components such as interest point detection in image-based benchmarks, different conclusions can be drawn when comparing the relative performance of feature descriptors. Therefore, for the sole purpose of evaluating learned local feature descriptors, patch-based evaluation is preferred.

Patch-based benchmarks provide unambiguous evaluation for local feature descriptors. First, the patch verification task is proposed in [170], formulated as binary classification on the relationship (same/different) between patch pairs. However, it has been observed that good performance on the patch verification task does not guarantee good matching performance in practice. Two more recent benchmarks, RomePatches [124] and HPatches [9], both consider the patch retrieval task, which simulates nearest neighbor matching by forming a retrieval task defined on local image patches, and is shown to be more realistic and challenging compared to patch verification. Average Precision is adopted in both benchmarks as the evaluation metric. HPatches also includes the image matching task, which is even more realistic and challenging compared to patch verification and retrieval.
Learning Local Feature Descriptors

For learning local feature descriptors, early works are based on shallow models. For example, handcrafted architectures are used by [153, 170]. Simonyan et al. [143] also explore the use of convex optimization with shallow models. Later approaches mainly use deep neural networks, where the optimization problems are non-convex. First, PhilippNet [48] learns deep descriptors by creating a surrogate classification task using “pseudo-classes”. Later works take metric learning style approaches, such as DeepDesc [142] which employs Siamese networks, as well as MatchNet [57] and DeepCompare [180], which both learn nonlinear distance metrics for matching. An interesting exception is [124], where a novel model architecture called Convolutional Kernel Networks is adopted. A series of recent works have considered more advanced model architectures and triplet-based deep metric learning formulations to learn feature descriptors, including UCN [34], TFeat [10], GLoss [89], L2Net [151], HardNet [115], and GOR [185].

Instead of optimizing triplet-based surrogate losses, in this thesis we employ our listwise learning to rank framework to directly optimize the performance of the matching stage, which, as we show later, can be measured by Average Precision. Although end-to-end optimization of the entire pipeline is attractive, it is unfortunately highly difficult and task-dependent. By focusing on the two task-independent stages (descriptor extraction and matching), our solution is general-purpose and can be potentially integrated into larger optimization pipelines.
Chapter 4

Hashing with Mutual Information

In this chapter, we describe our first contribution to the supervised hashing problem, where the goal is to learn compact binary embeddings for fast nearest neighbor retrieval.

Although many different formulations exist, all supervised hashing formulations essentially constrain the learned Hamming distance to agree with the given supervision. In this chapter, we exclusively consider the binary relevance setup, where pairs of objects are annotated with binary labels indicating their pairwise relationships as either “similar” or “dissimilar.” Given an anchor example \( \hat{x} \in X \), we can construct a set of instances that are labeled as similar to it, or simply, its neighbors, denoted \( S^+_\hat{x} \), and similarly the non-neighbor set \( S^-\hat{x} \). We also refer to the pair \((S^+_\hat{x}, S^-\hat{x})\) as the neighborhood structure of \( \hat{x} \).

In learning a hash mapping \( \Phi \), the goal is to preserve the neighborhood structure, so that neighbors have smaller distances to the anchor than non-neighbors. In other words, we would like to satisfy the following constraint,

\[
d_\Phi(\hat{x}, x_p) < d_\Phi(\hat{x}, x_n), \forall x_p \in S^+_\hat{x}, \forall x_n \in S^-\hat{x},
\]

(4.1)
in which case the neighborhood structure can be recovered by simply thresholding the learned \( d_\Phi \). However, this constraint is difficult to optimize directly. In the hashing literature, a common proxy strategy is affinity matching, where constraints
are enforced on the absolute values of the learned Hamming distance:

\[ d_\Phi(\hat{x}, x_p) < t_1, \quad d_\Phi(\hat{x}, x_n) > t_2, \quad \forall x_p \in S^+_\hat{x}, \forall x_n \in S^-_{\hat{x}}, \]

(4.2)

where \( t_1 \leq t_2 \) are threshold parameters. Alternatively, loss functions can also be defined in terms of relative distance comparisons within triplets of examples, where a neighbor is encouraged to have smaller distance to the anchor than a non-neighbor:

\[ d_\Phi(\hat{x}, x_p) + \eta \leq d_\Phi(\hat{x}, x_n), \quad \forall x_p \in S^+_\hat{x}, \forall x_n \in S^-_{\hat{x}}, \]

(4.3)

where \( \eta > 0 \) is a margin parameter. This can be termed local ranking.

Loss functions used in affinity matching and local ranking methods typically have simple forms so as to make the resulting optimization problems easily solvable. However, the downside is that these loss functions are only indirectly related to retrieval performance, and in order to optimize overall retrieval performance, it is often necessary to introduce additional regularization terms, or parameters such as margins, thresholds, and scaling factors. These formulations are also inflexible, since the same threshold or margin parameters are applied for all anchors \( \hat{x} \). Furthermore, it is often observed in practice that these parameters are nontrivial to tune.

We now discuss a different formulation for learning the Hamming embedding \( \Phi \). We propose a novel formulation based on the idea of minimizing neighborhood ambiguity, which is more directly related to the quality of nearest neighbor retrieval. We say that \( \Phi \) induces neighborhood ambiguity if the mapped image of some \( x_n \in S^-_{\hat{x}} \) is closer to that of \( \hat{x} \) than some \( x_p \in S^+_\hat{x} \) in the Hamming space, i.e., \( d_\Phi(\hat{x}, x_n) < d_\Phi(\hat{x}, x_p) \).

When this happens, it is no longer possible to exactly recover the neighborhood structure by thresholding \( d_\Phi \). Consequently, when \( \Phi \) is used to perform retrieval, the retrieved “nearest neighbors” of \( \hat{x} \) would be contaminated by non-neighbors. Therefore, a high-quality embedding should minimize neighborhood ambiguity.
Figure 4.1: Overview of the proposed hashing method, MIHash. We use a deep neural network to compute $b$-bit binary codes for: a (1) query image $\hat{x}$, (2) its neighbors in $S^+_{\hat{x}}$ (blue), and (3) its non-neighbors in $S^-_{\hat{x}}$ (orange). Computing hamming distances between the binary code of the query and the binary codes of neighbors and non-neighbors yields two distributions of Hamming distances. The information-theoretic quantity, Mutual Information, can capture the separability between these two distributions, and is used as our learning objective. In this example, hash mapping $\Phi_1$ is of higher quality than $\Phi_2$.

To concretely formulate the idea, we define two random variables:

$$D_{\hat{x}, \Phi} : \mathcal{X} \rightarrow \{0, 1, \ldots, b\}, x \mapsto d_\Phi(x, \hat{x}),$$

$$C_{\hat{x}} : \mathcal{X} \rightarrow \{0, 1\}, x \mapsto 1[x \in S^+_{\hat{x}}].$$

Then, we naturally have two conditional distributions of the Hamming distance: $P(D_{\hat{x}, \Phi}|C_{\hat{x}} = 1)$ and $P(D_{\hat{x}, \Phi}|C_{\hat{x}} = 0)$. Note that minimizing neighborhood ambiguity amounts to minimizing the overlap between these distributions. Please see Figure 4.1 for an illustration.

We use the **mutual information** between random variables $D_{\hat{x}, \Phi}$ and $C_{\hat{x}}$ to capture the amount of overlap between conditional Hamming distance distributions. The
mutual information is defined as

\[ I(\hat{D}, \hat{C}; C) = H(C) - H(C|\hat{D}, \Phi) \]

\[ = H(D|\hat{C}; \Phi) - H(D, \Phi|\hat{C}) \]

where \( H \) denotes (conditional) entropy. In the following, for brevity we will drop subscripts \( \Phi \) and \( \hat{x} \), and denote the two conditional distributions and the marginal \( P(D, C) \) as \( p^+_D, p^-_D \), and \( p_D \), respectively.

By definition, \( I(D; C) \) measures the decrease in uncertainty in the neighborhood information \( C \) when observing the Hamming distances \( D \). If \( I(D; C) \) attains a high value, which means \( C \) can be determined with low uncertainty by observing \( D \), then \( \Phi \) must have achieved good separation (low overlap) between \( p^+_D \) and \( p^-_D \). \( I \) is maximized when there is no overlap, and minimized when \( p^+_D \) and \( p^-_D \) are exactly identical. As \( H(C) \) is typically constant, maximizing mutual information corresponds to minimizing the conditional entropy \( H(C|D, \Phi) \). Mutual information is also related to the Kullback-Leibler divergence measure \( D_{KL} \), specifically as

\[ I(D; C) = \mathbb{E}_D \left[ D_{KL}(P(C|D)||P(C)) \right] \]

corresponding to the expected divergence between the distributions \( P(C|D) \) and \( P(C) \). Intuitively, if \( D \) were to be informative, these two Bernoulli distributions should differ. Indeed, maximizing the \( D_{KL} \) divergence maximizes the difference of the two distributions.

Next, for any hash mapping \( \Phi \), we can integrate \( I \) over the feature space to give a quality measure:

\[ O(\Phi) = \int_X I(D, \Phi; C) p(\hat{x}) d\hat{x} \]

An appealing property of this mutual information objective is that it is parameter-
free: the objective encourages distributions $p_+^D$ and $p_-^D$ to be separated, but does not include parameters dictating the distance threshold at which separation occurs, or the absolute amount of separation. The absence of such fixed parameters also increases flexibility, since the separation could occur at different distance thresholds depending on the anchor $\hat{x}$.

4.1 Optimizing Mutual Information

Having shown that mutual information is a suitable measure of hashing quality, we consider its use as a learning objective.

Clearly, optimizing $O(\Phi)$, as defined in (4.9), is intractable. As is usually the case in supervised learning, we optimize the parameters of $\Phi$ over a finite training set $\mathcal{T}$ of i.i.d. samples from $p(\hat{x})$. Our learning problem is then formulated as

$$\max_{\Phi} \frac{1}{|\mathcal{T}|} \sum_{x \in \mathcal{T}} I(D_x, \Phi; C_x).$$

(4.10)

It is worth noting that for each $x \in \mathcal{T}$, elements of $S^+_x$ and $S^-_x$ are now restricted to be within $\mathcal{T}$. Inspired by recent advances in stochastic optimization, we will use stochastic gradient descent to solve this above problem.

We start by deriving the gradients of $I$ with respect to the output of the hash mapping, $\Phi(x)$. First, note that with $b$-bit Hamming distances, the discrete distributions $p_+^D$ and $p_-^D$ can be modeled using normalized histograms over $\{0, \ldots, b\}$. Specifically, let $p_{D,l}$ be the $l$-th element of $p_+^D$, which is estimated by performing hard assignments on Hamming distances into histogram bins:

$$p_{D,l} = \frac{1}{|S^+_\hat{x}|} \sum_{x \in S^+_\hat{x}} 1[d_\Phi(\hat{x}, x) = l], \quad l = 0, \ldots, b,$$

(4.11)

where $1[\cdot]$ denotes the binary indicator.
The mutual information $\mathcal{I}$ is continuously differentiable, and using the chain rule we can write

$$\frac{\partial \mathcal{I}}{\partial \Phi(x)} = \sum_{l=0}^{b} \left[ \frac{\partial \mathcal{I}}{\partial p^+_D,l} \frac{\partial p^+_D,l}{\partial \Phi(x)} + \frac{\partial \mathcal{I}}{\partial p^-_D,l} \frac{\partial p^-_D,l}{\partial \Phi(x)} \right].$$

(4.12)

Due to symmetry, we next only focus on terms involving $p^+_D$. Let $p^+$ and $p^-$ be shorthands for the priors $P(C = 1)$ and $P(C = 0)$. For $l = 0, \ldots, b$, we have

$$\frac{\partial \mathcal{I}}{\partial p^+_D,l} = -\frac{\partial H(D|C)}{\partial p^+_D,l} + \frac{\partial H(D)}{\partial p^+_D,l}$$

$$= p^+(\log p^+_D,l + 1) - (\log p_D,l + 1) \frac{\partial p_D,l}{\partial p^+_D,l}$$

(4.13)

$$= p^+(\log p^+_D,l - \log p_D,l).$$

(4.14)

(4.15)

Note that for (4.15), we used the fact that

$$p_D,l = p^+p^+_D,l + p^-p^-_D,l.$$  

(4.16)

### 4.1.1 Continuous Relaxation

To complete the chain rule, we need to further derive the term $\partial p^+_D,l/\partial \Phi(x)$ in (4.12). However, the hash mapping $\Phi$ is discrete by nature, precluding the use of continuous optimization. While it is possible to maintain such constraints and resort to discrete optimization, the resulting optimization problems are NP-hard.

Instead, in order to apply gradient-based continuous optimization, we take the relaxation approach to sidestep the NP-hard problems. Correspondingly, we need to perform a continuously differentiable relaxation to $\Phi$. Recall from (2.3) that each element in $\Phi$ is obtained by thresholding neural network activations with the sign function. We relax $\Phi$ into a real-valued vector $\hat{\Phi}$ by adopting a standard technique in hashing, e.g. in [24, 106], where the discontinuous sign function is approximated...
with the tanh function,

\[
\hat{\Phi}(x) = (\hat{\phi}_1(x), \ldots, \hat{\phi}_b(x)) \tag{4.17}
\]

\[
= (\tanh(\gamma f_1(x; w)), \ldots, \tanh(\gamma f_b(x; w))). \tag{4.18}
\]

We include a tuning parameter \( \gamma \), used to control the “steepness” of the approximation. Typically, we choose \( \gamma \geq 1 \) so as to reduce the error introduced by the continuous relaxation, and determine the best value through validation. Although with \( \gamma \to \infty \) the approximation approaches the sign function, in practice a large \( \gamma \) can make gradients vanish due to the saturation of the tanh function. Other alternative relaxation strategies include using a quantization error term \([97, 166]\) and applying the continuation method \([25]\).

With the continuous relaxation in place, we move on to the partial differentiation of \( p^+_{\hat{x}} \) and \( p^-_{\hat{x}} \) with respect to \( \hat{\Phi}(x) \). As mentioned before, these discrete distributions can be estimated via histogram binning \((4.11)\); however, histogram binning is a non-differentiable operation, due to the use of the binary indicator function. In the following, we describe a differentiable approximation to the discrete histogram binning process, thereby enabling end-to-end backpropagation.

### 4.1.2 End-to-End Optimization

Without the continuous relaxation, \((4.11)\) performs histogram binning by assigning \( d_{\Phi}(\hat{x}, x) \), which is an integer, into a specific bin. With the continuous relaxation developed above, we note that \( d_{\Phi} \) in is no longer integer-valued, but is also continuously relaxed into

\[
\hat{d}_{\Phi}(\hat{x}, x) = \frac{b - \hat{\Phi}(\hat{x})^\top \hat{\Phi}(x)}{2}. \tag{4.19}
\]

When \( d_{\Phi} \) is relaxed into \( \hat{d}_{\Phi} \), we need to replace the hard assignment with soft
assignment. The key is to approximate the binary indicator $1[\cdot]$ with a differentiable function. For this purpose, we employ a technique from [155]. Specifically, the binary indicator is replaced by a triangular kernel function $\delta$ with slope parameter $\Delta$, centered on the histogram bin center, which linearly interpolates the real-valued $\hat{d}_\Phi(\hat{x}, x)$ into the $l$-th bin:

$$\delta(d, l) = \max \left\{ 0, 1 - \frac{|d - l|}{\Delta} \right\}.$$  

(4.20)

It is easy to see that this triangular approximation approaches the original binary indicator as $\Delta \to 0$. Also, this soft assignment admits simple subgradients:

$$\delta'_l(d) = \frac{\partial \delta(d, l)}{\partial d} = \begin{cases} \frac{1}{\Delta}, & d \in [l - \Delta, l], \\ -\frac{1}{\Delta}, & d \in [l, l + \Delta], \\ 0, & \text{otherwise}. \end{cases}$$  

(4.21)

We are now ready to tackle the term $\partial p^+_D,l/\partial \hat{\Phi}(x)$. From the definition of $p^+_D,l$ in (4.11), we have, for $x = \hat{x}$:

$$\frac{\partial p^+_D,l}{\partial \hat{\Phi}(\hat{x})} = \frac{1}{|S^+_x|} \sum_{x \in S^+_x} \frac{\partial \delta(\hat{d}_\Phi(\hat{x}, x), l)}{\partial \hat{\Phi}(\hat{x})} \frac{\partial \hat{d}_\Phi(\hat{x}, x)}{\partial \hat{\Phi}(\hat{x})}$$  

(4.22)

$$= \frac{1}{|S^+_x|} \sum_{x \in S^+_x} \frac{\partial \delta(\hat{d}_\Phi(\hat{x}, x), l)}{\partial \hat{d}_\Phi(\hat{x}, x)} \frac{\partial \hat{d}_\Phi(\hat{x}, x)}{\partial \hat{\Phi}(\hat{x})}$$  

(4.23)

$$= -\frac{1}{2|S^+_x|} \sum_{x \in S^+_x} \delta'_l(\hat{d}_\Phi(\hat{x}, x)) \hat{\Phi}(x).$$  

(4.24)

For the last step, we used the definition of $\hat{d}_\Phi$ in (4.19). Next, for $x \neq \hat{x}$:

$$\frac{\partial p^+_D,l}{\partial \hat{\Phi}(x)} = \frac{1}{|S^+_x|} \sum_{x \in S^+_x} \frac{\partial \delta(\hat{d}_\Phi(\hat{x}, x), l)}{\partial \hat{d}_\Phi(\hat{x}, x)} \frac{\partial \hat{d}_\Phi(\hat{x}, x)}{\partial \hat{\Phi}(x)}$$  

(4.25)

$$= -\frac{1}{2|S^+_x|} \sum_{x \in S^+_x} \delta'_l(\hat{d}_\Phi(\hat{x}, x)) \hat{\Phi}(\hat{x}).$$  

(4.26)
Lastly, to back-propagate gradients to \( \Phi \)'s inputs, and ultimately to the parameters of the underlying deep neural network, we only need to further differentiate the tanh approximation employed in \( \hat{\Phi} \) (4.18). The derivative of the tanh function has a closed form expression, and is omitted here.

4.1.3 Efficient Minibatch Backpropagation

So far, in information retrieval terminology, our derivations of mutual information and its gradients have assumed a single query example \( \hat{x} \), and a fixed database. However, the optimization objective in (4.10) is the average of mutual information values over all queries in a finite training set \( \mathcal{T} \). We now address this mismatch.

We face two challenges when working with a (potentially large) training set \( \mathcal{T} \). First, we need to perform the optimization in the stochastic/minibatch setting, since deep neural networks are typically trained by minibatch stochastic gradient descent (SGD), where it can be infeasible to access the entire database all at once. The second challenge is that, differently from traditional information retrieval, in many computer vision tasks (e.g. image retrieval), there is usually no clear split of a given training set into a set of queries and a database. This is due to the symmetry that an image can either be a query or a database item. Consequently, even if we were to create such a split, it can be arbitrary and does not fully utilize available supervision.

Here, we describe a way to efficiently utilize all the available supervision during minibatch SGD training, simultaneously addressing both challenges. Our reasoning is that, within a minibatch with \( M \) examples, a retrieval problem can be defined by retrieving one example (the query) against the other \( M - 1 \) examples (the database). Further, considering the symmetry mentioned above, retrieval can be repeated \( M \) times, each time using a different example as the query. Then, the overall objective value for the minibatch is the average over the \( M \) individual retrieval problems. This
way, the available supervision in the minibatch is utilized maximally.

Now consider a minibatch of size $M$, $\mathcal{B} = \{x_1, \ldots, x_M\}$. Since we only operate within the minibatch, for each example $x_i \in \mathcal{B}$, $1 \leq i \leq M$, when used as the query, its neighborhood structure is now defined within $\mathcal{B}$: we take $S^+_i = S^+_{x_i} \cup \mathcal{B}$, and $S^-_i = S^-_{x_i} \cup \mathcal{B}$. Also, let $\mathcal{I}_i$ be a shorthand for $\mathcal{I}(D_{x_i}, \Phi, C_{x_i})$. We group the relaxed hash mapping output for the entire minibatch into the following $b \times M$ matrix,

$$\hat{\Phi} = \begin{bmatrix} \hat{\Phi}(x_1) & \hat{\Phi}(x_2) & \cdots & \hat{\Phi}(x_M) \end{bmatrix} \in \mathbb{R}^{b \times M}. \quad (4.27)$$

Similar to (4.12), we can write the Jacobian matrix of the minibatch objective $\mathcal{O}_B$ with respect to $\hat{\Phi}$ as

$$\frac{\partial \mathcal{O}_B}{\partial \hat{\Phi}} = \frac{1}{M} \sum_{i=1}^{M} \frac{\partial \mathcal{I}_i}{\partial \hat{\Phi}} \quad (4.28)$$

$$= \frac{1}{M} \left[ \sum_{i=1}^{M} \sum_{l=0}^{b} \frac{\partial \mathcal{I}_i}{\partial p^+_{i,l}} \frac{\partial p^+_{i,l}}{\partial \hat{\Phi}} + \sum_{i=1}^{M} \sum_{l=0}^{b} \frac{\partial \mathcal{I}_i}{\partial p^-_{i,l}} \frac{\partial p^-_{i,l}}{\partial \hat{\Phi}} \right], \quad (4.29)$$

where $p^+_{i,l}$ ($p^-_{i,l}$) denotes the $l$-th element of $p^D_+$ ($p^D_-$) when the query is $x_i$. We will leave the full details of the derivation to Appendix A but note here that the Jacobian matrix can be efficiently evaluated using only matrix additions and multiplications with time complexity $O(bM^2)$.

Recently, Triantafillou et al. [152] also propose a minibatch-based learning formulation that is inspired by information retrieval, which attempts to maximize the utilization of supervision by treating each example in the minibatch as a query. However, we note that [152] tackles the problem of few-shot learning by learning real-valued embeddings, and it uses very different optimization machinery to approximately optimize mean Average Precision in a structured prediction framework.
4.2 Experiments

4.2.1 Datasets and Setup

We conduct experiments on widely used image retrieval benchmarks: CIFAR-10, NUS-WIDE, 22K LabelMe and ImageNet100. Each dataset is split into a test set and retrieval set, and instances from the retrieval set are used in training.

We follow a standard information retrieval setup: at test time, queries from the test set are used to rank instances from the retrieval set using Hamming distances, and the performance metric is averaged over the queries.

- **CIFAR-10** is a dataset for image classification and retrieval, containing 60K images from 10 different categories. We follow the setup of [91, 97, 166, 188], and consider two experimental settings. In the first setting (S1), we sample 500 images per category, resulting in 5,000 training examples to learn the hash mapping. The test set contains 100 images per category (1000 in total). The remaining images are then used to populate the database. In the second setting (S2), we sample 1000 images per category to construct the test set (10,000 in total). The remaining items are both used to learn the hash mapping and populate the database. Two images are considered neighbors if they belong to the same class.

- **NUS-WIDE** is a dataset containing 269K images from Flickr. Each image can be associated with multiple labels, and the total number of labels is 81. Following the setup in [91, 97, 166, 188], we only consider images annotated with the 21 most frequent labels. In total, this corresponds to 195,834 images. There are also two distinct experimental settings: S1 and S2. For both cases, a test set is constructed by randomly sampling 100 images per label (2,100 images in total). To learn the hash mapping, 500 images per label are randomly sampled.
in S1 (10,500 in total). The remaining images are then used as the database. In
the second case, all the images excluding the test set are used in learning and
populating the database. Two images are considered as neighbors if they share
at least one label.

- **22K LabelMe** consists of 22K images, each represented with a 512-dimensional
  GIST descriptor [122]. Following [24, 57], we randomly partition the dataset
  into a database and a test set, consisting of 20K and 2K instances, respectively.
  A 5K subset of the database is used in learning the hash mapping. As this
dataset is unsupervised, we use the Euclidean distance between GIST features in
determining the neighborhood structure. Two examples that have a Euclidean
distance below the 5% distance percentile are considered neighbors.

- **ImageNet100** is a subset of ImageNet [44], containing 130K images from 100
classes. We follow the setup in [25], and randomly sample 100 images per class
for training. All images in the selected classes from the ILSVRC 2012 [133]
validation set are used as the test set. Two images are considered neighbors if
they belong to the same class.

As for performance metric, we use the standard mean Average Precision (mAP),
or its variants with fixed cutoff values. We compare MIHash against both classical
and recent state-of-the-art hashing methods. These methods include:

- Spectral Hashing (SH) [169],
- Iterative Quantization (ITQ) [55],
- Sequential Projection Learning for Hashing (SPLH) [163],
- Supervised Hashing with Kernels (SHK) [106],
• Fast Supervised Hashing with Decision Trees (FastHash) [102],
• Structured Hashing (StructHash) [101],
• Supervised Discrete Hashing (SDH) [141],
• Efficient Training of Very Deep Neural Networks (VDSH) [186],
• Deep Supervised Hashing with Pairwise Labels (DPSH) [97],
• Deep Supervised Hashing with Triplet Labels (DTSH) [166], and
• Hashing by Continuation (HashNet) [25].

We finetune deep Convolutional Neural Network models that are pretrained on the ImageNet dataset [44], by replacing the final softmax classification layer with a new fully-connected layer, whose activations are thresholded to produce the binary bits. The new fully-connected layer is randomly initialized. For CIFAR-10 and NUS-WIDE experiments, we finetune a VGG-F [31] architecture, as in [97, 166]. For ImageNet100 experiments, following the protocol of HashNet [25], we finetune the AlexNet [85] architecture and scale down The learning rates for pretrained layers are scaled down by a factor of 0.1 (for CIFAR-10) or 0.01 (for NUS-WIDE and ImageNet100). For shallow methods, we use the output of the penultimate layer (fc7) of both architectures as input features, which are 4096-dimensional. A special case is the 22K LabelMe dataset, where all methods learn shallow models on top of precomputed GIST descriptors; for gradient-based hashing methods, this corresponds to learning a single fully connected layer.

We use SGD with momentum 0.9 and weight decay of $5 \times 10^{-4}$ with constant learning rate, and reduce the learning rate periodically by a predetermined factor (0.5 in most cases), which is standard practice. During training, the minibatches are randomly sampled from the training set. $\gamma = 1$ is usually a good value.
4.2.2 Results

Table 4.1 gives results for CIFAR-10 and NUS-WIDE under the first experimental setting (S1) in the first half, where mean values of AP and AP@5K (AP evaluated on the top 5,000 retrievals) are reported, respectively. First, deep learning based hashing methods outperform most shallow hashing solutions. This is not surprising as the hash mapping is learned simultaneously with feature learning. Shallow solutions such as FastHash and SDH also perform competitively, especially in NUS-WIDE experiments. Our proposed method, MIHash, surpasses all competing methods in the majority of the experiments. For example, with 32-bit and 48-bit binary embeddings MIHash surpasses the nearest competitor, DTSH, by 3%-4% in CIFAR-10. For NUS-WIDE, MIHash achieves state-of-the-art performances in all experiments excluding with 12 bits.

The performance improvement of MIHash is much more significant in the second experimental setting (S2), where more training data is available. In this setting, we only compare deep learning based hashing methods, and AP and AP@50K are used as evaluation metrics on CIFAR-10 and NUS-WIDE, respectively. Results are given the lower half of Table 4.1. As can be observed, in both datasets, MIHash achieves state-of-the-art results in nearly all code lengths. For instance, MIHash consistently outperforms DTSH, the closest competitor, by a large margin.

Retrieval results for ImageNet100 are given in Table 4.2. In these experiments, we compare against DTSH, the overall best competing method in past experiments and another recently introduced deep learning based hashing method, HashNet [25]. Note that, HashNet also outperforms shallow methods such as [106] and [141] with deep features on ImageNet100, as reported in [25]. The evaluation metric is taken to be AP@1K for consistency with the setup in [25]. In this benchmark, MIHash significantly outperforms both DTSH and HashNet for all embedding sizes.
<table>
<thead>
<tr>
<th>Method</th>
<th>CIFAR-10</th>
<th>NUS-WIDE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>12 Bits</td>
<td>24 Bits</td>
</tr>
<tr>
<td>SH [169]</td>
<td>0.183</td>
<td>0.164</td>
</tr>
<tr>
<td>ITQ [55]</td>
<td>0.237</td>
<td>0.246</td>
</tr>
<tr>
<td>SPLH [163]</td>
<td>0.299</td>
<td>0.33</td>
</tr>
<tr>
<td>SHK [106]</td>
<td>0.488</td>
<td>0.539</td>
</tr>
<tr>
<td>SDH [141]</td>
<td>0.478</td>
<td>0.557</td>
</tr>
<tr>
<td>FastHash [102]</td>
<td>0.553</td>
<td>0.607</td>
</tr>
<tr>
<td>StructHash [101]</td>
<td>0.664</td>
<td>0.693</td>
</tr>
<tr>
<td>VDSH [186]</td>
<td>0.538</td>
<td>0.541</td>
</tr>
<tr>
<td>DPSH [97]</td>
<td>0.713</td>
<td>0.727</td>
</tr>
<tr>
<td>DTSH [166]</td>
<td>0.710</td>
<td>0.750</td>
</tr>
<tr>
<td>MIHash</td>
<td><strong>0.738</strong></td>
<td><strong>0.775</strong></td>
</tr>
<tr>
<td>DRSCH [184]</td>
<td>0.608</td>
<td>0.611</td>
</tr>
<tr>
<td>DPSH [97]</td>
<td>0.903</td>
<td>0.885</td>
</tr>
<tr>
<td>MIHash</td>
<td><strong>0.927</strong></td>
<td><strong>0.938</strong></td>
</tr>
</tbody>
</table>

Table 4.1: Results on CIFAR-10 and NUS-WIDE datasets, first setting. The underlying deep learning architecture is VGG-F [31]. MIHash outperforms competing methods on CIFAR-10, and shows improvements, especially with lengthier codes, on NUS-WIDE.
<table>
<thead>
<tr>
<th>Method</th>
<th>ImageNet100 (AP@1K)</th>
<th>16 Bits</th>
<th>32 Bits</th>
<th>48 Bits</th>
<th>64 Bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>DTSH [166]</td>
<td>0.458</td>
<td>0.566</td>
<td>0.611</td>
<td>0.644</td>
<td></td>
</tr>
<tr>
<td>HashNet [25]</td>
<td>0.506</td>
<td>0.630</td>
<td>0.663</td>
<td>0.683</td>
<td></td>
</tr>
<tr>
<td>MIHash</td>
<td><strong>0.569</strong></td>
<td><strong>0.661</strong></td>
<td><strong>0.685</strong></td>
<td><strong>0.694</strong></td>
<td></td>
</tr>
</tbody>
</table>

Table 4.2: mAP@1K values on ImageNet100 using AlexNet. MIHash outperforms HashNet, a state-of-the-art deep hashing formulation using continuation methods [25].

<table>
<thead>
<tr>
<th>Method</th>
<th>LabelMe (AP)</th>
<th>16 Bits</th>
<th>32 Bits</th>
<th>48 Bits</th>
<th>64 Bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>DPSH [97]</td>
<td>0.295</td>
<td>0.346</td>
<td>0.391</td>
<td>0.427</td>
<td></td>
</tr>
<tr>
<td>DTSH [166]</td>
<td>0.304</td>
<td>0.342</td>
<td>0.361</td>
<td>0.378</td>
<td></td>
</tr>
<tr>
<td>FastHash [102]</td>
<td>0.324</td>
<td>0.394</td>
<td>0.433</td>
<td>0.456</td>
<td></td>
</tr>
<tr>
<td>StructHash [101]</td>
<td>0.369</td>
<td>0.474</td>
<td>0.538</td>
<td>0.582</td>
<td></td>
</tr>
<tr>
<td>MIHash</td>
<td><strong>0.384</strong></td>
<td><strong>0.496</strong></td>
<td><strong>0.554</strong></td>
<td><strong>0.598</strong></td>
<td></td>
</tr>
</tbody>
</table>

Table 4.3: 22K LabelMe results with GIST features. MIHash significantly improves over the state-of-the-art methods.

To further emphasize the merits of MIHash, we consider shallow model experiments on the 22K LabelMe dataset. In this benchmark, we only consider the overall best shallow and deep learning methods in past experiments. Also, to solely put emphasis on comparing the hash mapping learning objectives, all deep learning methods use a one-layer embedding on top of the GIST descriptor [122]. The GIST descriptor is prominently used even in many recent hashing studies (e.g., as in [102] and [101]), and its usage nullifies the feature learning aspect in deep hashing techniques, enabling a more direct comparison to shallow hashing methods. Table 4.3 gives the results, and we can see that FastHash and StructHash outperform deep learning methods DPSH and DTSH on this benchmark. This indicates that the prowess of DPSH and DTSH might come primarily through feature learning. On the other hand, MIHash is the best performing method across all code lengths, despite using a simpler
one-layer embedding function compared to FastHash and StructHash, which use non-linear boosted decision trees. This further validates the effectiveness of our mutual information objective in capturing the neighborhood structure.

4.2.3 Discussions

The Distribution Separating Effect

To demonstrate that MIHash indeed separates neighbor and non-neighbor distance distributions, we consider a simple experiment. Specifically, we learn a single-layer model on top of precomputed $fc7$-layer features on the CIFAR-10 dataset, with 20K training examples. The learning is done in an online fashion, which means that each training example is processed only once.

In Figure 4.2, we plot the distributions $p^+_D$ and $p^-_D$, averaged on the CIFAR-10 test set, before and after learning MIHash. The hash mapping parameters are initialized
using LSH, and lead to high overlap between the distributions, although they do not totally overlap due to the use of strong pretrained features. After learning, the overlap is significantly reduced, with $p^+_D$ pushed towards zero hamming distances. Consequently, the mAP value increases to 0.68 from 0.22.

**MI vs. Ranking Metrics**

To evaluate the performance of hashing algorithms for retrieval tasks, it is common to use ranking-based metrics, and the most notable example is mean Average Precision (mAP). We note that there exists strong correlations between our mutual information criterion and AP, and provide empirical evidence in Figure 4-3. In the left plot, we display the training objective value as computed from (4.10) and the mAP score with respect to the epoch index, from the 32-bit experiment on CIFAR-10 (setting 1). To further demonstrate the correlation, we also conducted an additional experiment in which 100 instances are selected as the query set, and the rest are used as the database. The hash mapping parameters are randomly sampled from a Gaussian
distribution, similar to LSH \[54\], and the experiment is conducted 50 times. From the right plot in Figure 4-3, we can see that the relationship between MI and AP is almost linear, which is also validated by the Pearson Correlation Coefficient score of 0.98.

We give an intuitive explanation to the strong correlation. Given a query, AP is optimized when all of its neighbors are ranked above all the non-neighbors in the database. On the other hand, MI is optimized when the distribution of neighbor distances has no overlap with the distribution of non-neighbor distances. Therefore, we can see that AP and MI are optimized by the same optimal solution. Conversely, AP is suboptimal when neighbors and non-neighbors are interleaved in the ranking, so is MI when the distance distributions have nonzero overlap. Therefore, mutual information serves as a general-purpose surrogate metric for ranking.

**Visualization of Learned Hash Codes**

We also visualize the learned hash codes using t-SNE \[157\]. In Figure 4-4, we plot the visualization for 48-bit binary embeddings produced by MIHash and the top
competing method, HashNet, on ImageNet100. For ease of visualization, we randomly sample 10 classes from the test set and color-code them.

MIHash produces binary embeddings that separate different classes well into separate clusters. This is consistent with the formulation of MIHash, in which the class overlap is quantified via mutual information, and minimized. On the other hand, binary codes generated by HashNet have higher overlap between classes. This is also consistent with the fact that HashNet does not specifically optimize for a criterion related to class overlap, but belongs to the simpler “affinity matching” family of approaches.

4.3 Summary

We take an information-theoretic approach to hashing and propose a novel hashing method, called MIHash, in this work. It is based on minimizing neighborhood ambiguity in the learned Hamming space, which is crucial in maintaining high performance in nearest neighbor retrieval. We adopt the well-studied mutual information from information theory to quantify neighborhood ambiguity, and show that it has strong correlations with standard ranking-based retrieval performance metrics. Then, to optimize mutual information, we take advantage of recent advances in deep learning and stochastic optimization, and parameterize our embedding functions with deep neural networks. We perform a continuous relaxation on the NP-hard optimization problem, and use stochastic gradient descent to optimize the networks. In particular, our formulation maximally utilizes available supervision within each minibatch, and can be efficiently implemented.

When evaluated on four standard image retrieval benchmarks, MIHash is shown to learn high-quality compact binary codes, and it achieves superior nearest neighbor retrieval performance compared to existing supervised hashing techniques. A limi-
tion of MIHash is that it only considers the binary relevance setup, lacking the ability to directly handle graded relevance. Also, although mutual information is shown to be a good ranking surrogate, a mismatch is still left between this training objective and test-time evaluation metrics such as AP and NDCG. In the next chapter, we will develop an improved supervised hashing solution to address these limitations.
Chapter 5

Hashing as Tie-Aware Learning to Rank

The previous chapter has developed MIHash, a hashing method for the binary relevance case, based on the optimization of Mutual Information (MI). An important insight is that MI is strongly correlated to standard ranking metrics such as Average Precision (AP) and Normalized Discounted Cumulative Gain (NDCG). Therefore, it can be used successfully as a surrogate objective for ranking.

In this chapter, we instead attempt to directly optimize learning objectives that closely match test-time performance measures. Since nearest neighbor retrieval performance is frequently measured using AP and NDCG, we propose a novel learning to rank formulation to tackle these difficult optimization problems. Our main contribution is a gradient-based method that directly optimizes these ranking metrics for hashing. Compared to MIHash, it additionally handles the case of graded relevance, which is common in many information retrieval tasks. Coupled with deep neural networks, this method achieves state-of-the-art results.

Our formulation is inspired by a simple observation. When performing retrieval with binary vector encodings and the integer-valued Hamming distance, the resulting ranking usually contains ties, and different tie-breaking strategies can lead to different results (Fig. 5.1). In fact, ties are a common problem in ranking, and much attention has been paid to it, including in Kendall’s classical work on rank correlation [80], and in the modern information retrieval literature [21, 112]. Unfortunately, the learning to hash literature largely lacks tie-awareness, and current evaluation protocols rarely
When applying hashing for nearest neighbor retrieval, the integer-valued Hamming distance produces ties (items that share the same distance). If left uncontrolled, different tie-breaking strategies could give drastically different values of the evaluation metric, e.g. AP. We address this issue by using tie-aware ranking metrics that implicitly average over all the permutations in closed form. We further use tie-aware ranking metrics as optimization objectives in deep hashing networks, leading to state-of-the-art results.

Our natural next step is to learn hash functions by optimizing tie-aware ranking metrics. Similar to the case of MIHash, to solve the associated discrete and NP-hard optimization problems, we relax the problems into their continuous counterparts where closed-form gradients are available, and then perform gradient-based optimization with deep neural networks. We specifically study the optimization of AP and NDCG, two ranking metrics that are widely used in evaluating nearest neighbor
retrieval performance. Our results establish the new state-of-the-art for these metrics in common image retrieval benchmarks.

5.1 Tie-Awareness in Hashing

When evaluating information retrieval systems, special attention is required when there exist ties in the distances [21, 112]. In this case, the ranking $R$ is not unique as the tied items can be ordered arbitrarily, and the tie-breaking strategy may have a sizable impact on the result. We have given an example in Fig. 5·1. Surprisingly, we found that current ranking-based hashing evaluation protocols usually do not take tie-breaking into account, which could result in ambiguous comparisons or even unfair exploitation. Perhaps more importantly, ties render the formulation of direct optimization unclear: what tie-breaking strategy should we assume when using AP or NDCG as optimization objectives? Thus, we believe that it is important to seek tie-aware evaluation metrics for hashing.

Rather than picking a fixed tie-breaking strategy or relying on randomization, the tie-aware solution that we propose is to average the value of the ranking metric over all possible permutations of tied items. In other words, we take the expectation of the metric over all tie-breaking strategies, assuming that they all have the same probability of being chosen. This solution is appealing in several ways: it is deterministic, it is unambiguous and cannot be exploited, and it reduces to the ordinary version when there are no ties. However, there is one caveat: generating all permutations for $n$ tied items requires $O(n!)$ time, which is super-exponential and prohibitive. Fortunately, McSherry and Najork [112] observe that the average can be computed implicitly for commonly used ranking metrics, and gives their tie-aware versions in closed form. Based on this result, we further describe how to efficiently compute tie-aware ranking metrics by exploiting the structure of the Hamming distance.
We focus on AP and NDCG, and denote the tie-aware versions of AP and (N)DCG as AP\(_T\) and (N)DCG\(_T\), respectively. First, we define some notation. With integer-valued Hamming distances, we redefine the ranking \( R \) to be a collection of \((b + 1)\) “ties”, i.e. \( R = \{ R^{(0)}, \ldots, R^{(b)} \} \), where

\[
R^{(d)} = \{ i | d_\Phi(x_q, x_i) = d \}
\]

is the index set of retrieved items having Hamming distance \( d \) to the query. We define a set of histograms conditioned on relevance values, \((n_{0,v}, \ldots, n_{b,v})\), and their cumulative sums \((N_{0,v}, \ldots, N_{b,v})\), where

\[
n_{d,v} = |R^{(d)} \cap \{i | A_q(i) = v\}|, \forall v \in \mathcal{V}, \tag{5.1}
\]

\[
N_{d,v} = \sum_{j \leq d} n_{j,v}. \tag{5.2}
\]

We also define the total histograms as \( n_d = \sum_{v \in \mathcal{V}} n_{d,v} \) with cumulative sum \( N_d = \sum_{j \leq d} n_j \).

Next, Proposition \( \square \) gives the closed forms of AP\(_T\) and DCG\(_T\). We give proof in the appendix.

**Time complexity Analysis.** Let \(|S| = N\). Given the computed Hamming distances \( \{d_\Phi(x_q, x) | x \in S\} \), the first step is to generate the ranking \( R \), or populate the ties \( \{R^{(d)}\} \). This step is essentially the **counting sort** for integers, which has \( O(bN) \) time complexity. Computing either AP\(_T\) or DCG\(_T\) then takes \( O(\sum_d n_d) = O(N) \) time, which makes the total time complexity \( O(bN) \). In our formulation, the number of bits \( b \) is a constant, and therefore the complexity is linear in \( N \). In contrast, for real-valued distances, sorting generally takes \( O(N \log N) \) time.

For the normalized NDCG\(_T\), the normalizing factor is unaffected by ties, but computing it still requires sorting the gain values in descending order. Under the
Proposition 1. Both $\text{AP}_T$ and $\text{DCG}_T$ decompose additively over the ties. For $\mathcal{V} = \{0, 1\}$, let $n^+_d \triangleq n_{d,1}, N^+_d \triangleq N_{d,1}$, and $N^+ = \sum_d n^+_d$, the contribution of each tie $R^{(d)}$ to $\text{AP}_T$ is computed as

$$\text{AP}_T(R^{(d)}) = \frac{n^+_d}{n_d N^+} \sum_{t=N_{d-1}+1}^{N_d} \frac{N^+_{d-1} + \frac{(t - N_{d-1} - 1)}{t} n^+_{d-1} - 1}{n^+_d - 1}.$$  \hspace{1cm} (5.4)

For $\text{DCG}_T$, the contribution of $R^{(d)}$ is

$$\text{DCG}_T(R^{(d)}) = \sum_{i \in R^{(d)}} \frac{G(A_q(i))}{n_d} \sum_{t=N_{d-1}+1}^{N_d} D(t) = \sum_{v \in \mathcal{V}} \frac{G(v)n_{d,v}}{n_d} \sum_{t=N_{d-1}+1}^{N_d} D(t).$$  \hspace{1cm} (5.5)

Proof. See appendix. \hfill \Box

assumption that the set of relevance values $\mathcal{V}$ consists of non-negative integers, the number of unique gain values is $|\mathcal{V}|$, and counting sort can be applied in $O(|\mathcal{V}|N)$ time. The total time complexity is thus $O((b + |\mathcal{V}|)N)$, which is also linear in $N$ provided that $|\mathcal{V}|$ is known. We note that counting sort on Hamming distances is also used by Lin et al. \cite{01} to speed up loss-augmented inference for their NDCG surrogate loss.

5.2 Optimizing Tie-Aware Ranking Metrics

In this section, we describe our approach to optimizing tie-aware ranking metrics. Similar to the optimization of mutual information for discrete hashing, the optimization problems considered here are also NP-hard, since they involve combinatorial search over all configurations of binary bits. Instead, we are interested in a relaxation approach using gradient-based deep neural networks. Therefore, we apply continuous relaxation to the discrete optimization problems.
5.2.1 Continuous Relaxation

Our continuous relaxation needs to address two types of discrete variables. First, as is universal in hashing formulations, the bits in the hash code are binary. Second, the tie-aware metrics involve integer-valued histogram bin counts \( \{n_{d,v}\} \).

We first tackle the binary bits. Similar to the previous chapter, we relax the binary bits using a tanh approximation:

\[
\Phi(x) = (\phi_1(x), \ldots, \phi_b(x)),
\]

\[
\approx (\tanh(\gamma f_1(x; w)), \ldots, \tanh(\gamma f_b(x; w))).
\]

The constant \( \gamma \) is a scaling parameter.

After this relaxation, the real-valued hash mapping and distance function are denoted \( \hat{\Phi} \) and \( \hat{d}_\Phi \), respectively. The remaining discreteness is from the histogram bin counts \( \{n_{d,v}\} \). We again relax them into real-valued “soft histograms” \( \{c_{d,v}\} \), whose cumulative sums are denoted \( \{C_{d,v}\} \). However, we face another difficulty: the definitions of \( \text{AP}_T \) \( \text{(5.4)} \) and \( \text{DCG}_T \) \( \text{(5.5)} \) both involve a finite sum with lower and upper limits \( N_{d-1} + 1 \) and \( N_d \), which themselves are variables to be relaxed. We approximate these finite sums by continuous integrals, removing the second source of discreteness. We outline the results in Proposition 2 and leave proof and error analysis to the appendix.

Importantly, both relaxations have closed-form derivatives. The differentiation for \( \text{AP}_r \) \( \text{(5.8)} \) is straightforward, while for \( \text{DCG}_r \) it removes the integral in \( \text{(5.9)} \).

5.2.2 End-to-end Optimization

We perform end-to-end learning with gradient ascent. First, as mentioned above, the continuous relaxations \( \text{AP}_r \) and \( \text{DCG}_r \) have closed-form partial derivatives with respect to the soft histograms \( \{c_{d,v}\} \). Next, we consider differentiating the histogram
Proposition 2. The continuous relaxations of $\text{AP}_T$ and $\text{DCG}_T$, denoted as $\text{AP}_r$ and $\text{DCG}_r$ respectively, are as follows:

\[
\text{AP}_r(R^{(d)}) = \frac{c_d^+ (c_d^+ - 1)}{N^+(c_d^+ - 1)} + \frac{c_d^+}{N^+ c_d} \left[ C_{d-1}^+ + 1 - \frac{c_d^+ - 1}{c_d - 1} (C_{d-1} - 1) \right] \ln \frac{C_d}{C_{d-1}}, \tag{5.8}
\]

\[
\text{DCG}_r(R^{(d)}) = \ln 2 \sum_{v \in V} G(v) c_{d,v} \int_{C_d}^{C_{d+1}} \frac{dt}{\ln t}. \tag{5.9}
\]

Proof. See appendix. \hfill \Box

entries. Note that before relaxation, the discrete histogram $(n_{0,v}, \ldots, n_{b,v})$ for $\forall v \in V$ is constructed as follows:

\[
n_{d,v} = \sum_{x_i : A_q(i) = v} 1[d_{q}(x_q, x_i) = d], \quad d = 0, \ldots, b. \tag{5.10}
\]

To relax $n_{d,v}$ into $c_{d,v}$, we employ the same technique from the previous chapter (details are thus omitted):

\[
c_{d,v} = \sum_{x_i : A_q(i) = v} \delta(d_{q}(x_q, x_i), d), \tag{5.11}
\]

and we differentiate $c_{d,v}$ using chain rule, e.g.

\[
\frac{\partial c_{d,v}}{\partial d_{q}(x_q)} = \sum_{x_i : A_q(i) = v} \frac{\partial \delta(d_{q}(x_q, x_i), d)}{\partial d_{q}(x_q, x_i)} - \hat{\Phi}(x_i), \tag{5.12}
\]

As shown in Fig. 5·2, we train our models using minibatch-based stochastic gradient ascent. Similar to the case of MIHash, in order to fully utilize supervision, within a minibatch of size $M$, each example is used as the query once, and participates in the database for some other example $M - 1$ times. Then, the objective is averaged over the $M$ queries. The full details of the backpropagation are given in Appendix A.
Figure 5.2: The flow of computation in the proposed TALR method. Input images are mapped to $b$-bit binary codes by a DNN ($b = 4$ in this example). During training, in a minibatch, each example is used as query to rank the rest of the batch, producing a histogram of Hamming distances with $(b+1)$ bins. Tie-aware ranking metrics ($\text{AP}_T$ shown here) are computed on these histograms, and averaged over the batch. To maintain end-to-end differentiability, we derive continuous relaxations for $\text{AP}_T$ and $\text{NDCG}_T$, and employ two differentiable approximations to non-differentiable operations.

5.3 Implementation Details

We have mentioned that the gradients of the continuous relaxations to the tie-aware ranking metrics can be evaluated in closed form. This fact is important for performing gradient ascent. However, it can be seen that the continuous relaxations are quite complicated, thus deriving and implementing the gradients by hand can be tedious. Also, automatic differentiation tools can only offer limited help in this case.

Below, we present simplified versions of the continuous objectives, that are much easier to derive and implement. Specifically, for $\text{AP}_r$ we give an inexact approximation, and for $\text{DCG}_r$ we give a lower bound. Empirically, optimizing the simplified versions gives performances that are very similar to the optimizing the original continuous relaxations.
5.3.1 A Simplified Version of Tie-Aware AP

We repeat the definition of $AP_r$ below:

$$AP_r(R^{(d)}) = \frac{c_d^+(c_d^+ - 1)}{(c_d - 1)N^+} + \frac{c_d^+}{c_d N^+} \left[ C_{d-1}^+ 1 - \frac{c_d^+ - 1}{c_d - 1}(C_{d-1} + 1) \right] \ln \frac{C_d}{C_{d-1}}. \quad (5.13)$$

Suppose that we want to differentiate $AP_r$ with respect to some histogram bin $c_d^+$.

The exact partial derivative can be written as:

$$\sum_{d=0}^{b} \frac{\partial AP_r(R^{(d)})}{\partial c_d^+} = \sum_{d=0}^{b} \frac{\partial}{\partial c_d^+} \left\{ \frac{c_d^+(c_d^+ - 1)}{(c_d - 1)N^+} + \frac{c_d^+}{c_d N^+} \left[ C_{d-1}^+ 1 - \frac{c_d^+ - 1}{c_d - 1}(C_{d-1} + 1) \right] \ln \frac{C_d}{C_{d-1}} \right\}. \quad (5.15)$$

Again, computing this can be tedious and error-prone.

To derive an inexact version with a simpler form, we first do a change of variables in the definition of $AP_T$ (5.4) and rewrite it as

$$AP_T(R^{(d)}) \approx \frac{1}{N^+} \frac{n_d^+}{n_d} \cdot \frac{N_{d-1}^+ + 1 + \frac{n_{d-1}^+ n_{d-1}^+ - 1}{n_{d-1}}}{N_{d-1} + \frac{n_{d-1}^+}{2}} \quad (5.16)$$

Then, we simply replace the sum $\sum_{j=1}^{n_d^+} \eta_d(j)$ by repeating its midpoint: $\sum_{j=1}^{n_d^+} \eta_d(j) \approx n_d \eta_d(\frac{n_d^+ + 1}{2})$. Then, the simplified version is derived as

$$AP_T(R^{(d)}) \approx \frac{1}{N^+} \frac{n_d^+}{n_d} \cdot \frac{N_{d-1}^+ + 1 + \frac{n_{d-1}^+ n_{d-1}^+ - 1}{n_{d-1}}}{N_{d-1} + \frac{n_{d-1}^+}{2}} \quad (5.18)$$
The simplified continuous relaxation, which we name $AP_s$, is now as follows:

$$AP_s(R^{(d)}) = \frac{C_d^+}{N^+} \cdot \frac{C_{d-1}^+ + C_d^+ + 1}{C_{d-1} + C_d + 1}.$$  \hfill (5.20)

Deriving the closed-form gradients for this simplification is much less involved.

### 5.3.2 A Lower Bound for Tie-Aware DCG

Similar to the case of tie-aware AP, we also derive a simplified version for the tie-aware DCG. Somewhat differently, we will actually derive a lower bound for $DCG_T$ and then continuously relax it. Maximizing this lower bound is a principled way to maximize $DCG_T$.

First we recall the definition of $DCG_T$, plugging in the actual forms of the gain and discount functions:

$$DCG_T(R^{(d)}) = \sum_{v \in V} \frac{(2^v - 1)n_{d,v}}{n_d} \cdot \sum_{t=N_{d-1}+1}^{N_d} \frac{1}{\log_2 (t+1)}.$$  \hfill (5.21)

Note that the function $1/\log_2(\cdot)$ is a convex function. Therefore we can use Jensen’s inequality to lower bound the second sum:

$$DCG_T(R^{(d)}) \geq \sum_{v \in V} \frac{(2^v - 1)n_{d,v}}{n_d} \cdot n_d \cdot \frac{1}{\log_2 (N_{d-1} + \frac{n_d+1}{2})}$$

$$= \sum_{v \in V} \frac{(2^v - 1)n_{d,v}}{\log_2 (N_{d-1} + \frac{1}{2}n_d + \frac{3}{2})}.$$  \hfill (5.22)

And now we can see that the continuous relaxation of the lower bound, denoted as $DCG_s$, should be

$$DCG_s(R^{(d)}) = \frac{\sum_{v \in V} (2^v - 1)c_{d,v}}{\log_2 (C_{d-1} + \frac{1}{2}c_d + \frac{3}{2})}.$$  \hfill (5.23)
5.4 Experiments

5.4.1 Datasets and Setup

For evaluating our proposed method in this chapter, we use the same datasets and splits as in MIHash (Chapter 4), namely, CIFAR-10 \[84\] (two settings), NUS-WIDE \[36\] (two settings), 22K LabelMe \[134\], and ImageNet100 \[25\]. Each dataset is split into a test set and a database, and examples from the database are used in training. At test time, queries from the test set are used to perform Hamming ranking on the database, and the performance metric is averaged over the test set. We use both AP and NDCG as evaluation metrics.

Retrieval-based evaluation of supervised hashing was recently put into question by Sablayrolles et al. \[136\], who point out that for multi-class datasets, binary encoding of classifier outputs is already a competitive solution. While this is an important point, deriving pairwise relevance from multi-class label agreement is a special case in our formulation. As mentioned in Section 2.1, our formulation uses a general pairwise relevance oracle $A$, which may or may not be derived from labels, and can be either binary or graded. In fact, the datasets we consider range from multi-class/single-label (CIFAR-10, ImageNet100) to multi-label (NUS-WIDE) and unlabeled (LabelMe), and only the first case can be addressed by multi-class classification. For graded relevance, we also propose a new evaluation protocol using NDCG.

We term our method TALR (Tie-Aware Learning to Rank), and compare it against a range of classical and state-of-the-art hashing methods. Due to the vast hashing literature, an exhaustive comparison is not feasible. Focusing on the learning to rank aspect, we select representative methods from all three categories:

- **Pointwise (pair-based).** Methods that define loss functions on instance pairs: Binary Reconstructive Embeddings (BRE) \[87\], Fast Supervised Hashing.
ing (FastHash) \cite{102}, Hashing using Auxiliary Coordinates (MACHash) \cite{127}, Deep Hashing with Pair Supervision (DPSH) \cite{97}, and Hashing by Continuation (HashNet) \cite{25}.

- **Pairwise (triplet-based).** We include a recent method, Deep Hashing with Triplet Supervision (DTSH) \cite{166}.

- **Listwise.** We compare to two listwise ranking methods: Structured Hashing (StructHash) \cite{101} which optimizes an NDCG surrogate, and Hashing with Mutual Information (MIHash) \cite{62} which optimizes mutual information as a ranking surrogate for the binary relevance case.

Since tie-aware evaluation of Hamming ranking performance has not been reported in the hashing literature, we re-train and evaluate all methods using publicly available implementations.

### 5.4.2 Binary Relevance: AP

We evaluate AP on the three labeled datasets, CIFAR-10, NUS-WIDE, and ImageNet100. As we mentioned earlier, for labeled data, relevance can be inferred from label agreements. Specifically, in CIFAR-10 and ImageNet100, two examples are neighbors (i.e. have relevance 1) if they share the same class label. In the multi-labeled NUS-WIDE, two examples are neighbors if they share at least one label.

**CIFAR-10 and NUS-WIDE**

We first carry out AP optimization experiments on the two well-studied datasets, CIFAR-10 and NUS-WIDE. For these experiments, we perform finetuning using the ImageNet-pretrained VGG-F network \cite{31}. For methods that are not amenable to end-to-end training, we train them on fc7-layer features from VGG-F. On CIFAR-10,
we compare all methods in the first setting (S1), and in the second setting (S2) we compare the end-to-end methods: DPSH, DTSH, MIHash, and ours.

We present AP optimization results in Table 5.1. By optimizing the relaxation of AP\textsubscript{T} in an end-to-end fashion, our method (TALR-AP) achieves the new state-of-the-art in AP on both datasets, outperforming all the pair-based and triplet-based methods by significant margins. Compared to listwise ranking solutions, TALR-AP outperforms StructHash significantly by taking advantage of deep learning, and outperforms MIHash by matching the training objective to the evaluation metric. A side note is that for NUS-WIDE, it is customary in previous work [97, 166] to report AP evaluated at maximum cutoff of 5K (AP@5K), since ranking the full database is inefficient using general-purpose sorting algorithms. However, focusing on the top of the ranking overestimates the true AP, as seen in Table 5.1. Using counting sort, we are able to evaluate AP\textsubscript{T} on the full database efficiently, and TALR-AP also outperforms other methods in terms of AP@5K.

**ImageNet100**

For ImageNet100 experiments, we again follow the setup in HashNet [25] and fine-tune the AlexNet architecture [85] pretrained on ImageNet. As in [25], the minibatch size is set to 256 for all methods, and the learning rate for the pretrained convolution and fully connected layers are scaled down, since the model is fine-tuned on the same dataset that it was originally trained on. AP at cutoff 1000 (AP@1000) is used as the evaluation metric.

ImageNet100 results are summarized in Table 5.2. TALR-AP outperforms both competing methods, and the improvement is especially significant with short hash codes (16 and 32 bits). This indicates that our direct optimization approach produces better compact binary representations that preserve desired rankings.
<table>
<thead>
<tr>
<th>Method</th>
<th>CIFAR-10 (S1, AP_T)</th>
<th>NUS-WIDE (S1, AP_T)</th>
<th>NUS-WIDE (S1, AP@5K)</th>
</tr>
</thead>
<tbody>
<tr>
<td>BRE [87]</td>
<td>0.361 0.448 0.502 0.533</td>
<td>0.561 0.578 0.589 0.607</td>
<td></td>
</tr>
<tr>
<td>MACHash [127]</td>
<td>0.628 0.707 0.726 0.734</td>
<td>0.361 0.361 0.361 0.361</td>
<td></td>
</tr>
<tr>
<td>FastHash [102]</td>
<td>0.678 0.729 0.742 0.757</td>
<td>0.646 0.686 0.698 0.712</td>
<td></td>
</tr>
<tr>
<td>StructHash [101]</td>
<td>0.664 0.693 0.691 0.700</td>
<td>0.639 0.645 0.666 0.669</td>
<td></td>
</tr>
<tr>
<td>DPSH [97] *</td>
<td>0.720 0.757 0.757 0.767</td>
<td>0.658 0.674 0.695 0.700</td>
<td></td>
</tr>
<tr>
<td>DTSH [166]</td>
<td>0.725 0.773 0.781 0.810</td>
<td>0.660 0.700 0.707 0.723</td>
<td></td>
</tr>
<tr>
<td>MIHash [62]</td>
<td>0.687 0.775 0.786 0.822</td>
<td>0.652 0.693 0.709 0.723</td>
<td></td>
</tr>
<tr>
<td>TALR-AP</td>
<td><strong>0.732</strong> <strong>0.789</strong> <strong>0.800</strong> <strong>0.826</strong></td>
<td><strong>0.709</strong> <strong>0.734</strong> <strong>0.745</strong> <strong>0.752</strong></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Method</th>
<th>NUS-WIDE (S2, AP_T)</th>
<th>NUS-WIDE (S2, AP@5K)</th>
</tr>
</thead>
<tbody>
<tr>
<td>DPSH [97] *</td>
<td>0.908 0.909 0.917 0.932</td>
<td>0.758 0.793 0.818 0.830</td>
</tr>
<tr>
<td>DTSH [166]</td>
<td>0.916 0.924 0.927 0.934</td>
<td>0.773 0.813 0.820 0.838</td>
</tr>
<tr>
<td>MIHash [62]</td>
<td>0.929 0.933 0.938 0.942</td>
<td>0.767 0.784 0.809 0.834</td>
</tr>
<tr>
<td>TALR-AP</td>
<td><strong>0.939</strong> <strong>0.941</strong> <strong>0.943</strong> <strong>0.945</strong></td>
<td><strong>0.795</strong> <strong>0.835</strong> <strong>0.848</strong> <strong>0.862</strong></td>
</tr>
</tbody>
</table>

* Trained using parameters recommended by authors of DTSH.

**Table 5.1:** AP comparison on CIFAR-10 and NUS-WIDE with VGG-F architecture. On CIFAR-10, we compare all methods in the first setting (S1), and deep learning methods in the second (S2). We report the tie-aware AP_T, and additionally AP@5K for NUS-WIDE. TALR-AP optimizes tie-aware AP using stochastic gradient ascent, and achieves state-of-the-art performance.
<table>
<thead>
<tr>
<th>Method</th>
<th>16 Bits</th>
<th>32 Bits</th>
<th>48 Bits</th>
<th>64 Bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>DTSH [166]</td>
<td>0.458</td>
<td>0.566</td>
<td>0.611</td>
<td>0.644</td>
</tr>
<tr>
<td>HashNet [25]</td>
<td>0.5059</td>
<td>0.6306</td>
<td>0.6633</td>
<td>0.6835</td>
</tr>
<tr>
<td>MIHash [62]</td>
<td>0.5688</td>
<td>0.6608</td>
<td>0.6852</td>
<td>0.6947</td>
</tr>
<tr>
<td>TALR-AP</td>
<td><strong>0.5892</strong></td>
<td><strong>0.6689</strong></td>
<td><strong>0.6985</strong></td>
<td><strong>0.7053</strong></td>
</tr>
</tbody>
</table>

**Table 5.2:** AP@1000 results on ImageNet100 with AlexNet [85]. Our method, TALR-AP, outperforms state-of-the-art solutions using mutual information [62] and continuation methods [25].

State-of-the-art performance with compact codes has important implications for cases where memory and storage resources are restricted (*e.g.* mobile applications), and for indexing large-scale databases.

5.4.3 Graded Relevance: NDCG

We evaluate NDCG optimization with a graded relevance setup, *i.e.* the set $\mathcal{V}$ is a finite set of non-negative integers. Graded relevance is common in information retrieval tasks, and offers more fine-grained specification of the desired structure of the learned Hamming space. To our knowledge, this setup has not been considered in the supervised hashing literature.

In the multi-label NUS-WIDE dataset, we define the relevance value between two examples as the number of labels they share, and keep other settings the same as in the AP experiment. For the unlabeled LabelMe dataset, we derive relevance values by thresholding the Euclidean distances between examples. Inspired by an existing binary relevance setup [24] that defines neighbors as having Euclidean distance within the top 5% on the training set, we use four thresholds: $\{5\%, 1\%, 0.2\%, 0.1\%\}$ and assign relevance values: $\{1, 2, 5, 10\}$. This emphasizes assigning high ranks to the closest neighbors in the original feature space. We learn shallow models on precomputed GIST features [122] on LabelMe. For gradient-based methods, we learn a single fully-
connected layer, which means the learned hash functions are linear hyperplanes. For methods that are not designed to use graded relevance (FastHash, MACHash, DPSH, and MIHash), we convert the relevance values back to binary by thresholding at zero; this reduces to the standard binary relevance setup on both datasets.

We give NDCG results in Table 5.3. Again, our method with the tie-aware NDCG objective (TALR-NDCG) outperforms all competing methods on both datasets. Interestingly, on LabelMe where all methods are restricted to learn shallow models on GIST features, we observe slightly different trends compared to other datasets. For example, without learning deep representations, DPSH and DTSH appear to perform less competitively, indicating a mismatch between their objectives and the evaluation metric. The closest competitors to TALR-NDCG on LabelMe are indeed the two listwise ranking methods: StructHash which optimizes a NDCG surrogate using boosted decision trees, and MIHash which is designed for binary relevance. TALR-NDCG outperforms both methods, and notably does so with linear hash functions, which have lower learning capacity compared StructHash’s boosted decision trees. This highlights the benefit of our direct optimization formulation.

5.4.4 Tightening the Continuous Relaxation

As is common among relaxation-based hashing methods, we relax the binary constraints by replacing the discontinuous sgn function with tanh (5.7). With this simple relaxation, the performance gains are mainly due to optimizing the proposed objectives. Nevertheless, it is conceivable that the continuation method, e.g. as employed by HashNet [25], can tighten the relaxation and lead to better results. To provide a concrete example, in Table 5.4 we report improved results for TALR-AP on CIFAR-10 (setting 1), when we increase the scaling $\gamma$ in tanh over time, instead of using a fixed value.
<table>
<thead>
<tr>
<th>Method</th>
<th>NUS-WIDE</th>
<th>LabelMe</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>16 Bits</td>
<td>32 Bits</td>
</tr>
<tr>
<td>BRE [87]</td>
<td>0.805</td>
<td>0.817</td>
</tr>
<tr>
<td>MACHash [127]</td>
<td>0.821</td>
<td>0.821</td>
</tr>
<tr>
<td>FastHash [102]</td>
<td>0.885</td>
<td>0.896</td>
</tr>
<tr>
<td>DPSH [97]</td>
<td>0.895</td>
<td>0.905</td>
</tr>
<tr>
<td>DTSH [166]</td>
<td>0.896</td>
<td>0.905</td>
</tr>
<tr>
<td>StructHash [101]</td>
<td>0.889</td>
<td>0.893</td>
</tr>
<tr>
<td>MIHash</td>
<td>0.886</td>
<td>0.903</td>
</tr>
<tr>
<td>TALR-NDCG</td>
<td><strong>0.903</strong></td>
<td><strong>0.910</strong></td>
</tr>
</tbody>
</table>

* Evaluated on the 5K training subset due to kernel-based formulation.

**Table 5.3:** NDCG comparison on NUS-WIDE (VGG-F architecture) and LabelMe (shallow models on GIST features). TALR-NDCG optimizes tie-aware NDCG using stochastic gradient ascent, and consistently outperforms competing methods.
<table>
<thead>
<tr>
<th></th>
<th>12 bits</th>
<th>24 bits</th>
<th>32 bits</th>
<th>48 bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fixed $\gamma$</td>
<td>0.732</td>
<td>0.789</td>
<td>0.800</td>
<td>0.826</td>
</tr>
<tr>
<td>$\gamma \to \infty$</td>
<td>0.751</td>
<td>0.804</td>
<td>0.813</td>
<td>0.830</td>
</tr>
</tbody>
</table>

*Table 5.4:* Performance comparison between different continuous relaxation strategies for TALR-AP, on the CIFAR-10 dataset with the first experimental setting.

### 5.4.5 Effects of Tie-Braking Strategies

We lastly discuss the effect of tie-breaking in evaluating hashing algorithms. As mentioned in Section 5.1, tie-breaking is an uncontrolled parameter in current evaluation protocols, which can affect results, and even be exploited. To demonstrate this, we consider for example the AP experiment in CIFAR-10’s first setting. For each method included in this experiment, we plot the range of test set mAP spanned by all possible tie-breaking strategies. As can be seen in Fig. 5.3, the ranges corresponding to different methods generally overlap; therefore, without controlling for tie-breaking, relative performance comparison between different methods is essentially ambiguous. The ranges shrink as code length increases, since the number of ties generally decreases when there are more bins in the histogram.

Current hashing methods usually compute test-time AP and NDCG using random tie-breaking and general-purpose sorting algorithms. Interestingly, in all of our experiments, we observe that this produces values very close to the tie-aware $\text{AP}_T$ and $\text{NDCG}_T$. The reason is that with a randomly ordered database, averaging the tie-unaware metric over a sufficiently large test set behaves similarly to the tie-aware solution of averaging over all permutations. Therefore, the results reported in the current literature are indeed quite fair, and so far we have found no evidence of exploitation of tie-breaking strategies. Still, we recommend using tie-aware ranking metrics in evaluation, as they completely eliminate ambiguity, and counting sort on Hamming distances is much more efficient than general-purpose sorting.
We note that although random tie-breaking is an approximation to tie-awareness at test time, it does not answer the question of how to optimize the ranking metrics during training. Our original motivation is to optimize ranking metrics for hashing, and the existence of closed-form tie-aware ranking metrics is what makes direct optimization feasible.

5.5 Summary

We have proposed a new approach to hashing for nearest neighbor retrieval, with an emphasis on directly optimizing evaluation metrics used at test-time. A study into the commonly used “retrieval by Hamming ranking” setup led us to consider the issue of ties, and we advocate for using tie-aware versions of ranking metrics. We then make the novel contribution of optimizing tie-aware ranking metrics for hashing, focusing on the important special cases of AP and NDCG. To tackle the resulting discrete and NP-hard optimization problems, we derive their continuous relaxations. Then, we perform end-to-end stochastic gradient ascent with deep neural networks. This results in the new state-of-the-art for common image retrieval benchmarks.
that are typically based on optimizing their surrogates in a structured prediction framework (inspired structural SVMs [154]), our approach is one of “direct relaxation” that works on the specific functional forms of the tie-aware metrics. For the discrete Hamming distance, the good news is that the sorting operation is instantiated by histogram binning, which allows for simple and convenient differentiable relaxations. As we shall see in the following chapters, for real-valued distances, histogram-based quantization can also be employed to get good approximations of ranking metrics that require sorting to compute. Such an approximation can also be understood as a smoothing technique, in the spirit of [30].
Chapter 6

Local Descriptors Optimized for Average Precision

Extracting feature descriptors from local image patches is a common stage in many computer vision tasks involving alignment or matching. To replace handcrafted feature engineering, recently much attention has been paid to learning local feature descriptors. Despite exciting progress, certain levels of handcrafting are currently present in the design of learning objectives for local feature descriptors, making it difficult to have performance guarantees when the learned descriptors are integrated into larger pipelines. Indeed, according to a recent study [139], traditional handcrafted features such as SIFT [108] can still outperform learned ones in complicated tasks such as 3D reconstruction. In this chapter, we aim to improve the learning of local feature descriptors by optimizing better objective functions.

We argue that local feature descriptor learning is not a standalone problem, but rather a component in the optimization of larger pipelines. Therefore, the learning objectives should be designed in accordance with other pipeline components. Upon inspection of common local feature matching pipelines, we find that feature matching can be exactly formulated as nearest neighbor retrieval. Thus, we propose to employ listwise learning to rank for learning local feature descriptors. In particular, we use deep neural networks to directly optimize a ranking-based retrieval performance metric, Average Precision, by reusing and extending our earlier formulations in the
Figure 6.1: An example local feature-based image matching pipeline, where the task is to estimate the fundamental matrix $F$ between images $I = (I_1, I_2)$, using robust estimation techniques such as RANSAC [49]. We model the feature descriptor extractor using deep neural networks, and directly optimize a ranking-based objective (Average Precision) for the subsequent stage of descriptor matching.

Our proposed approach can learn both binary and real-valued local feature descriptors. Compared to recent approaches, we optimize a commonly adopted evaluation metric, and eliminate complex optimization heuristics. Descriptors learned with our approach achieve state-of-the-art results in benchmarks including UBC Phototour [170], HPatches [9], RomePatches [124], and the Oxford dataset [113].

An important feature of our proposed formulation is that it is general-purpose, as it optimizes the performance of the task-independent nearest neighbor matching stage, rather than a task-specific pipeline. Nevertheless, to better tailor the learned descriptors for feature matching, we also augment our formulation with task-specific improvements. First, we make use of the Spatial Transformer module [73] to effectively handle geometric noise and improve the robustness of matching, without requesting extra supervision. Also, for the challenging HPatches dataset, we design a clustering-based technique to mine additional patch-level supervision, which improves the performance of learned descriptors in the image matching task.
6.1 Feature Matching is Nearest Neighbor Retrieval

In this section, we motivate our approach by analyzing the descriptor matching stage, and point out that it corresponds to nearest neighbor retrieval. Then we discuss a learning to rank formulation to optimize ranking-based retrieval performance.

Consider Fig. 6·1 which depicts a pipeline for estimating the fundamental matrix between matching images $I_1$ and $I_2$. It consists of four stages: feature detection, descriptor extraction, descriptor matching, and robust estimation. Suppose we detect and extract $M$ local features from each image. The descriptor matching stage operates as follows: it computes the pairwise distance matrix with $M^2$ entries, and for each feature in $I_1$, looks for its nearest neighbor in $I_2$, and vice versa. Feature pairs that are mutual nearest neighbors become candidate matches in the robust estimation stage, such as RANSAC [49].

We point out that this matching process is exactly performing nearest neighbor retrieval: each feature in $I_1$ is used to query a database, which is the set of features in $I_2$. For good performance, true matches should be returned as top retrievals, while false matches are ranked as low as possible. Performance of the matching stage also directly reflects the quality of the learned descriptors, since it has no learnable parameters (only performs distance computation and sorting). To assess nearest neighbor matching performance, we again adopt Average Precision (AP), a commonly used evaluation metric. Recall that AP evaluates the performance of retrieval systems under the binary relevance assumption: retrievals are either “relevant” or “irrelevant” to the query. This naturally fits the local feature matching setup, where given a reference feature, features in a target image are either its true match or false match. Next, we learn binary and real-valued local feature descriptors to optimize AP.

\footnote{For simplicity, the distance ratio check \cite{108} is not considered.}
6.2 Optimizing Binary and Real-valued Descriptors

Binary Descriptors

Binary descriptors offer compact storage and fast matching, which are useful in applications with speed or storage restrictions. Although binary descriptors can be learned one bit at a time [153], here we take a gradient-based relaxation approach to learn fixed-length “hash codes”.

In the previous chapter, we have already developed a method for optimizing AP for the supervised hashing problem. It is easy to see that the problem of learning local feature descriptors is closely related to supervised hashing; in fact, the only major difference is that the descriptors are now extracted from local image patches, rather than entire images. Therefore, barring details in implementation, we can directly reuse our TALR-AP technique to learn binary local feature descriptors. The details are not repeated here.

Real-Valued Descriptors

To complete our formulation, we next consider real-valued descriptors, which are preferred in high-precision scenarios. In the real-valued case, let the descriptor be $\Psi$, which is modeled as a vector of neural network activations, with $L_2$ normalization:

$$\Psi_0(x) = (\psi_1(x; w), \psi_2(x; w), \ldots, \psi_m(x; w)) \in \mathbb{R}^m, \quad \Psi(x) = \frac{\Psi_0(x)}{\|\Psi_0(x)\|}. \quad (6.1)$$

$w$ stands for the learnable parameters in the network. The distance metric, $d$, is now the Euclidean distance between unit vectors, whose partial derivative $\partial d / \partial \Psi$ is

$$\frac{\partial d(x, x')}{\partial \Psi(x)} = \frac{\partial \sqrt{2 - 2\Psi(x)\Psi(x')^\top}}{\partial \Psi(x)} = -\frac{\Psi(x')}{d(x, x')}. \quad (6.2)$$
Lastly, backpropagation through the $L_2$ normalization operation is as follows:

$$\frac{\partial g}{\partial \Psi_0(x)} = \frac{1}{\|\Psi_0(x)\|} \left[ \frac{\partial g}{\partial \Psi(x)} - \Psi(x) \left( \Psi(x)^\top \frac{\partial g}{\partial \Psi(x)} \right) \right],$$  \hspace{1cm} (6.3)

where $g$ is some function of $\Psi$.

The main challenge in optimizing AP for real-valued descriptors is again the non-differentiable sorting, but real-valued sorting has no simple alternative form. However, histogram binning can be used as an approximation: we quantize real-valued distances using histogram binning, obtain the histograms, and then reduce the optimization problem to the previous one. With $L_2$-normalized vectors, the quantization is easy to implement as the Euclidean distance has closed range $[0, 2]$; we simply uniformly divide the interval $[0, 2]$ into $b$ bins. The chain rules in this case only need minimal modifications to account for the $L_2$ normalization operation, which we just discussed.

Differently from the case of binary descriptors, the number of histogram bins $b$ is now a free parameter, which involves a trade-off. On the one hand, a large $b$ reduces quantization error, which in fact achieves zero if each histogram bin contains at most one item. On the other hand, gradient computation for approximate histogram binning has linear complexity in $b$. In our experiments, we consistently obtain good results using $b \leq 25$.

### 6.3 Task-specific Improvements

In addition to the general-purpose learning to rank formulation, we develop two improvements that take the nature of local feature matching into account.

#### 6.3.1 Spatial Transformer Module

To improve the robustness of local features for matching, it is key to build invariance to geometric noise into the descriptor: for example, SIFT \cite{108} estimates orienta-
tion and affine shape to normalize input patches, and LIFT \cite{176} includes a learned orientation estimation module. Likewise, we can also include a geometric alignment module in our descriptor networks. Our choice is the Spatial Transformer \cite{73}, which aligns input patches by predicting a 6-DOF affine transformation, without requiring extra supervision. As is standard practice, the Spatial Transformer is initialized to output identity (directly copy input patches), and the learning rate of the affine transformation prediction layer is scaled down by 100 times compared to other layers in the network. In our experiments, this module is able to correct geometric distortion, and consistently improve performance.

A naïve application of the Spatial Transformer, however, leads to the boundary effect \cite{99}: when the predicted transformation requires sampling outside the boundaries of the input, the default zero-padding creates unfilled regions near the boundaries in the output. Since the input patches to the Spatial Transformer have limited size (42x42 in our network), out-of-boundary sampling frequently happens in operations such as zooming out and rotation, and can affect alignment by introducing unwanted image gradients. Instead, we first pad the input patch by repeating its boundary pixels and then sample according to the predicted transformation, which prevents sharp image gradients near boundaries. This is visually illustrated in Fig. 6·2, using patches from the challenging HPatches dataset, which has the largest amount of geometric noise among the datasets that we consider. Although using zero padding still produces decent alignment, it affects the appearance of sampled patches, and does not help to improve final performance. Our boundary padding produces much more visually plausible patches after sampling, and gives a good approximation to re-sampling from the original images.

\footnote{Implemented in Matlab using the \texttt{replicate} mode of the \texttt{padarray} function.}
Figure 6.2: Patch alignment using the Spatial Transformer [73] in HPatches, where patches come in groups of 16. The aligned patches are used as inputs to the descriptor network. First row: original patches. Second row: aligned patches, using our boundary padding. Third row: aligned patches, using the default zero padding.
6.3.2 Label Mining

While our formulation directly optimizes for the task of patch retrieval, it is also possible to address higher-level tasks. We demonstrate this with the image matching task in the challenging HPatches dataset \cite{9}, which contains patches extracted from matching image sequences.

The image matching task in HPatches is formulated similarly as patch retrieval, which involves retrieving matching patches in a pool of “distractors”. However, the distractors are defined differently. In patch retrieval, distractors do not include patches in the same image sequence as the query, due to concern of repeating structures in images. In image matching, images are matched against others in the same sequence, which means that all distractors are actually in-sequence. Thus, image matching performance can be improved by including in-sequence distractors when optimizing patch retrieval.

Since the 3D point correspondence for each training patch is given, it may appear that we can simply mark all patches that do not correspond to a certain 3D point as distractors for the corresponding patch. However, the risk is that when an image has repeating structures (e.g. windows on a building), patches that correspond to different 3D points could have nearly identical appearance, and forcing the network to distinguish between them would cause overfitting. Instead, we perform label mining to augment the set of distractors when optimizing patch retrieval in HPatches. To avoid noisy labels in the presence of repeating structures, we use a simple heuristic: clustering. For each image sequence, we cluster all patches based on visual appearance. Then, patches having high inter-cluster distance are marked as distractors for each other (with 3D verification). Note that label mining is not related to the hard negative mining heuristic, since its goal is to add additional supervision.

We use handcrafted visual features to represent patches in clustering. The best
Figure 6.3: We demonstrate label mining in HPatches, using four randomly selected image sequences. From top to bottom: v_london, i_steps, v_maskedman, i_yellowtent. The first image in each sequence is shown on the left, and on the right we visualize 5 randomly selected patch clusters, obtained using $K$-means. Each row corresponds to a cluster. A red arrow between clusters indicates that the inter-cluster distance is above a threshold, and their patches are used as distractors for each other. A gray arrow means that the inter-cluster distance is not high enough. Patches are generally more similar in appearance within the same sequence than across sequences, therefore mining the in-sequence distractors provides meaningful “hard negatives” for the learning.
feature found in our experiments is a combination of HOG [42] and raw pixel values, which captures both the geometric and illumination patterns. It is constructed as follows: a patch is resized to 64x64 to extract HOG features with 8x8 cell size, and then the same patch is resized to 16x16 and appended to the feature vector. The final feature dimensionality is 2240. Afterwards, we perform \( K \)-means clustering with \( K = 100 \) clusters. To derive a distance threshold, we compute all the pairwise distances between the cluster centers, and set the threshold at the \( p \)-th percentile of these distances. If two clusters have larger distance than the threshold, their patches are considered distractors for each other. Otherwise, they are considered “too visually similar”, and are ignored from each other’s distractor set. We use \( p = 20 \). Label mining is demonstrated in Fig. 6·3.

6.4 Experiments

We experiment with three patch-based datasets (examples are in Fig. 6·4): UBC Phototour [170], HPatches [9], RomePatches [124], and an image-based dataset, the
Oxford Dataset [113]. We use the CNN architecture recently proposed in L2Net [151], which consists of seven convolution layers, and is regularized with Batch Normalization [72] and Dropout [149]. We do not use the more complex “Center Surround” architecture. The input to the network is 32x32 grayscale, and we resize input patches to this size. When adding the Spatial Transformer module, we increase the input size to 42x42, and use 3 convolution layers to predict a 6-DOF affine transformation, which is then used to sample a 32x32 patch.

We name our descriptor DOAP (Descriptors Optimized for Average Precision), and test its binary and real-valued versions.

6.4.1 UBC Phototour Dataset

We first conduct experiments on the UBC Phototour dataset [170], a classical benchmark of descriptor performance. Patches are extracted from Difference-of-Gaussian detections in three image sequences: Liberty, Notre Dame, and Yosemite. Following the standard setup, we use six training/test combinations formed by the three sequences, and report patch verification performance in terms of false positive rate at 95% recall (FPR95).

We train our models on UBC Phototour with data augmentation, in the form of random flipping and 90-degree rotations, which showed consistent performance improvement in previous work. We compare to a range of existing descriptors, including both binary and real-valued, listed in Table 6.1. L2Net [151] and HardNet [115] are two leading methods, which optimize triplet-based losses with the same CNN architecture as ours. We also include methods that use the “Center Surround” architecture: CS-SNet-Gloss [89] and CS-L2Net, and we have applied the recent global regularization technique in [185] to HardNet, resulting in a more competitive method which we call HardNet-GOR. Compared to existing approaches, DOAP achieves state-of-the-
<table>
<thead>
<tr>
<th>Method</th>
<th>Train</th>
<th>Test</th>
<th>Liberty</th>
<th>Notredame</th>
<th>Yosemite</th>
<th>Liberty</th>
<th>Notredame</th>
<th>Yosemite</th>
<th>FPR95</th>
<th>Mean</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Real-valued descriptors</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SIFT [108]</td>
<td>128</td>
<td>29.84</td>
<td>22.53</td>
<td>27.29</td>
<td>26.55</td>
<td>8.05</td>
<td>6.64</td>
<td>4.36</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DeepDesc [142]</td>
<td>128</td>
<td>8.82</td>
<td>11.47</td>
<td>3.82</td>
<td>5.65</td>
<td>11.6</td>
<td>8.70</td>
<td>4.09</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MatchNet [57]</td>
<td>128</td>
<td>7.04</td>
<td>10.31</td>
<td>3.06</td>
<td>3.80</td>
<td>8.06</td>
<td>7.24</td>
<td>4.19</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TL-AS-GOR [185]</td>
<td>128</td>
<td>4.80</td>
<td>6.45</td>
<td>1.95</td>
<td>2.38</td>
<td>5.40</td>
<td>5.15</td>
<td>4.36</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DC-2ch2st+ [180]</td>
<td>512</td>
<td>4.85</td>
<td>7.20</td>
<td>1.90</td>
<td>2.11</td>
<td>5.00</td>
<td>8.39</td>
<td>4.19</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CS-SNet-GLoss+ [89]</td>
<td>256</td>
<td>3.69</td>
<td>4.91</td>
<td>0.77</td>
<td>1.14</td>
<td>3.09</td>
<td>2.67</td>
<td>2.71</td>
<td></td>
<td></td>
</tr>
<tr>
<td>L2Net+ [151]</td>
<td>128</td>
<td>2.36</td>
<td>4.7</td>
<td>0.72</td>
<td>1.29</td>
<td>2.57</td>
<td>1.71</td>
<td>2.23</td>
<td></td>
<td></td>
</tr>
<tr>
<td>HardNet+ [115]</td>
<td>128</td>
<td>2.28</td>
<td>3.25</td>
<td>0.57</td>
<td>0.96</td>
<td>2.13</td>
<td>2.22</td>
<td>1.90</td>
<td></td>
<td></td>
</tr>
<tr>
<td>HardNet-GOR+ [115, 185]</td>
<td>128</td>
<td>1.89</td>
<td>3.03</td>
<td>0.54</td>
<td>0.90</td>
<td>2.41</td>
<td>2.39</td>
<td>1.86</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CS-L2Net+ [151]</td>
<td>256</td>
<td>1.71</td>
<td>3.87</td>
<td>0.56</td>
<td>1.09</td>
<td>2.07</td>
<td>1.30</td>
<td>1.76</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DOAP+</td>
<td>128</td>
<td>1.54</td>
<td>2.62</td>
<td>0.43</td>
<td>0.87</td>
<td>2.00</td>
<td>1.21</td>
<td>1.45</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DOAP-ST+</td>
<td>128</td>
<td><strong>1.47</strong></td>
<td>2.29</td>
<td><strong>0.39</strong></td>
<td>0.78</td>
<td><strong>1.98</strong></td>
<td>1.35</td>
<td><strong>1.38</strong></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| **Binary descriptors**        |       |      |         |           |          |         |           |          |       |      |
| L2Net+ [151]                  | 128   | 7.44 | 10.29   | 3.81      | 4.31     | 8.81    | 7.45      | 7.01     |       |      |
| CS-L2Net+ [151]               | 256   | 4.01 | 6.65    | 1.90      | 2.51     | 5.61    | 4.04      | 4.12     |       |      |
| DOAP+                         | 256   | 3.18 | 4.32    | 1.04      | 1.57     | 4.10    | 3.87      | 3.01     |       |      |
| DOAP-ST+                      | 256   | **2.87** | 4.17 | **0.96** | 1.76 | **3.93** | **3.64** | **2.89** |       |      |

Table 6.1: Patch verification performance on UBC Phototour, where metric is false positive rate at 95% recall (FPR95). The best results are in **bold**. Second column shows dimensionality, and methods with suffix “+” are trained with data augmentation. Both the binary and real-valued versions of DOAP and DOAP-ST achieve state-of-the-art results.
We attribute the performance of DOAP and DOAP-ST to the listwise AP optimization. Listwise optimization automatically includes the “hard negative mining” heuristic in local ranking approaches, since it implicitly enforces the correct classification of all induced pairs and triplets. We then expect performance to improve when increasing training batch size, as larger batches lead to longer lists and increased likelihood of including hard negatives. We validate this by training the 128-dimensional DOAP model on Liberty, varying batch size between 256 and 4096, and monitoring the average of FPR95 on Notre Dame and Yosemite. Indeed, Fig. 6·5 shows that performance improves with batch size and saturates after 2048. Similar trends are also observed in HardNet [115], with saturation occurring at batch size 512. In contrast, the listwise optimization allows the performance of DOAP to saturate at a later stage.

**Figure 6·5:** Influence of training batch size for the 128-dimensional DOAP descriptor trained on Liberty, with data augmentation. Vertical axis: average of FPR95 on Notre Dame and Yosemite.
6.4.2 HPatches Dataset

HPatches \cite{9} consists of a total of over 2.5 million patches extracted from 116 image sequences, each with 6 images with known homography. Both viewpoint and illumination changes are included, and test cases have levels of difficulty *easy*, *hard*, and *tough*, according to the amount of geometric noise. Three evaluation tasks are considered (in increasing order of difficulty): patch verification, patch retrieval, and image matching.

In this experiment, we focus on comparing real-valued descriptors. We first include four baselines reported in \cite{9}: SIFT \cite{108}, RootSIFT \cite{6}, DeepDesc \cite{142}, and TFeat \cite{10}. Next, as \cite{115} reports results for L2Net and HardNet trained on the *Liberty* sequence of UBC Phototour, for fair comparison, we also report results for our models trained on *Liberty*. Finally, we train and evaluate three versions of our descriptor on HPatches: DOAP, DOAP-ST with the Spatial Transformer, and DOAP-ST-LM, which additionally uses label mining. We compare to the L2Net model trained on HPatches, and HardNet++, which is the HardNet model trained on the union of *Liberty* and HPatches. Note that CS-L2Net is excluded as it performs worse than L2Net in this more realistic dataset, which is consistent with the observations in \cite{89, 151}. When determining training/test sets, we use the “a” split: the test set contains 40 image sequences (20 viewpoint and 20 illumination), and the training set contains the other 76 sequences.

Fig. 6-6 presents results on HPatches.\footnote{Results for L2Net and HardNet are obtained using their publicly released models and may slightly differ from those reported in \cite{115}.} Our descriptors achieve state-of-the-art results for all three tasks, and all variants are better at handling *tough* test cases than competing methods. Specifically, DOAP and DOAP-ST obtain the best patch retrieval performance, which directly results from the optimization of patch retrieval
Figure 6.6: Results on the HPatches dataset, evaluated on the test set of the “a” split. No ZCA normalization [9] is used. Suffix indicates training set used (Lib: Liberty, no suffix: HPatches). HardNet++ is trained on the union of Liberty and HPatches. DOAP outperforms competing methods in all tasks, and all of its variants excel in handling tough test cases.
mAP. This optimization also gives state-of-the-art performance in patch verification. For the most challenging task of image matching, as mentioned in [9], patch retrieval performance is well correlated. However, due to the difference in task definition that we mentioned in Section 6.3, all methods see lower performance when tested for image matching. With the clustering-based label mining, DOAP-ST-LM significantly improves image matching mAP compared to the next best models: around 6% and 10% over DOAP-ST and L2Net, respectively. Notably, it achieves over 50% mAP even in the toughest test cases (tough geometric noise, illumination change). The inclusion of extra supervision also boosts patch retrieval performance, since in-sequence distractors provide harder negatives to learn from.

6.4.3 RomePatches Dataset

We next consider the RomePatches dataset [124], which contains 20,000 image patches of size 51x51, split equally into training and test sets. The task is patch retrieval. This dataset is constructed by performing SIFT matching on images taken in Rome, and keeping matching patches that satisfy 3D constraints. With such tailored construction, SIFT is unsurprisingly a strong baseline on RomePatches. In fact, in terms of test set mAP, previous methods, including pretrained AlexNet [85] and PhilippNet [48], could not surpass SIFT. The only method to do so was the CKN-grad variant proposed in [124], using 1024-dimensional descriptors.

We found that adding Spatial Transformers does not improve results, possibly because the patches are already well aligned (see examples in Fig. 6.4); therefore we only report results for the binary and real-valued DOAP. As seen in Table 6.2, the real-valued DOAP outperforms SIFT and other descriptors with 88.4% mAP on the test set, while the binary version also performs competitively. The comparison between DOAP and SIFT is fair, since they have the same input coverage and out-
<table>
<thead>
<tr>
<th>Method</th>
<th>Coverage</th>
<th>Dim.</th>
<th>Train</th>
<th>Test</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIFT</td>
<td>51x51</td>
<td>128</td>
<td>91.6</td>
<td>87.9</td>
</tr>
<tr>
<td>AlexNet-conv3</td>
<td>99x99</td>
<td>384</td>
<td>81.6</td>
<td>79.2</td>
</tr>
<tr>
<td>PhilippNet</td>
<td>64x64</td>
<td>512</td>
<td>86.1</td>
<td>81.4</td>
</tr>
<tr>
<td>CKN-grad</td>
<td>51x51</td>
<td>1024</td>
<td>92.5</td>
<td>88.1</td>
</tr>
<tr>
<td>DOAP</td>
<td>51x51</td>
<td>128</td>
<td>95.2</td>
<td>86.8</td>
</tr>
<tr>
<td>Binary DOAP</td>
<td>51x51</td>
<td>256</td>
<td>95.9</td>
<td>88.4</td>
</tr>
</tbody>
</table>

**Table 6.2:** Patch retrieval mAP comparison on RomePatches. SIFT is a strong baseline, previously only surpassed by the high-dimensional CKN-grad [124]. DOAP is the first descriptor to outperform SIFT with the same dimensionality.

Put dimensionality. Note that the closest competitor to DOAP, CKN-grad [124], is unsupervised and needs high dimensionality to perform well. By exploiting supervised learning and directly optimizing the evaluation metric, we are able to get better training and test performance while using 8x fewer dimensions (128 vs. 1024).

### 6.4.4 Oxford Dataset

Lastly, we use our learned descriptors to perform image matching in six image sequences from the classical Oxford dataset [113], where the matching pipeline also includes interest point detection. We use the implementation from VL-Benchmarks [95]; features are detected by the Harris-Affine detector, and then patches are extracted with a magnification factor of 3 relative to the detected feature frames. The evaluation metric is mean Average Precision (mAP), computed as the area under the precision-recall curve derived from nearest neighbor matching. We compare to SIFT, LIOP [167] (the best-performing handcrafted descriptor in the experiments reported by [151]), and 128-dimensional real-valued versions of L2Net and HardNet with different training sets. We use the 256-bit binary and 128-dimensional versions of DOAP-ST trained on Liberty, as well as the 128-dimensional DOAP-ST-LM trained on HPatches.
Figure 6.7: Image matching performance on the Oxford dataset [113]. Suffixes indicate the training set used (Lib: Liberty, HP: HPatches). Here, all versions of DOAP include the Spatial Transformer.

From the results in Fig. 6.7, we can see that SIFT is indeed difficult to beat, and good results on the UBC benchmark does not guarantee equally good high-level task performance, especially in the case of HardNet. The real-valued DOAP consistently outperforms SIFT and other descriptors with significant margins, especially in the more challenging sequences such as graf and boat. The binary DOAP trained on Liberty also outperforms other real-valued descriptors on average, including L2Net trained on HPatches, and HardNet trained on the union of Liberty and HPatches.

6.5 Implementation Details and Discussions

We normalize the input patches as follows: first subtract the mean pixel value in the patch, and then divide by the standard deviation. We train our networks from scratch using SGD with momentum 0.9 and weight decay $10^{-4}$. The initialization scheme proposed in [60] is adopted, since the architecture uses ReLU activations. Through validation experiments, we found that an initial learning rate of 0.1 works well with batch size $M = 1024$ in all datasets. For other batch sizes, we scale the learning rate linearly, according to the suggestion in [56].

For UBC Phototour, inspired by HardNet [115], the learning rate is decreased
linearly to zero within 100 epochs. For HPatches, we actually found a more traditional strategy to work better: we use a constant learning rate and divide it by 5 every 10 epochs, for 32 epochs total. Finally, due to the small size of RomePatches, we found it necessary to increase weight decay in SGD to $5 \times 10^{-4}$, and Dropout rate from 0.1 to 0.5 in the L2Net architecture.

Our implementation uses MatConvNet \cite{160}. For competing methods, we use the publicly released models/implementations.

- We use pretrained L2Net models\footnote{https://github.com/yuruntian/L2-Net} with data augmentation.
- We use pretrained HardNet models\footnote{https://github.com/DagnyT/hardnet} with data augmentation.
- For SIFT and LIOP, we use the implementation in VLFeat \cite{158}.

Performance on HPatches is evaluated using the HPatches benchmark\footnote{https://github.com/hpatches/hpatches-benchmark}. For the image matching experiment in Oxford dataset, the detection of interest points and extraction of patches are performed using the \texttt{vl\_covdet} function in VLFeat, with the \texttt{PatchRelativeExtent} parameter set to 3.

### 6.5.1 Minibatch Sampling

We discuss the minibatch sampling strategy used in training our models. First, note that in all datasets considered, patches are provided in groups: patches within a group correspond to the same 3D point and thus match each other (see Fig.~6.4). The group size, denoted $n$, is between 2 and 3 on average in UBC Phototour, and equals 10 in RomePatches. For HPatches, $n = 16$, as each patch has a reference version, and five variations from each difficulty level.

Our sampling strategy differs from those in local ranking approaches, where patch groups are often broken up to form pairs or triplets in a pre-processing step before
training. Instead, we directly sample groups to construct training minibatches, so that patches belonging to the same group are always in the same batch. This allows our listwise optimization to utilize supervision with maximum efficiency. Let minibatch size be $M$, every training patch is associated with a listwise ranking constraint, that its $n-1$ matches need to be ranked at the top of a list of size $M-1$. This constraint alone needs $(n-1)(M-n)$ triplets to fully capture. Take UBC Phototour as an example, assuming $n=2.5$ on average, a single minibatch of size 1024 induces about $1.6 \times 10^6$ triplets, which is already $1/32$ of the total number of training triplets used in HardNet. For HPatches ($n=16$), this number would be $1.5 \times 10^7$. However, triplets do not need to be explicitly generated in our listwise optimization.

For RomePatches, the training set has 10,000 patches, or 1,000 groups of 10 patches, which is quite small. To stabilize the training, we increase the number of minibatches in each epoch to 1,000 as follows: the $k$-th batch first includes the $k$-th group, and then randomly samples other groups to fill the batch. With this strategy, each epoch processes the training set multiple times, and we found 5 epochs to be sufficient to ensure convergence.

For HPatches, there are 76 image sequences in the training set. Without label mining, we uniformly sample patch groups from all sequences to construct training minibatches, so on average only about $1/76$ of the patches in each minibatch are from the same sequence. In this case, even if the in-sequence distractor labels are known, their contribution to the gradients is limited. Therefore, we use a modified minibatch sampling strategy when label mining is in effect, so that more patches from the same sequence are placed in a minibatch. Specifically, to construct a minibatch, we first sample two image sequences. Then, an equal number of patch groups (each containing 16 matching patches) are sampled from each sequence. For example, if batch size $M = 1024 = 64 \times 16$, then 32 groups are sampled from each of the two
sequences. This way, for each patch, roughly half of its distractors are out-of-sequence in-sequence patches, and the other half are in-sequence, which are generally harder to distinguish. With this strategy, a minibatch involves a pair of sequences, and a training epoch loops over all the \(76 \times (76 - 1)/2 = 2850\) pairs. This simple heuristic gave about 6\% absolute improvement in image matching mAP in our experiments.

### 6.5.2 Time Complexity

For a minibatch of size \(M\), the pairwise distances between all examples are computed, and then binned into \(b\)-bin histograms. The time complexity is \(O(bM^2)\). The quadratic dependency on \(M\) is in fact optimal, due to distance computation.

There is also a trade-off involving the batch size \(M\). A larger batch size leads to longer lists and better performance, but slows training. Similar to the case of UBC Phototour (Fig. 6·5), performance saturation is also observed around \(M = 2048\) in HPatches and RomePatches.

### 6.6 Summary

In this chapter, we use deep neural networks to learn binary and real-valued local feature descriptors that optimize nearest neighbor matching performance. This is achieved through a listwise learning to rank formulation that directly optimizes Average Precision, building on and extending our developed formulations in the previous chapter. We further enhance our general-purpose formulation with task-specific components: handling geometric noise with the Spatial Transformer, and mining labels using clustering.

Our learned descriptors, named DOAP, achieve state-of-the-art performance in patch verification, patch retrieval, and image matching. In particular, it is the first deep learning method to fairly beat SIFT on two benchmarks: RomePatches [124]
and the Oxford dataset \cite{113}. Also, on the challenging HPatches benchmark \cite{9}, DOAP achieves a 17\% relative performance gain in image matching over previous state-of-the-art descriptors \cite{115, 151}. We believe that our contribution can serve as a stepping stone for the direct optimization of larger computer vision pipelines.
Chapter 7

Deep Metric Learning with FastAP

In this chapter, we study the deep metric learning problem. Having a good distance metric is a prerequisite in many pattern recognition tasks. To start with, distance metrics play a central role in the main application considered in this thesis: nearest neighbor retrieval. Another important application area is nearest neighbor classification, especially in extreme classification or few-shot learning \[37, 126, 152\]. In such scenarios, with a large number of classes and limited examples per class, it becomes difficult to learn parameterized classification models for each class, but a nearest neighbor scheme can still suffice. Other important application areas for distance metric learning include clustering and data visualization.

As we mentioned in Chapter 1 and Chapter 3, deep metric learning is essentially a vector embedding problem, instantiated with deep neural networks. As is generally the case with metric learning, deep metric learning algorithms are tasked with producing embeddings that capture \textit{semantic similarities} between examples. In practice, it is often the case that semantic similarities are derived from class labels, which are a common form of supervision. Binary relevance, \textit{i.e.} similar/dissimilar relationships, can be inferred simply from the agreement/disagreement of class labels. More fine-grained relevance models \textit{(e.g.} graded relevance) can also be defined by mapping pairs of class labels to non-binary relevance values; an example is defining the relevance between ImageNet class labels using WordNet similarity \[125\]. In this chapter, for simplicity we focus on the widely used binary relevance setting.
Our goal in this chapter is to learn a distance metric that, put in simple terms, can produce small within-class distances and large between-class distances. Our primary contribution is to suggest an optimization objective for deep metric learning, which captures this intuition. As illustrated in Figure 7.1, we can again formulate this problem as learning to rank: given any instance, we would like to ensure that other instances from the same class have the lowest distance to it, or in other words, they are ranked in front of all other instances. We can see that the notions of query and database naturally arise in this formulation, and that the ranking can be repeated for all possible query instances in a training set. Under the binary relevance assumption, we will again use Average Precision (AP) as the optimization objective. As is the case with our other solutions presented in previous chapters, using such a listwise ranking metric eliminates many heuristics, such as setting nontrivial threshold/margin parameters, and hard negative mining/sampling. Instead, we rely on two things to obtain good results: large minibatches (long lists) during stochastic optimization, and optimizing over all possible query-database combinations within minibatches.

In previous chapters, we have in fact studied the optimization of AP for both
binary and real-valued vector embeddings. In Chapter 5 we derived the closed-
form AP for binary embeddings, and in Chapter 6 we reduced the AP optimization
for real-valued embeddings to the binary case via distance quantization. It appears
that we could simply reuse the solution in Chapter 6 for the real-valued deep metric
learning problem. That is indeed true; however, we note that it is not a natural
solution: the closed-form AP derived in Chapter 5 is based on tie-awareness, but ties
do not naturally exist with real-valued distances. Another problem with the earlier
solution is that the closed-form expressions are very complex, which require certain
simplifications in actual implementation. In contrast, in this chapter we propose a
formulation to optimize AP specifically for real-valued embeddings. The formulation
uses much the same optimization machinery that we developed for previous solutions,
but has elegant simple expressions. In the remainder of this chapter, we will detail
the new formulation, discuss its connection to our previous formulations, and present
experimental results.

7.1 FastAP: Quantization-based AP Approximation

We now discuss our novel AP approximation technique for real-valued embeddings in
detail. As usual, we assume a query $x_q \in \mathcal{X}$ and a database $S \subset \mathcal{X}$, with $|S| = N$. Under the binary relevance setting, we will let the number of $x_q$’s neighbors in the
database be $N^+$. We learn an embedding function $\Psi : \mathcal{X} \rightarrow \mathbb{R}^m$ that maps inputs
to an $m$-dimensional Euclidean space, where the distance metric is denoted $d_\Psi$. To
perform retrieval, we rank the items in $S$ according to their distances to $x_q$, producing
a ranked list $(x_1, x_2, \ldots, x_N)$.

Next, we go down the ranked list to obtain the \textit{precision-recall curve}. First, given
the ranked list, we compute a set of precision-recall pairs as

$$\text{PR}(x_q) = \{ (\text{Prec}(i), \text{Rec}(i)), i = 0, \ldots, n \},$$

(7.1)

where \(\text{Prec}(i)\) and \(\text{Rec}(i)\) are the precision and recall evaluated at the \(i\)-th position in the ranking, respectively. We define \(\text{Prec}(0) = \text{Rec}(0) = 0\) for convenience, and it is easy to see that the range of \(\text{Rec}\) is \([0, 1]\). As we mentioned in Section 2.2, if we view both precision and recall as continuous values and view precision as a function of recall, then \(\text{AP}\) is the area under the precision-recall curve:

$$\text{AP} = \int_0^1 \text{Prec}(\text{Rec}) \, d\text{Rec}. \quad (7.2)$$

This is achieved by interpolating precision and recall according to the observed set \(\text{PR}(x_q)\). The interpolated precision-recall curve often has a distinct sawtooth-like shape, and we show an example in Figure 7.2.

To tractably evaluate \(\text{AP}\) according to this definition, we take the usual strategy in numerically evaluating continuous integrals: approximating by finite sums. Specifically, if we quantize the precision-recall curve exactly at positions 1 through \(N\) in the ranking, then the integral can be evaluated approximately using finite sum:

$$\text{AP} \approx \sum_{i=1}^{N} \text{Prec}(i)(\text{Rec}(i) - \text{Rec}(i - 1)). \quad (7.3)$$

A qualitative illustration of the quantization-based approximation is also shown in Figure 7.2.

A problem with this approximation is that to obtain the precision-recall curve, the ranked list first needs to be generated, which involves the sorting operation. As we argued in previous chapters, the non-differentiable nature of sorting is a major hurdle for gradient-based optimization. Instead, our main insight is that there exists
Figure 7.2: A typical precision-recall curve and its quantization-based approximation. We propose a quantization-based technique, FastAP, to approximate Average Precision as the area under the precision-recall curve. Note that differently from the illustration, the actual formulation of FastAP uses a change of variables to quantize the distance instead of the recall value.

an alternative quantization strategy for approximating AP, and it is based on representing precision and recall as functions of distance, rather than ranked items. After this change of variables, we can reduce the problem of AP approximation to one of distance quantization, which is familiar to us.

Formally, we note that the continuous precision-recall curve (as opposed to the finite set in (7.1)) can be defined as:

$$\mathcal{PR}(x_q) = \{(\text{Prec}(z), \text{Rec}(z)), z \in (-\infty, +\infty)\},$$

where $z$ denotes distance values between the query and items in $S$. We will describe how to define precision and recall in this case shortly. With this change of variables,
AP can be defined as:

$$AP = \int_{-\infty}^{+\infty} \text{Prec}(z) \, d\text{Rec}(z).$$  \hfill (7.5)

We next define some probabilistic quantities so as to evaluate (7.5). Let \( Z \) be the random variable corresponding to distances \( z \), and let \( S^+, S^- \subseteq S \) denote the sets of neighbors and non-neighbors to the query, respectively. Then, the distance distributions for \( S^+ \) and \( S^- \) are denoted as \( p_Z(z|S^+) \) and \( p_Z(z|S^-) \). We also need the priors \( P(S^+) \) and \( P(S^-) = 1 - P(S^+) \), which indicate the skewness of the database \( S \) with respect to the query. Finally, let \( F_Z(z) = P(Z < z) \) denote the cumulative distribution function (CDF) for \( Z \). For brevity, we will drop the subscript \( Z \) below.

Given these definitions, we can redefine precision and recall as follows:

$$\text{Prec}(z) = P(S^+|Z < z) = \frac{P(Z < z|S^+)P(S^+)}{P(Z < z)} = \frac{F(z|S^+)P(S^+)}{F(z)},$$  \hfill (7.6)

$$\text{Rec}(z) = P(Z < z | S^+) = F(z|S^+).$$  \hfill (7.7)

Substituting these terms in (7.5), we get:

$$AP = \int_{-\infty}^{+\infty} \text{Prec}(z) \, d\text{Rec}(z)$$  \hfill (7.8)

$$= \int_{-\infty}^{+\infty} P(S^+|Z < z) \, dP(Z < z|S^+)$$  \hfill (7.9)

$$= \int_{-\infty}^{+\infty} \frac{F(z|S^+)P(S^+)}{F(z)} p(z|S^+) \, dz,$$  \hfill (7.10)

where we have used the fact that \( dP(Z < z|S^+) = p(z|S^+) \, dz \), i.e. the derivative of the CDF is its corresponding PDF.

It is clear now that (7.10) can also be approximately evaluated using finite sums,
in a way analogous to that of (7.3). Specifically, we will quantize the distances into a finite set \( Z = \{ Z_1, Z_2, \ldots \} \). We name this approximation FastAP, and simplify it using histogram notation:

\[
\text{FastAP} = \sum_{z \in Z} \frac{F(z|S^+)P(S^+)}{F(z)} P(z|S^+) = \sum_{z \in Z} \frac{N_z^+ / N^+ \cdot N^+/N}{N_z/N} \cdot \frac{n_z^+}{N^+}
\]

\[= \frac{1}{N^+} \sum_{z \in Z} \frac{N_z^+ n_z^+}{N_z}, \tag{7.13}\]

where

\[n_z = \sum_{x \in S} \mathbf{1} \left[ |d_\psi(x_q, x) - z| = \min_{z' \in Z} |d_\psi(x_q, x) - z'| \right], \tag{7.14}\]

\[N_z = \sum_{z' \leq z} n_{z'}. \tag{7.15}\]

In other words, \( n_z \) is the count of items in the \( z \)-th bin in the quantized histogram, and \( N_z \) is the cumulative sum of the histogram. Similarly, \( n_z^+ \) counts the number of relevant items in a histogram bin, and \( N_z^+ \) is the corresponding cumulative sum.

The optimization of FastAP follows the same procedure as in previous chapters. First, we apply differentiable relaxation \([155]\) to the histogram binning operation (7.14) to get a continuous relaxation of (7.13). Then, we derive closed-form derivatives for the continuous relaxation, and plug them into our minibatch-based framework that is described in detail in Appendix A.
7.2 Experiments

7.2.1 Datasets and Setup

We consider the Stanford Online Products dataset, which is proposed in [147] for evaluating deep metric learning algorithms. It contains 120,053 images of 22,634 online products from eBay.com, where each product is annotated with a distinct class label. In this dataset, a binary relevance setup is defined by treating images of the same product instance as relevant, and non-relevant otherwise. Each product also belongs to one of 12 meta-classes, such as bicycle, chair, and toaster, but such information is not used in the binary relevance setup. Notably, this is a few-shot dataset: each distinct class has 5.3 images on average, making it hard to learn the corresponding concepts from limited training examples. Please see Figure 7·3 for example images from the dataset.

For evaluation, we also follow [147] and consider two types of metrics. The first
is a retrieval metric, recall rate at $k$ (R@$k$), defined as the percentage of queries having at least one correct neighbor retrieved in the first $k$ results, and reported for different values of $k$. The second metric, Normalized Mutual Information (NMI), is commonly used to measure the agreement between different clusterings of data \cite{110}. In this setup, the embedding vectors are first clustered using K-means, and the result is compared to the clustering induced by ground truth class labels, \textit{i.e.}, two images are in the same ground-truth cluster if and only if they have the same label. NMI achieves its maximum value of 1 if the two clusterings are identical. Note that, in order to reduce the effects of the randomness in K-means clustering, we run K-means multiple times with different random seeds and report the best resulting NMI.

We compare our method, FastAP, to a series of recent works on deep metric learning. These include:

- Lifted Structured Embedding (LiftStruct) \cite{147},
- Histogram Loss (Histogram) \cite{155},
- Facility Location (Clustering) \cite{146},
- Deep Spectral Clustering (Spectral) \cite{94},
- Hard-Aware Deeply Cascaded Embedding (HDC) \cite{178},
- Margin \cite{171},
- Proxy NCA \cite{119},
- Boosting Independent Embeddings Robustly (BIER) and its augmented version with adversarial loss (A-BIER) \cite{123}, and
- Attention-Based Ensemble (ABE) \cite{81}.
Most of these methods either exclusively use triplet-based local ranking losses, or use triplet losses in ensembles. The exceptions are Clustering [146] and Spectral [94] in which clustering objectives are optimized, as well as Histogram [155] that proposes a quadruplet-based loss that is less common in the literature. Some of these methods also propose novelties that focus on aspects other than the loss function: Proxy NCA [119] uses generated “proxy” examples to aid training, and ABE [81] employs an attention mechanism.

7.2.2 Implementation details

Following [147], we use 59,551 images from 11,318 classes for training, and the remaining 60,502 images from 11,316 classes for testing. The images are first scaled to 256x256, and then 224x224 crops are used as inputs to the embedding network. Random crops and random flipping are used at training time for data augmentation, and a single center crop is used at test time.

Similar to [171], we finetune a ResNet-50 architecture [61] pretrained on ImageNet, and replace the final softmax classification layer with a fully-connected layer to produce the embedding vectors with $L_2$ normalization. The new layer is randomly initialized, and its learning rate is amplified by 10 times compared to the pretrained layers. We experiment with different values of the embedding dimensionality: 128, 256, and 512. We use the Adam optimizer [82] with base learning rate $5 \times 10^{-5}$ for 128 dimensions, and $10^{-5}$ for 256 and 512 dimensions. For distance quantization, we uniformly divide the interval $[0, 2]$ into 25 histogram bins. All experiments are run on an NVIDIA Titan X Pascal GPU with 12GB memory.

We also use a semi-random minibatch sampling strategy when training FastAP models on the products dataset, similar to the group-based strategy discussed in Section 6.5.1 for training the DOAP model. First, a “group” in this case consists of all
the images of a unique product, usually no more than 10. Second, we found that randomly sampling unique products is also suboptimal, as it ignores valuable meta-class information. For example, differentiating a bicycle from a toaster is much easier than differentiating it from a different bicycle. Therefore, once we have a reasonable (but still suboptimal) model, if all product instances contained in a batch are from different meta-classes, they are essentially “easy negatives” for each other, which makes the gradients estimated from the batch uninformative. Motivated by this observation, we utilize meta-class information when sampling minibatches: specifically, to generate each batch, we first sample a pair of meta-classes, and then sample individual products from the selected meta-classes to fill the batch, with a 50/50 split. This ensures, for example, that a batch containing some bicycle instance always contains other bicycles, so as to prevent the task of differentiating between different product instances from becoming too trivial. Essentially, this heuristic is analogous to “hard negative mining”, but is based on domain knowledge of the meta-classes. For each pair of meta-classes, we set the number of sampled batches to 5, and with 12 meta-classes in the products dataset, this gives $12 \times (12 - 1)/2 \times 5 = 330$ batches per epoch. We observe that this semi-random group-based sampling strategy significantly outperforms purely random sampling.

### 7.2.3 Results

We present retrieval and clustering results on the Stanford Online Products dataset in Table 7.1. FastAP achieves strong performance in retrieval, obtaining the current best results in recall rate at 10, 100, and 1000. For recall rate at 1, FastAP ranks second after ABE-8 [81]. With regard to the clustering metric NMI, FastAP is also competitive with state-of-the-art. We further note that the performance of FastAP increases with the dimensionality of the embedding vectors, and even the 128-dimensional ver-
<table>
<thead>
<tr>
<th>Method</th>
<th>Dim.</th>
<th>Stanford Online Products</th>
<th></th>
<th></th>
<th></th>
<th>NMI</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>R@1</td>
<td>R@10</td>
<td>R@100</td>
<td>R@1000</td>
<td></td>
</tr>
<tr>
<td>LiftStruct</td>
<td>512</td>
<td>62.1</td>
<td>79.8</td>
<td>91.3</td>
<td>97.4</td>
<td>88.7</td>
</tr>
<tr>
<td>Histogram</td>
<td>512</td>
<td>63.9</td>
<td>81.7</td>
<td>92.2</td>
<td>97.7</td>
<td>–</td>
</tr>
<tr>
<td>Clustering</td>
<td>64</td>
<td>67.0</td>
<td>83.7</td>
<td>93.2</td>
<td>–</td>
<td>89.5</td>
</tr>
<tr>
<td>Spectral</td>
<td>512</td>
<td>67.6</td>
<td>83.7</td>
<td>93.3</td>
<td>–</td>
<td>89.4</td>
</tr>
<tr>
<td>HDC</td>
<td>512</td>
<td>69.5</td>
<td>84.4</td>
<td>92.8</td>
<td>97.7</td>
<td>–</td>
</tr>
<tr>
<td>Margin</td>
<td>128</td>
<td>72.7</td>
<td>86.2</td>
<td>93.8</td>
<td>98.0</td>
<td>90.7</td>
</tr>
<tr>
<td>Proxy NCA</td>
<td>64</td>
<td>73.7</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>90.6</td>
</tr>
<tr>
<td>BIER</td>
<td>512</td>
<td>72.7</td>
<td>86.5</td>
<td>94.0</td>
<td>98.0</td>
<td>–</td>
</tr>
<tr>
<td>A-BIER</td>
<td>512</td>
<td>74.2</td>
<td>86.9</td>
<td>94.8</td>
<td>98.2</td>
<td>–</td>
</tr>
<tr>
<td>ABE-8</td>
<td>512</td>
<td>76.3</td>
<td>88.4</td>
<td>94.8</td>
<td>98.2</td>
<td>–</td>
</tr>
<tr>
<td>FastAP</td>
<td>128</td>
<td>73.8</td>
<td>88.0</td>
<td>94.9</td>
<td>98.3</td>
<td>89.8</td>
</tr>
<tr>
<td>FastAP</td>
<td>256</td>
<td>74.6</td>
<td>88.3</td>
<td>95.1</td>
<td>98.4</td>
<td>89.4</td>
</tr>
<tr>
<td>FastAP</td>
<td>512</td>
<td>75.7</td>
<td><strong>89.1</strong></td>
<td><strong>95.4</strong></td>
<td><strong>98.5</strong></td>
<td>89.5</td>
</tr>
</tbody>
</table>

* Ensemble methods.

Table 7.1: Metric learning performance comparison on the Stanford Online Products dataset [147]. R@$k$(%) stands for recall rate at $k$, and NMI(%) stands for Normalized Mutual Information. FastAP learns embedding vectors with a ResNet-50 architecture [61], and achieves performance competitive with the state-of-the-art. **Bold** denotes first place and **underline** denotes second place.

In fact, in terms of retrieval performance, only A-BIER [123] and ABE-8 [81], both of which are 512-dimensional, can partially outperform the 128-dimensional FastAP; FastAP still performs better when longer ranked lists are considered, as indicated by R@100 and R@1000.

There is a fair comparison between the 128-dimensional FastAP and Margin [171], which is a leading triplet-based method equipped with a principled sampling strategy, and is also trained with a ResNet-50 architecture. FastAP achieves significantly better retrieval performance, which can be attributed to its listwise ranking optimization. Interestingly, this comes at the cost of reduced NMI, which suggests that there may exist certain discrepancies in optimizing for retrieval vs. for clustering.
Another interesting comparison is between FastAP and the ensemble methods, namely, HDC \cite{178}, BIER and A-BIER \cite{123}, and ABE-8 \cite{81}. These methods combine embedding vectors obtained either from different layers in the same network, or from different networks trained with different losses. These ensemble models can be very complex and demanding to train: for example, A-BIER ensembles three different learners with a different loss in each, and applies an additional adversarial loss, and ABE-8 is an ensemble of 8 different learners, trained on a GPU with 24GB memory. In contrast, FastAP trains a single network to produce embedding vectors at the last layer, uses a single loss function, and uses 12GB of GPU memory. Therefore, FastAP achieves a much better complexity-performance trade-off.

The differentiable approximation to histogram binning, which has been a key component in our formulations, is initially proposed by Usnitova and Lempitsky \cite{155} for deep metric learning. Therefore, it would be informative to contrast FastAP with \cite{155}. The histogram loss proposed in \cite{155} is a quadruplet-based loss function that encourages the distance distributions of neighbors and non-neighbors to be separated, similar in spirit to the mutual information objective in MIHash (Chapter 4). However, this loss is only loosely correlated with ranking-based retrieval metrics, and we suspect that designing appropriate sampling strategies for quadruplets is even more challenging than for triplets. In Table \ref{tab:7.1} we can see that by changing the loss function and the accompanying minibatch sampling strategies, FastAP strongly outperforms the histogram loss.

Lastly, we present an ablation study where we train the 512-dimensional FastAP with different batch sizes. As we also explained in previous chapters, a batch size of $M$ means that the list size in our listwise optimization is $M - 1$ (minus 1 for the query). Here, we again want to test the hypothesis that with listwise ranking, longer lists lead to better results. With a ResNet-50 architecture and 12GB of GPU memory,
<table>
<thead>
<tr>
<th>Method</th>
<th>$M$</th>
<th>Stanford Online Products</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>$R@1$</td>
</tr>
<tr>
<td>FastAP</td>
<td>32</td>
<td>72.8</td>
</tr>
<tr>
<td>FastAP</td>
<td>64</td>
<td>74.9</td>
</tr>
<tr>
<td>FastAP</td>
<td>96</td>
<td>75.7</td>
</tr>
</tbody>
</table>

Table 7.2: Ablation study on the batch size $M$. We show results for 512-dimensional FastAP with different batch sizes on the products dataset. With ResNet-50, the maximum allowable value of $M$ is 96 on a 12GB GPU. As predicted, performance increases along with $M$.

the maximum $M$ that we could fit onto the GPU is 96, therefore $M = 96$ is used in all experiments reported earlier. In Table 7.2, we show the results when $M$ is reduced to 64 and 32, respectively, while all other parameters are kept fixed. As expected, with each reduction in $M$ there is a visible drop in overall performance, but even with $M = 32$ results are still fairly robust.

7.3 Discussions

7.3.1 Connection Between FastAP and Tie-Aware AP

By now, we have derived two principled closed-form formulations of AP in this thesis: the tie-aware $AP_T$ for binary embeddings (Chapter 5), and FastAP for real-valued embeddings. Both can be approximately optimized by taking advantage of differentiable histogram binning. Intuitively, these two versions of AP should be closely related, as they are different instantiations/approximations of the same quantity. Indeed, here we mathematically show that despite the apparent differences in their expressions, FastAP and $AP_T$ are closely connected, and equivalent in certain cases.

Recall from (5.16) that $AP_T$ can be written as

$$AP_T = \sum_{d=0}^{b} \frac{n_d^+}{N + n_d} \sum_{j=1}^{n_d} \frac{N_{d-1}^+ + 1 + (j - 1) \frac{n_{d-1}^+ - 1}{n_{d-1}}} {N_{d-1} + j},$$

(7.16)
where \( d \) indexes the Hamming distance from 0 to \( b \). In the real-valued case, as we did earlier, we will instead use variable \( z \) to index a set \( Z \) of quantized distances. So now we rewrite \( \text{AP}_T \) as

\[
\text{AP}_T = \sum_{z=1}^{\mid Z \mid} \frac{N^+_{z-1} + 1 + (j - 1)\frac{n^z_{z-1}}{n^z_{z-1}}}{N_{z-1} + j} \sum_{j=1}^{n^z} \frac{n^z_j}{N^+ + n^z} \sum_{j=1}^{n^z} N^+_{z-1} + 1 + (j - 1)\frac{n^z_{z-1}}{n^z_{z-1}}.
\]  

(7.17)

To connect this expression with that of FastAP, the key modification is to replace the second sum by repeating its last summand, i.e., we replace all occurrences of \( j \) with \( n^z_z \):

\[
\text{AP}_T = \sum_{z=1}^{\mid Z \mid} \frac{n^+_{z}}{N^+ + n^z} \sum_{j=1}^{n^z} \frac{N^+_{z-1} + 1 + (n^z - 1)\frac{n^z_{z-1}}{n^z_{z-1}}}{N_{z-1} + n^z}.
\]  

(7.18)

\[
= \frac{1}{N^+} \sum_{z=1}^{\mid Z \mid} \frac{n^+_{z}}{n^z} \sum_{j=1}^{n^z} \frac{N^+_{z}}{N_{z}}.
\]  

(7.19)

\[
= \frac{1}{N^+} \sum_{z=1}^{\mid Z \mid} \frac{n^+_{z}N^+_{z}}{N_{z}},
\]  

(7.20)

and we can see that (7.20) is exactly the same as FastAP (7.13).

From the above substitution, it is clear that FastAP and \( \text{AP}_T \) are equivalent (with respect to the quantization \( Z \)) when \( n^z_z \in \{0, 1\}, \forall z \). That is to say, if the quantization makes it so that every histogram bin has at most one element, then the two are exactly equal. The difference increases with coarser quantization. FastAP could be viewed as another simplification of \( \text{AP}_T \) that has a particularly simple expression.

### 7.3.2 Graded Relevance Extension

The FastAP formulation that we develop in this chapter focuses on learning to rank with binary relevance. Of course, the binary relevance assumption is restrictive, and a more complete formulation would need to address more fine-grained relevance models, such as graded (multi-level) relevance. For example, the Stanford Online Products
dataset in fact already provides a natural testbed for a two-level relevance setup, where a weaker relevance grade can be defined between different product instances within the same meta-class.

For graded relevance, a straightforward idea would be to use NDCG as the learning objective, as we have done in TALR for supervised hashing (Chapter 5). However, whereas FastAP relies on the probabilistic interpretation of AP as the area under the precision-recall curve, NDCG does not have a similar probabilistic interpretation, which makes it nontrivial to directly apply quantization-based approximation. Nevertheless, alternatives to NDCG do exist. In particular, we believe that a promising avenue is to explore the optimization of the Graded AP (GAP) proposed by Robertson et al. [131], which is a principled extension of AP in the graded relevance setting. The basic idea behind GAP is to convert graded relevance back to binary by thresholding on the relevance grades, and then taking the expectation of the resulting AP values over a user-defined distribution over the thresholds. Crucially, GAP also has a probabilistic interpretation as the area under the graded precision-recall curve, which would allow us to apply quantization-based approximation. Then, the FastAP formulation can be easily modified to optimize GAP. We plan to conduct such experiments in future work.

7.4 Summary

We tackle the deep metric learning problem in this chapter, and propose a learning to rank solution based on optimizing Average Precision. Despite being motivated by nearest neighbor retrieval, this formulation captures intuitions that are also crucial for nearest neighbor classification and clustering. We propose FastAP, a novel quantization-based approximation of AP, based on its probabilistic interpretation as the area under the precision-recall curve. We perform a change of variables to enable
the quantization to be expressed in terms of distances, and then apply our histogram-based framework to optimize the approximation. FastAP is closely connected to the tie-aware AP that we previously derived for Hamming distances, and it can be potentially extended to handle graded relevance.

Experimentally, FastAP is tested on a few-shot image dataset, and outperforms many competing deep metric learning methods that use higher-dimensional embeddings or more complex models.
Chapter 8

Conclusions

In this chapter we recap the contributions made in this thesis, discuss their strengths and limitations, and propose interesting directions for future research.

8.1 Main Contributions

The central contribution of this thesis is a gradient-based optimization framework, which produces embedding vectors that optimize listwise ranking evaluation metrics in nearest neighbor retrieval. In particular, this framework is able to optimize Average Precision (AP) and Normalized Discounted Cumulative Gain (NDCG), two of the most important retrieval performance metrics, providing a new solution to a long-standing problem in Information Retrieval. Our solution falls into the listwise ranking category \cite{26, 172} in Learning to Rank terminology \cite{105}, which are superior to pointwise and pairwise approaches in general. Regarding optimization, the use of differentiable histogram binning \cite{155} can be seen as a smoothing technique, which is key in circumventing the non-differentiable sorting operation required to compute listwise ranking metrics.

For the supervised hashing problem, our two proposed solutions are driven by different insights. In MIHash \cite{62, 64}, our motivation is to quantify and minimize neighborhood ambiguity in the learned Hamming space, using an information-theoretic approach. In TALR \cite{63}, we instead focus on the issue of ties which is universal in hashing, and anchor the discussion on tie-aware ranking metrics. To achieve end-
to-end optimization, both solutions employ a continuous relaxation component to transform the original NP-hard discrete optimization problems into their continuous counterparts.

For learning local feature descriptors, our first contribution is to identify local feature matching as a nearest neighbor retrieval problem. This allows us to reuse the previously developed AP optimization technique for the supervised hashing problem to learn binary descriptors. Additionally, we extend the formulation to also learn real-valued descriptors via distance quantization. We also note that the label mining technique described in Section 6.3 could be of independent interest, as it is not tied to a specific learning framework or loss function, but addresses the issue of missing supervision. The problem of missing supervision is common in semi-supervised and weakly-supervised settings that are far more realistic in practice, compared to the fully-supervised setting that we consider in this thesis. We believe that the insights gathered from solving the fully-supervised problems can inspire better solutions for semi-supervised and weakly-supervised problems in future work.

For deep metric learning, our quantization-based approximation of AP, named FastAP, gives a principled way to optimize this important non-decomposable objective. Furthermore, as we discussed, a straightforward extension of FastAP to handle graded relevance should be possible by exploiting Graded AP [131]. The histogram-based approximation can be seen as the “counting sort flavored” counterpart to the “quicksort flavored” technique by Mohapatra et al. [116, 117] for optimizing ranking-based loss functions. The main differences are twofold: first, Mohapatra et al. focus on learning the scoring functions in learning to rank, whereas we learn vector embeddings and implicitly induce scoring functions through distance metrics. Secondly, Mohapatra et al. rely on the structured prediction framework and loss-augmented inference to optimize bounds on the true objectives, whereas we take a direct relaxation
approach through “smoothing” the sorting operation.

8.2 Discussion of Limitations

Supervision and Modality

Throughout this thesis we have focused on fully supervised learning problems. Specifically, we assume that there is a relevance oracle that can return the relevance between any two examples in the training set, where the relevance is often derived from class labels. The objectives that we optimize, e.g. mutual information, AP, and NDCG, are only defined after the relevance values are given. Of course, this can be restrictive: in practical scenarios, it is often the case that not all examples are labeled, or that the available labels do not carry all the desired information. For example, it could be that relevance values for some example pairs are missing, or that in a graded relevance setting, some pairs are only marked as “similar” instead of with the actual relevance grade. The former case is known as semi-supervised learning, and the latter is often referred to as weakly-supervised learning. Our current solutions are not designed for these scenarios – they assume that supervision is fully available, and that the labels are accurate and informative.

Another limitation lies in the modalities considered, currently only consisting of images or local image patches. Correspondingly, we have only used convolutional neural networks to learn vector embeddings, as they are the dominant model architecture for images. Nevertheless, the learning formulations that we present are generic, as they are derived from first principles in Information Retrieval. Thus, it is possible to apply our formulations to applications in other modalities and/or other gradient-based model architectures.
An interesting observation we have made in the deep metric learning experiments is that listwise ranking is not always superior to triplet-based methods. Specifically, although our technique can faithfully optimize listwise ranking metrics, the histogram binning approximation seems to sacrifice some precision, especially near the top of the ranking, *e.g.* as measured by recall rate at 1. We believe that the current uniform binning strategy does not take the top-heavy nature of AP/NDCG into account, which could result in “over-smoothing” of the metrics near the top. Although the quantization strategy could be further optimized or even learned (*e.g.* using techniques similar to NetVLAD [5]), the benefits are currently unclear. On the other hand, carefully tuned pairwise ranking methods based on triplet supervision tend to perform well at optimizing the top of the ranking.

On a higher level, our observations seem to reflect a long-standing debate between local (pointwise/pairwise) and listwise ranking approaches in the LTR literature, where it is argued that although listwise ranking enjoys certain advantages, its optimization is usually more difficult, and local ranking methods can be preferred due to their simplicity [129]. Indeed, we observe that listwise methods are harder to derive mathematically and implement, whereas triplet-based and pair-based methods enjoy the practical advantage of being simpler to understand and implement, despite the need for more optimization heuristics and careful tuning. Another concern, unique to the stochastic optimization setting, is that approximating non-decomposable objectives such as AP/NDCG using minibatch statistics still introduces a mismatch between training and testing, as the list size at training time is inevitably constrained by available memory. As we have demonstrated experimentally, this can be mitigated by using large enough minibatches, but could still be a problem when using larger or deeper models.
Learning Local Features

Specific to the local feature learning problem, the main limitation in our DOAP model is that it only addresses the stage of feature descriptor extraction from image patches. Two other immediately related stages are interest point detection and orientation/affine shape estimation, which are built into the standard SIFT pipeline. It is reasonable to expect that all three could be unified into a single, end-to-end trainable network. Another limitation is the patch-based nature of the DOAP model, which means that its power is limited by the patch extraction process and any pre-processing therein. For image matching and related tasks, an image-based solution using fully convolutional networks, similar to Universal Correspondence Network [34], could be more desirable.

Another limitation, as seen from our experimental results in Section 6.4 (especially on the HPatches dataset), is the relative lack of robustness against illumination changes – in fact, all the tested descriptors based on deep learning perform significantly worse under illumination changes compared to viewpoint changes. In our model, illumination variations are heuristically removed during the input normalization stage, as we normalize each input patch by subtracting its own mean and dividing by its own standard deviation. Therefore, the only thing that the network learns to handle is the geometric variation. The network underperformed when we tried to let it also learn illumination invariance. Although this limitation could certainly be mitigated by having more training data of illumination variation patterns, it is also necessary to address the illumination challenge architecture-wise. Various topics related to illumination have been extensively studied in the computer vision literature. For example, it is possible to estimate parameterized models of illumination from single or multiple images [70, 92], or perform intrinsic image decomposition [74] to obtain shading maps. Perhaps a more closely related problem is achieving illum-
nation invariance in tracking [50], object detection [42, 47], etc. However, whereas geometric variations can be effectively estimated, and to a large extent removed, by the Spatial Transformer, it appears that more research is needed to find the exact counterpart for handling illumination variations in a deep learning context.

8.3 Directions for Future Research

The above discussions of current limitations lead us to consider future directions of research. Here we discuss three of them.

Hybrid Objectives in Learning to Rank

Based on our earlier discussion on the limitations of listwise ranking, we believe that a promising research direction is to combine pairwise and listwise approaches into a single learning to rank framework. In doing so, we aim to take the best from both worlds: the ability to optimize listwise ranking metrics globally, and fine-grained control of the top portion of the ranking through appropriate sampling of local triplets. In the learning to rank literature, similar ideas are explored to some extent [118, 129], but not in a deep learning context. Interestingly, for the deep learning of local feature descriptors, Kumar et al. [89] and Zhang et al. [185] add global loss or regularization terms to triplet-based losses, which seem to be motivated by the same observations. An advantage of our listwise ranking framework is that it does not put strong constraints on the minibatch construction during stochastic optimization (purely random in hashing, semi-random in learning local feature descriptors and deep metric learning), so that triplet-based losses could be easily added, e.g. using in-batch mining techniques [115]. Another interesting question is whether mining or sampling techniques, which are essential for the success of triplet-based pairwise ranking methods, can also help listwise ranking optimization. We plan to investigate
these questions in the future.

In general, as we have mentioned earlier, metric learning is ultimately a task-driven principle. Therefore, it may also be interesting to depart from the Learning to Rank paradigm which is primarily inspired by nearest neighbor retrieval, and consider objectives geared towards other tasks, such as the clustering objective Normalized Mutual Information (NMI) adopted in [146], and the spectral relaxation of the K-means objective in [94]. Even within the realms of nearest neighbor retrieval, an important next step for the metric learning community is to combine metric learning and subsequent hashing/quantization in an end-to-end trainable framework for efficient large-scale similarity search, as recently explored by [75, 135]. Although the focus of this thesis is nearest neighbor retrieval and learning to rank, we note that the optimization objectives for metric learning, as well as forms of supervision, should ideally be selected according to the task.

**Optimizing Computer Vision Pipelines**

We have discussed that a limitation of our work in learning local feature descriptors is that it does not address the entire matching pipeline. End-to-end optimization of low-level computer vision pipelines using deep learning is an attractive alternative to handcrafted ones that are still widespread in practice. A recent attempt is LIFT [176], which separately learns three components in the feature matching pipeline: interest point detection, orientation estimation, and descriptor extraction. Another example is DSAC [16], which achieves approximate end-to-end optimization of the camera localization pipeline using a differentiable approximation to RANSAC [49]. We believe that this is a very promising direction for future research.

For the specific problem of unifying interest point detection and descriptor extraction, a possibility is to adopt the Faster R-CNN [128] framework that has been highly
successful in object detection. Faster R-CNN has two main components: a region proposal network (RPN) to generate class-agnostic object proposals, and a classification network to classify object proposals into object categories. In essence, for the local feature problem, the RPN should produce “interest point proposals” instead of object proposals, and the classification network should be repurposed to extract feature descriptors. In fact, a similar model architecture is recently explored in [53], but only for 3D depth map data. A key difference from object detection is that the interest point RPN is essentially underconstrained, since there are no ground truth interest point annotations. In this case, a possibility is to add auxiliary loss functions to enforce desired properties, such as ranking constraints \[138, 183\], or covariant properties \[96\].

However, there are also pros and cons when it comes to end-to-end optimization. An immediate concern is that when an entire pipeline is being learned jointly, errors might become more difficult to locate and correct when different components in the pipeline can learn to adapt with each other, or even exploit regularities in each other’s output. Also, for many practical tasks involving large pipelines, it is entirely possible for end-to-end optimization to overfit and underperform, due to insufficient training data for the large number of learnable parameters. A good such example is recently discussed in a follow-up work to DSAC by Brachmann and Rother \[17\], who show that for camera localization, learning a single scene coordinate regression component in a differentiable pipeline in fact outperforms the end-to-end optimized DSAC, and is more stable. Therefore, optimization of computer vision pipelines is problem-dependent and data-dependent, and care must be taken in arriving at good design choices.
Cross-Modal Retrieval and Early Fusion

Finally, we note that this thesis has exclusively focused on applications in the image domain. However, nearest neighbor retrieval is often needed in cross-modal tasks, such as sketch-based image retrieval [46], retrieving objects in images using natural language [71], and retrieving clips from everyday videos using natural language [52, 66]. We are also interested in extending our learning to rank formulations to cross-modal retrieval tasks.

Most existing cross-modal retrieval approaches are based on projecting data from different modalities into the same vector embedding space. Then, ranking losses from uni-modal retrieval tasks can be applied, in a way that is largely agnostic to the specifics of the embedding model architectures. An example can be seen in [28]. Although this is a quite successful approach, it can be viewed as late fusion in the sense that information across modalities are not shared until the final ranking loss layer. However, in problems involving complex modalities such as untrimmed videos or natural language sentences, it might be beneficial to move beyond vector embeddings and use early fusion to fuse information at an earlier stage, which helps to both reduce the search space and improve the quality of the learned similarity metric.

In fact, our concurrent work [175] explores the use of early fusion in the text-to-clip retrieval task [52, 66], and demonstrates convincing improvements over late fusion (vector embedding) approaches. Specifically, it conditions the processing of the query sentence on a visual embedding vector, in a novel Long Short-Term Memory (LSTM) model [69]. Interestingly, this contrasts with the approach of de Vries et al. [43], which modulates the processing of visual information in a feed-forward convolutional network using a language embedding vector. We believe that future improvements for this and related tasks will come from an earlier and deeper level of fusion between vision and language.
Appendix A

Efficient Minibatch Backpropagation

We detail the derivation of the minibatch backpropagation rules, which is used in all of our proposed solutions. As mentioned earlier, our models are trained using minibatch SGD. To maximally utilize supervision, we use the following strategy: each example in the minibatch is used to query the rest of the batch (which acts as the database), and the resulting objective values are averaged. We give a general derivation that naturally handles both binary and multi-level/graded relevance.

Consider a minibatch of size $M$, $\{x_1, \ldots, x_M\}$. We use a unified shorthand $O_i$ to denote the (relaxed) objective value when $x_i$ is the query, which can either be AP$_r$ or DCG$_r$ in our formulation. The overall minibatch objective is then $O = \frac{1}{M} \sum_{i=1}^{M} O_i$.

For the entire minibatch, we group the relaxed hash mapping output into a $b \times M$ matrix,

$$
\hat{\Phi} = \begin{bmatrix}
\hat{\Phi}(x_1) & \hat{\Phi}(x_2) & \cdots & \hat{\Phi}(x_M)
\end{bmatrix} \in \mathbb{R}^{b \times M}.
$$

We consider the multi-level affinity setup where affinity values are from a finite set $\mathcal{V}$, which includes binary affinities as a special case, i.e. when $\mathcal{V} = \{0, 1\}$. The Jacobian of the minibatch objective with respect to $\hat{\Phi}$ can be written as

$$
\frac{\partial O}{\partial \hat{\Phi}} = \frac{1}{M} \sum_{i=1}^{M} \frac{\partial O_i}{\partial \hat{\Phi}} = \frac{1}{M} \sum_{i=1}^{M} \sum_{d=0}^{b} \sum_{v \in \mathcal{V}} \frac{\partial O_i}{\partial c_{d,v}^{(i)}} \frac{\partial c_{d,v}^{(i)}}{\partial \hat{\Phi}},
$$

where as defined earlier, $c_{d,v}^{(i)}$ is the continuous relaxation of $n_{d,v}^{(i)}$, the $d$-th bin in the
distance histogram conditioned on affinity level \( v \). The superscript \( (i) \) indicates that the current query is \( x_i \).

Evaluating this Jacobian involves two steps. First, we need to compute the partial derivative \( \frac{\partial \mathcal{O}_i}{\partial c_{d,v}^{(i)}} \), \( \forall d, \forall v \). For MIHash, the derivatives of Mutual Information are given in \[4.15\]. For TALR, this is exactly the differentiation of \( \text{AP}_r \) and \( \text{DCG}_r \), and as we pointed out in Section \[5.2\] both can be evaluated in closed form. Collectively, we use variables \( \alpha \) to denote these partial derivatives, noting that the exact instantiation is different for each objective function:

\[
\alpha_{d,v}^{(i)} = \frac{\partial \mathcal{O}_i}{\partial c_{d,v}^{(i)}}. \tag{A.3}
\]

Next, we need to evaluate the Jacobian \( \frac{\partial c_{d,v}^{(i)}}{\partial \hat{\Phi}} \), which is essentially differentiating the soft histogram binning process. Let us consider each column of this Jacobian. First, for \( \forall j \neq i \), using chain rule,

\[
\frac{\partial c_{d,v}^{(i)}}{\partial \Phi(x_j)} = \frac{\partial c_{d,v}^{(i)}}{\partial \hat{d}_\Phi(x_i, x_j)} \frac{\partial \hat{d}_\Phi(x_i, x_j)}{\partial \Phi(x_j)}
= 1[A_i(j) = v] \frac{\partial \delta(\hat{d}_\Phi(x_i, x_j), d)}{\partial \hat{d}_\Phi(x_i, x_j)} \frac{\partial \hat{d}_\Phi(x_i, x_j)}{\partial \Phi(x_j)}
= 1[A_i(j) = v] \delta'_d(\hat{d}_\Phi(x_i, x_j)) \frac{\hat{\Phi}(x_i)}{2}
\Delta = \beta_{d,v}(i, j) \frac{\hat{\Phi}(x_i)}{2}, \tag{A.7}
\]

where \( \delta'_d \) is the derivative of the single-argument function \( \delta(\cdot, d) \) \[4.21\], and we define the shorthand

\[
\beta_{d,v}(i, j) = 1[A_i(j) = v] \delta'_d(\hat{d}_\Phi(x_i, x_j)). \tag{A.8}
\]

Note that \( \beta \) is symmetric, \( i.e. \beta_{d,v}(i, j) = \beta_{d,v}(j, i) \), which follows from the fact that both the affinity \( \mathcal{A} \) and the distance function \( \hat{d}_\Phi \) are symmetric.
For \( j = i \), we have a similar result:

\[
\frac{\partial c_{d,v}(i)}{\partial \Phi(x_i)} = \sum_{k \neq i} \frac{\partial c_{d,v}(i)}{\partial d\Phi(x_i, x_k)} \frac{\partial d\Phi(x_i, x_k)}{\partial \Phi(x_i)} = \sum_{k \neq i} \beta_{d,v}(i, k) \frac{-\Phi(x_k)}{2}.
\]  

(A.9)

To unify these two cases, we require that \( \beta_{d,v}(i, i) \equiv 0, \forall i \). We now have a unified expression for the \( j \)-th column in the Jacobian \( \partial c_{d,v}^{(i)}/\partial \hat{\Phi} \):

\[
\frac{\partial c_{d,v}^{(i)}}{\partial \Phi(x_j)} = -\frac{1}{2} \frac{\Phi(x_i)}{2} \left[ \beta_{d,v}(i, j) \Phi_{x_i} \right] \mathbf{e}_i^\top - \frac{M}{2} \sum_{k=1}^{M} \beta_{d,v}(i, k) \Phi_{x_k} \mathbf{e}_i^\top.
\]  

(A.10)

We now obtain a compact matrix form for \( \partial c_{d,v}^{(i)}/\partial \Phi \). First we define \( M \)-vector \( \beta_{d,v}(i) = (\beta_{d,v}(i, 1), \ldots, \beta_{d,v}(i, M)) \in \mathbb{R}^M \). Also, let \( \mathbf{e}_i \) be the \( i \)-th standard basis vector in \( \mathbb{R}^M \), \( i.e. \) the \( i \)-th element is 1 and all others are 0. We have that

\[
\frac{\partial c_{d,v}^{(i)}}{\partial \Phi} = -\frac{1}{2} \Phi(x_i) (\beta_{d,v}^{(i)})^\top - \left[ \sum_{k=1}^{M} \frac{1}{2} \beta_{d,v}(i, k) \Phi(x_k) \right] \mathbf{e}_i^\top
\]  

(A.11)

\[
= -\frac{1}{2} \left[ \Phi(x_i) (\beta_{d,v}^{(i)})^\top + \hat{\Phi} \beta_{d,v}^{(i)} \mathbf{e}_i^\top \right].
\]  

(A.12)

Finally, we complete (A.2) using the result above. The main trick is to change the ordering of sums: we bring the sum over \( i = 1, \ldots, M \) inside,

\[
\frac{\partial \mathcal{O}}{\partial \Phi} = \frac{1}{M} \sum_{i=1}^{M} \mathbf{\alpha}_{d,v}^{(i)} \mathbf{\delta}_{d,v}^{(i)}
\]  

(A.13)

\[
= \sum_{i=1}^{M} \sum_{v \in \mathcal{V}} \left[ \sum_{l=0}^{b} \frac{\alpha_{d,v}^{(i)}}{2} \left( \frac{\Phi(x_i)}{2} \beta_{d,v}^{(i)} \right)^\top + \hat{\Phi} \beta_{d,v}^{(i)} \mathbf{e}_i^\top \right].
\]  

(A.14)

\[
= \sum_{i=1}^{M} \sum_{v \in \mathcal{V}} \left[ \sum_{l=0}^{b} \frac{\alpha_{d,v}^{(i)}}{2} \hat{\Phi}(x_i) (\beta_{d,v}^{(i)})^\top + \hat{\Phi} \sum_{l=1}^{M} \alpha_{d,v}^{(i)} \beta_{d,v}^{(i)} \mathbf{e}_i^\top \right].
\]  

(A.15)
To further simplify this result, we define two $M \times M$ matrices:

$$A_{d,v} = \text{diag}(\alpha_{d,v}^{(1)}, \ldots, \alpha_{d,v}^{(M)}) \in \mathbb{R}^{M \times M}, \quad (A.16)$$

$$B_{d,v} = \begin{bmatrix} \beta_{d,v}^{(1)} \cdots \beta_{d,v}^{(M)} \\ \vdots \\ \beta_{d,v}(1,1) \cdots \beta_{d,v}(M,1) \end{bmatrix} \in \mathbb{R}^{M \times M}. \quad (A.17)$$

Then, we arrive at the following simplification of (A.15) and (A.2),

$$\frac{\partial \mathcal{O}}{\partial \hat{\Phi}} = -\frac{1}{2M} \sum_{d=0}^b \sum_{v \in V} \left[ \hat{\Phi} A_{d,v} (B_{d,v})^\top + \hat{\Phi} B_{d,v} A_{d,v} \right] \quad (A.19)$$

$$= -\frac{\hat{\Phi}}{2M} \sum_{d=0}^b \sum_{v \in V} (A_{d,v} B_{d,v} + B_{d,v} A_{d,v}). \quad (A.20)$$

Note that we have used the fact that $B_{d,v}$ is a symmetric matrix (A.18), which is because $\beta$ is symmetric, as mentioned earlier. This operation can be implemented efficiently using only matrix multiplications and additions. Also, since $A_{d,v}$ is a diagonal matrix, multiplying it with $B_{d,v}$ essentially scales the rows or columns of $B_{d,v}$, which is an $O(M^2)$ operation as opposed to $O(M^3)$ as in general matrix multiplication. The entire time complexity is therefore $O(b|V|M^2)$.

At this point, we have completed the differentiation of the minibatch objective $\mathcal{O}$ with respect to the relaxed hash mapping output, $\hat{\Phi}$. Further backpropagation is straightforward, since $\hat{\Phi}$ is obtained by applying a pointwise tanh function on the raw activations from the previous layer. The derivative of the tanh function has a closed-form expression, and is omitted here.
Appendix B

Proof of Proposition 1

Proof. Our proof essentially restates the results in [112] using our notation. In [112], a tie-vector $T = (t_0, \ldots, t_{d+1})$ is defined, where $t_0 = 0$ and the next elements indicate the ending indices of the equivalence classes in the ranking, e.g. $t_1$ is the ending index of $R^{(0)}$, and so on. Using our notation, we can see that $R^{(d)} = (R_{1+t_d}, \ldots, R_{t_d+1})$, and $t_d = N_{d-1} = \sum_{j=0}^{d-1} n_j$.

We first consider $\text{AP}_T$. In Section 2.4 of [112], the tie-aware AP at cutoff $k$ is defined as

$$\text{AP}_T \@ k(R) = \frac{\sum_{j=1}^{k} n_i^+ \left( N_{i-1}^+ + (j - t_i - 1) \frac{n_i^+}{n_{i-1}} + 1 \right)}{\sum_{j=1}^{\left| S \right|} A_q(j)}, \quad (B.1)$$

where $i$ is the index of the tie that item $j$ is in. To derive $\text{AP}_T$ in our formulation, we take $k$ to be the maximum possible cutoff $\left| S \right|$, and substitute by definition $N^+ = \sum_{j=1}^{\left| S \right|} A_q(j)$, $t_i = N_{i-1}$:

$$\text{AP}_T(R) = \frac{1}{N^+} \sum_{j=1}^{\left| S \right|} n_i^+ \left( N_{i-1}^+ + (j - N_{i-1} - 1) \frac{n_i^+}{n_{i-1}} + 1 \right), \quad (B.2)$$

It is clear that this sum decomposes additively over $j$. Therefore, we can explicitly compute the contribution from items in a single tie $R^{(d)}$,

$$\text{AP}_T(R^{(d)}) = \frac{1}{N^+} \sum_{j=N_{d-1}+1}^{N_d} n_j^+ \left( N_{d-1}^+ + (j - N_{d-1} - 1) \frac{n_j^+}{n_{d-1}} + 1 \right), \quad (B.3)$$

and this gives (5.4).
Next, tie-aware DCG is given in Section 2.6 of [112] as

\[
\text{DCG}_{T@k}(R) = \sum_d \left[ \left( \frac{1}{n_d} \sum_{j=t_d+1}^{t_d+1} G(A_q(j)) \right)^{\min(t_d+1,k)} \sum_{j=t_d+1}^{\min(t_d+1,k)} D(j) \right]. \tag{B.4}
\]

Again, we consider a single tie \( R^{(d)} \), take \( k = |S| \), and make the substitution \( t_d = N_{d-1} \):

\[
\text{DCG}_{T}(R^{(d)}) = \frac{1}{n_d} \sum_{j \in R^{(d)}} G(A_q(j)) \sum_{j=N_{d-1}+1}^{N_d} D(j) \tag{B.5}
\]

\[
= \frac{1}{n_d} \sum_{v \in V} \sum_{j \in R^{(d)}} 1[v = A_q(j)] G(v) \sum_{j=N_{d-1}+1}^{N_d} D(j) \tag{B.6}
\]

\[
= \frac{1}{n_d} \sum_{v \in V} G(v) n_{d,v} \sum_{j=N_{d-1}+1}^{N_d} D(j). \tag{B.7}
\]

This completes the derivation for (5.5). \qed
Appendix C

Proof of Proposition 2

C.1 Proof

Proof. First, we denote the summand in the definition of $AP_T$ \((5.4)\) as $\beta_d(t)$, and rewrite it as

\[
\beta_d(t) = \frac{N_{d-1}^+ + (t - N_{d-1} - 1) \frac{n_d^+ - 1}{n_d - 1} + 1}{t} \tag{C.1}
\]

\[
= \frac{n_d^+ - 1}{n_d - 1} + \frac{N_{d-1}^+ + 1 - \frac{n_d^+ - 1}{n_d - 1} (N_{d-1} + 1)}{t} \tag{C.2}
\]

It is of the form $A + B/t$ where $A, B$ are constant in $t$. We proceed with the summation over $t$ in \((5.4)\):

\[
AP_T(R^{(d)}) = \frac{n_d^+}{n_d N^+_{d-1}} \sum_{t=N_{d-1}+1}^{N_d} \beta_d(t) \tag{C.3}
\]

\[
= \frac{n_d^+}{n_d N^+_{d-1}} \left[ \frac{n_d^+ - 1}{n_d - 1} n_d + \left( N_{d-1}^+ + 1 - \frac{n_d^+ - 1}{n_d - 1} (N_{d-1} + 1) \right) \sum_{t=N_{d-1}+1}^{N_d} \frac{1}{t} \right]. \tag{C.4}
\]

The main obstacle in the continuous relaxation is the finite sum in \((C.4)\), which has in its limits $N_{d-1}$ and $N_d$, variables to be relaxed. However, it is a partial sum of the harmonic series, which can be well approximated by differences of the natural logarithm:

\[
\sum_{t=N_{d-1}+1}^{N_d} \frac{1}{t} \approx \int_{N_{d-1}}^{N_d} \frac{dt}{t} = \ln(N_d) - \ln(N_{d-1}). \tag{C.5}
\]
In fact, (C.5) corresponds to the midpoint rule in approximating definite integrals by finite sums, but is applied in the reverse direction. The relaxation of $\mathrm{AP}_T$ is then derived as follows:

$$
\mathrm{AP}_T(R^{(d)}) \approx \frac{n_d^+}{n_d N^+} \left[ \frac{n_d^+ - 1}{n_d - 1} n_d + \left( N_{d-1}^+ + 1 - \frac{n_d^+ - 1}{n_d - 1} (N_{d-1} + 1) \right) \ln \frac{N_d}{N_{d-1}} \right].
$$

(C.6)

$$
\Rightarrow \quad \mathrm{AP}_r(R^{(d)}) = \frac{c_d^+ (c_d^+ - 1)}{(c_d - 1) N^+} + \frac{c_d^+}{c_d N^+} \left[ C_{d-1}^+ + 1 - \frac{c_d^+ - 1}{c_d - 1} (C_{d-1} + 1) \right] \ln \frac{C_d}{C_{d-1}}.
$$

(C.7)

Note that $N^+ = \sum_d n_d^+$ is a constant for a fixed query and fixed database, thus it is not affected by the relaxation.

Next, we consider $\mathrm{DCG}_T$, where the sum of logarithmic\footnote{Other types of discounts are also used in the literature, including linear discount: $D(t) \propto \frac{1}{t}$. It is easy to see that our technique also applies.} discount values similarly involves variables to be relaxed in its limits. Thus, the same approximation strategy using continuous integrals is used.

$$
\sum_{t=N_{d-1}+1}^{N_d} D(t) = \sum_{t=N_{d-1}+1}^{N_d} \frac{1}{\log_2(t+1)} \approx \int_{N_{d-1}}^{N_d} \frac{dt}{\log_2(t+1)} = \ln 2 \int_{N_{d-1}}^{N_d+1} \frac{dt}{\ln t}. 
$$

(C.8)

Combining with the definition in (5.5), we get the continuous relaxation of $\mathrm{DCG}_T$:

$$
\mathrm{DCG}_r(R^{(d)}) = \ln 2 \sum_{v \in V} \frac{G(v) c_{d,v}}{c_d} \int_{C_{d-1}+1}^{C_d+1} \frac{dt}{\ln t} 
$$

\[ = \ln 2 \sum_{v \in V} \frac{G(v) c_{d,v}}{c_d} \left[ \text{Li}(C_d + 1) - \text{Li}(C_{d-1} + 1) \right] \]

(C.9)

\(\text{where Li is the logarithmic integral function: } \text{Li}(x) = \int_0^x \frac{dx}{\ln x}.\)
C.2 Approximation Error Analysis

We now analyze the approximation error when doing the continuous relaxations. We take AP$_T$ as example, and note that the analysis for DCG$_T$ is similar.

The continuous relaxation for AP$_T$(R$^d$) is given in (C.5), which replaces a finite sum with a definite integral, where the finite sum has $N_d - N_{d-1} = n_d$ summands. First, we consider the case where there are no ties, or $n_d \in \{0, 1\}$, i.e. the $d$-th histogram bin is either empty or contains a single item. In this case, we can directly evaluate the lefthand side sum in (C.5) to be either 0 or $\frac{1}{N_d}$, without using the integral approximation. Therefore, when there are no ties, there is no approximation error.

Next we consider $n_d \geq 2$. Let the $N$-th harmonic number be $H(N) = \sum_{t=1}^{N} \frac{1}{t}$, then the lefthand side of (C.5) is exactly $H(N_d) - H(N_{d-1})$. It is well known that the harmonic number can be closely approximated as

$$H(N) = \gamma + \ln(N) + \frac{1}{2N} + O\left(\frac{1}{12N^2}\right),$$

where $\gamma \approx 0.5772$ is Euler's constant. A direct application of this approximation gives the following:

$$H(N_d) = \gamma + \ln(N_d) + \frac{1}{2N_d} + O\left(\frac{1}{12N_d^2}\right)$$

$$H(N_{d-1}) = \gamma + \ln(N_{d-1}) + \frac{1}{2N_{d-1}} + O\left(\frac{1}{12N_{d-1}^2}\right)$$

$$\Rightarrow H(N_d) - H(N_{d-1}) = \ln(N_d) - \ln(N_{d-1}) + O\left(\frac{1}{2N_{d-1}} - \frac{1}{2N_d}\right).$$

Comparing (C.14) with (C.5), we see that the approximation error is

$$O\left(\frac{1}{2N_{d-1}} - \frac{1}{2N_d}\right) = O\left(\frac{n_d}{2N_{d-1}N_d}\right) = O\left(\frac{n_d}{2N_{d-1}^2}\right).$$

The error is proportional to $n_d$, the number of items in the $d$-th bin in the Hamming
distance histogram. However, even if $n_d$ is large, the error is in general still small, since it has $N_d^{d-1}$ in the denominator. Note that (C.5) can be further tightened by including the $\frac{1}{2N}$ term, or even higher order terms in the approximation of Harmonic numbers, but the approximation using the first two terms (Euler’s constant and natural log) is already quite tight, and is in fact used widely.
References


CURRICULUM VITAE

Kun He

EDUCATION

**Ph.D. in Computer Science**, Boston University, Boston, MA 2013 - 2018
Thesis: *Learning Deep Embeddings by Learning to Rank*

**M.Sc. in Computer Science**, Boston University, Boston, MA 2013
Thesis: *Stochastic Functional Descent for Learning Support Vector Machines*

**B.Eng. in Computer Science**, Zhejiang University, China 2010
Thesis: *A Real-Time Feature Tracking System on Desktop Environment*

RESEARCH EXPERIENCE

**Boston University, Boston, MA** 2010 - Present
*Research Assistant*

- **Ph.D. thesis**: Using deep neural networks and “learning to rank” formulations to learn low-dimensional feature embeddings for visual data. This enables fast and accurate nearest-neighbor retrieval/matching in applications including: image search, low-level computer vision, and cross-modal retrieval between everyday video and natural language.

- **Object detection and pose estimation**: Developed a kernel-based structured prediction framework to jointly solve object detection and continuous pose estimation, using structural SVMs.

- **Weakly-supervised learning**: Worked on learning object detectors with weak supervision, and the associated non-convex optimization problems.

- **Image segmentation**: Collaborated on quantifying and predicting the inherent ambiguity in the image foreground segmentation task, and investigated its use in crowdsourcing applications.

**Brown University, Providence, RI** 01/2014 - 08/2014
*Visiting Student*

- Host: Prof. Pedro Felzenszwalb

- Collaborated on object detection with latent variable models, and non-convex optimization in the Majorization-Minimization framework. Developed a new optimization framework, named Generalized Majorization-Minimization, that improves upon the original.
Zhejiang University, Hangzhou, China 03/2009 - 06/2010

Undergraduate Research Assistant  Advisors: Dr. Guofeng Zhang and Prof. Hujun Bao

- Low-level computer vision research training, topics include stereo matching, optical flow estimation, and augmented reality. Completed Bachelor’s thesis on real-time augmented reality.

WORK EXPERIENCE

Honda Research Institute USA, Mountain View, CA  Summer 2017

Research Intern  Host: Dr. Yan Lu

- Worked on learning deep neural networks to extract local image features, for improving long-term self-localization in autonomous driving.

Disney Research, Pittsburgh, PA  Fall 2013, Summer 2015

Research Intern  Host: Dr. Leonid Sigal

- **Summer 2015:** Worked on weakly-supervised object recognition guided by natural language.
- **Fall 2013:** Developed a structured prediction method for object detection and continuous pose estimation. Also worked on improving visual speech synthesis with “dynamic visemes” and machine learning.

PUBLICATIONS

Preprints:

[P1] Hashing with Mutual Information  
  Kun He*, Fatih Cakir*, Sarah Adel Bargal, and Stan Sclaroff  
  (*equal contribution)  
  Tech report arXiv:1803.00974, 2018

[P2] Text-to-Clip Video Retrieval with Early Fusion and Re-Captioning  
  Huijuan Xu, Kun He, Leonid Sigal, Stan Sclaroff, and Kate Saenko  
  Tech report arXiv:1804.05113, 2018

Journal Publications:

[J1] Predicting Foreground Object Ambiguity and Efficiently Crowdsourcing the Segmentation(s)  
  Danna Gurari, Kun He, Bo Xiong, Jianming Zhang, Mehrnoosh Sameki, Suyog Dutt Jain, Stan Sclaroff, Margrit Betke, and Kristen Grauman  
  International Journal of Computer Vision (IJCV), 2018
Conferences Publications:

[C1] Hashing with Binary Matrix Pursuit
Fatih Cakir, Kun He, and Stan Sclaroff
European Conference on Computer Vision (ECCV), 2018 (to appear)

[C2] Local Descriptors Optimized for Average Precision
Kun He, Yan Lu, and Stan Sclaroff
IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2018

[C3] Hashing as Tie-Aware Learning to Rank
Kun He, Fatih Cakir, Sarah Adel Bargal, and Stan Sclaroff
IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2018

[C4] MIHash: Online Hashing with Mutual Information
Kun He*, Fatih Cakir*, Sarah Adel Bargal, and Stan Sclaroff
(*equal contribution)
IEEE International Conference on Computer Vision (ICCV), 2017

[C5] Parameterizing Object Detectors in the Continuous Pose Space
Kun He, Leonid Sigal, and Stan Sclaroff
European Conference on Computer Vision (ECCV), 2014

[C6] Scale Resilient, Rotation Invariant Articulated Object Matching
Hao Jiang, Tai-Peng Tian, Kun He, and Stan Sclaroff
IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2012

Others:

[O1] Generalized Majorization-Minimization
Sobhan Naderi Parizi, Kun He, Stan Sclaroff, and Pedro Felzenszwalb

Kun He

PROFESSIONAL ACTIVITIES AND SERVICES

Student member, IEEE

Journal reviewer:
- International Journal of Computer Vision
- IEEE Transactions on Pattern Recognition and Machine Intelligence
- IEEE Transactions on Multimedia
- Computer Vision and Image Understanding
· PLOS ONE
  Conference reviewer:
  · IEEE Conference on Computer Vision and Pattern Recognition (CVPR) 2012-13, 2015-16, 2018
  · IEEE International Conference on Computer Vision (ICCV) 2011
  · European Conference on Computer Vision (ECCV) 2016
  · Asian Conference on Computer Vision (ACCV) 2012, 2018
  · IEEE Winter Conference on Applications of Computer Vision (WACV) 2018-19
  · IEEE International Conference on Advanced Video and Signal-Based Surveillance (AVSS) 2012
  · IEEE Connected and Automated Vehicles Symposium (CAVS) 2018

INVITED PRESENTATIONS

Doctoral Consortium at IEEE CVPR 2018
New England Computer Vision Workshop, 2017
New England Computer Vision Workshop, 2016
Brown University Computer Vision Reading Group, 2014

HONORS AND AWARDS

Honorary graduate of the Chu Kochen Honors College, Zhejiang University
Outstanding Bachelor’s Thesis, School of Computer Science and Technology, Zhejiang University, 2010

TEACHING EXPERIENCE

At Boston University:
· Teaching Assistant, CS542 Machine Learning, Fall 2017
· Teaching Fellow, CS591-S2 Deep Learning, Spring 2017
· Teaching Fellow, CS542 Machine Learning, Spring 2016
· Teaching Fellow, CS542 Machine Learning, Spring 2015
· Teaching Assistant, CS542 Machine Learning, Spring 2013
· Teaching Fellow, CS103 Introduction to Internet Technologies and Web Programming, Spring 2012