Efficient visual search in images and videos

Tan, Yu

2019

Tan, Y. (2019). Efficient visual search in images and videos. Doctoral thesis, Nanyang Technological University, Singapore.

<https://hdl.handle.net/10356/83290>

<https://doi.org/10.32657/10356/83290>

EFFICIENT VISUAL SEARCH IN IMAGES AND VIDEOS

TAN YU

Interdisciplinary Graduate School

A thesis submitted to the Nanyang Technological University
in partial fulfillment of the requirement for the degree of
Doctor of Philosophy

2019

Statement of Originality

I hereby certify that the work embodied in this thesis is the result of original research, is free of plagiarised materials, and has not been submitted for a higher degree to any other University or Institution.

[Input Date Here]
2019-04-25
.....
Date

[Input Signature Here]

.....
[Input Name Here]
Tan Yu

Supervisor Declaration Statement

I have reviewed the content and presentation style of this thesis and declare it is free of plagiarism and of sufficient grammatical clarity to be examined. To the best of my knowledge, the research and writing are those of the candidate except as acknowledged in the Author Attribution Statement. I confirm that the investigations were conducted in accord with the ethics policies and integrity standards of Nanyang Technological University and that the research data are presented honestly and without prejudice.

[Input Date Here]

26-April-2019

.....
Date

[Input Supervisor Signature Here]

.....
[Input Supervisor Name Here]

Kai-Kuang Ma

Authorship Attribution Statement

This thesis contains material from 5 paper(s) published in the following peer-reviewed journal(s) / from papers accepted at conferences in which I am listed as an author.

Chapter 3.1 is published as "Product Quantization Network for Fast Image Retrieval." In *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 186-201. 2018. Yu, Tan, Junsong Yuan, Chen Fang, and Hailin Jin.

The contributions of the co-authors are as follows:

- Prof. Yuan provided the initial project direction.
- Prof. Yuan, Dr. Fang, Dr. Jin and I discussed the project.
- I proposed and implemented the algorithm for the project.
- I prepared the manuscript drafts. The manuscript was revised by Prof. Yuan, Dr. Fang and Dr. Jin.

Chapter 4.1 is published as "Efficient object instance search using fuzzy objects matching." In *Thirty-First AAAI Conference on Artificial Intelligence*. 2017. Yu, Tan, Yuwei Wu, Sreyasee Bhattacharjee, and Junsong Yuan.

The contributions of the co-authors are as follows:

- Prof. Yuan provided the initial project direction.
- Prof. Yuan, Dr. Wu, Dr. Sreyasee and I joined discussions of the project.
- I proposed and implemented the algorithm for the project.
- I prepared the manuscript drafts. The manuscript was revised by Prof. Yuan, Dr. Wu and Dr. Sreyasee.

Chapter 4.2 is published as "Hope: Hierarchical object prototype encoding for efficient object instance search in videos." *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2017. Yu, Tan, Yuwei Wu, and Junsong Yuan.

The contributions of the co-authors are as follows:

- Prof. Yuan provided the initial project direction.
- Prof. Yuan, Dr. Wu and I joined discussions of the project.

- I proposed and implemented the algorithm for the project.
- I prepared the manuscript drafts. The manuscript was revised by Prof. Yuan, Dr. Wu.

Chapter 5.1 is published as "Compressive quantization for fast object instance search in videos." In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 726-735. 2017. Yu, Tan, Zhenzhen Wang, and Junsong Yuan.

The contributions of the co-authors are as follows:

- Prof. Yuan provided the initial project direction.
- Prof. Yuan, Zhenzhen Wang and I joined discussions of the project.
- I proposed and implemented the algorithm for the project.
- I prepared the manuscript drafts. The manuscript was revised by Prof. Yuan, Zhenzhen Wang.

Chapter 5.2 is published as "Is My Object in This Video? Reconstruction-based Object Search in Videos." In *IJCAI*, pp. 4551-4557. 2017. Yu, Tan, Jingjing Meng, and Junsong Yuan.

The contributions of the co-authors are as follows:

- Prof. Yuan provided the initial project direction.
- Prof. Yuan, Prof. Meng, and I joined discussions of the project.
- I proposed and implemented the algorithm for the project.
- I prepared the manuscript drafts. The manuscript was revised by Prof. Yuan, Prof. Meng.

[Input Date Here]
2019-04-25

 Date

[Input Signature Here]

 [Input Name Here]

Tan Yu

To *my parents* and *my sister*.

Acknowledgments

First and foremost, I would like to express my gratitude to Professor Junsong Yuan, who has supported me throughout my PhD study with his patience and knowledge, and encouraged me to explore my own way of research. During these years of studying under his supervision, I have learned a lot. I have been greatly impressed by his professional attitude towards research and teaching. I would also like to thank Professor Kai-Kuang Ma and Professor Xueyan Tang for their valuable suggestions on various problems in my research.

I am grateful for the opportunities to collaborate with a great many researchers and students at the Rapid Rich Object Search (ROSE) Lab, the Information Systems Lab and Media Technology Lab at Nanyang Technological University in the past four years. Yuwei Wu, Jingjing Meng, Sreyasee Das Bhattacharjee, Jiong Yang, Weixiang Hong, Zhenzhen Wang, thank you.

My sincere appreciation also goes to my QE examiners, my TAC members, the anonymous reviewers of my thesis and the panel of my PhD oral defense. Thank you for your invaluable suggestions to my research and presentation. I am thankful to my mentors at Snapchat, Zhou Ren, Yuncheng Li, En-Hsu Yen and Ning Xu, and mentors at Adobe Research, Chen Fang and Hailin Jin.

Finally, I would like to thank my parents and my sister. Without their support, this thesis would not be possible.

Abstract

Visual search has been a fundamental research topic in computer vision. Given a query, visual search aims to find the query’s relevant items from the database. Accuracy and efficiency are two key aspects for a retrieval system. This thesis focuses on the latter one, the search efficiency. To be specific, we address two basic problems in efficient visual search: 1) deep compact feature learning and 2) fast object instance search in images and videos.

To learn a compact feature for an image, we incorporate product quantization in a convolutional neural network and propose product quantization network (PQN). Through softening the hard quantization, the proposed PQN is differentiable and supports an end-to-end training. Meanwhile, we come up with a novel asymmetric triplet loss, which effectively boosts the retrieval accuracy of the proposed PQN based on asymmetric similarity. Inspired by the success of residual quantization in traditional fast image retrieval, we extend the proposed PQN to residual product quantization network (RPQN). The residual quantization triggers the residual learning, improving the representation’s discriminability. Moreover, to obtain a compact video representation, by exploiting the temporal consistency in the video, we extend PQN to temporal product quantization network (TPQN). In TPQN, we integrate frame feature learning, feature aggregation and video feature quantization in a single neural network.

For the fast object instance search in images and videos, we classify it into two categories: 1) fast object instance search in images and video frames and 2) fast object instance search in video clips. For the former one, it not only retrieves the reference images or video frames containing the query object instance, but also localizes the query object in the retrieved images or frames. To effectively capture the relevance between the query and reference images or video frames, we

utilize object proposals to obtain the candidate locations of objects. Nevertheless, matching the query with each of the object proposals in an exhaustive way is computationally intensive. To boost the efficiency, we propose fuzzy object matching (FOM) and hierarchical object prototype encoding (HOPE) which exploit the redundancy among object proposals from the same image or video frame. They encode the features of all object proposals through sparse coding and considerably reduce the memory and computation cost for the object instance search in images and video frames. For the later task, fast object instance search in video clips, we only retrieve relevant video clips and rank them according to the query, while not locating all the instances within the retrieved video clip. We propose compressive quantization (CQ) and reconstruction model (RM) to obtain a compact representation to model numerous object proposals extracted from a video clip. CQ compresses a large set of high-dimension features of object proposals into a small set of binary codes. RM characterizes a huge number of object proposals' features through a compact reconstruction model. Both RM and CQ significantly reduce the memory cost of the video clip representation and speed up the object instance search in video clips.

In a nutshell, this thesis contributes to improving the search efficiency for visual search in images and videos. Novel neural networks are developed to learn a compact image or video representation in an end-to-end manner. Meanwhile, fast object instance search strategies in images and videos are proposed, significantly reducing the memory cost and speeding up the object instance search.

Contents

Acknowledgments	i
Abstract	ii
List of Figures	viii
List of Tables	xii
List of Abbreviations	xv
List of Notations	xvi
1 Introduction	1
1.1 Background and Motivation	1
1.2 Thesis Contribution	9
1.3 Organization of the Thesis	10
2 Related Work	12
2.1 Approximate Nearest Neighbor Search	12
2.1.1 Hashing	12
2.1.2 Product Quantization	14

2.1.3	Non-exhaustive Search	15
2.2	Object Instance Search	17
2.3	Fast Video Search	18
3	Deep Compact Feature Learning	19
3.1	Product Quantization Network	19
3.1.1	From Hard Quantization to Soft Quantization	20
3.1.2	Soft Product Quantization Layer	21
3.1.3	Experiments	26
3.1.4	Summary	34
3.2	Residual Product Quantization Network	34
3.2.1	Product Quantization and Residual Quantization	35
3.2.2	Residual Product Quantization Network	37
3.2.3	Experiments	41
3.2.4	Summary	47
3.3	Temporal Product Quantization Network	48
3.3.1	Convolutional and pooling layers	50
3.3.2	Temporal Product Quantization Layer	52
3.3.3	Experiments	58
3.3.4	Summary	63
4	Fast Object Instance Search in Images and Video Frames	64
4.1	Fuzzy Object Matching	65
4.1.1	Fuzzy Object Encoding	65
4.1.2	Fuzzy Objects Matching	68

4.1.3	Fuzzy Objects Refinement	69
4.1.4	Experimental Results	71
4.1.5	Summary	79
4.2	Hierarchical Object Prototype Encoding	79
4.2.1	Problem Statement	81
4.2.2	Frame-level Object Prototype Encoding	82
4.2.3	Dataset-level Object Prototype Encoding	85
4.2.4	Non-exhaustive Search	88
4.2.5	Experiments	89
4.2.6	Summary	98
5	Fast Object Instance Search in Video Clips	99
5.1	Compressive Quantization	99
5.1.1	A Baseline by Quantization + Hashing	101
5.1.2	Compressive Quantization	103
5.1.3	Experiments	110
5.2	Reconstruction Model	119
5.2.1	Reconstruction-based Object Search	122
5.2.2	Experiments	127
5.2.3	Summary	135
6	Conclusions and Future Research	137
6.1	Conclusions	137
6.2	Future Research	139
6.2.1	Fast Search Based on Deep Learning	139

6.2.2	Self-supervised Learning for Fast Visual Search	139
6.2.3	Distilling for Compact Representation Learning	139
Author's Publications		140
Bibliography		142

List of Figures

1.1	Whole image retrieval and object instance search.	5
1.2	The scope of this thesis.	6
1.3	The structure of this thesis.	11
3.1	The overview of the proposed product quantization network.	20
3.2	The influence of parameters on PQN when using 3CNet on CIFAR-10 dataset.	27
3.3	Comparisons of our asymmetric triple loss (ATL) with cross-entropy loss (CEL) and triplet loss (TL).	30
3.4	Visualization of retrieval results from the proposed PQN. The left column contains the query images and the right columns contain the top-5 retrieved images.	31
3.5	Evaluation on the extremely short code.	32
3.6	The proposed residue product quantization layer.	36
3.7	The architecture of the proposed residual product quantization network.	37
3.8	The influence of M_p	42
3.9	The influence of α	42

3.10	The architecture of the proposed temporal product quantization network (TPQN).	51
3.11	The influence of T .	56
3.12	The influence of M .	57
3.13	The influence of α .	59
3.14	Visualization of retrieval results from the proposed TPQN.	63
4.1	Local convolutional features of each proposal are extracted from the last convolutional layer of CNN and they are further pooled or aggregated into a global feature.	65
4.2	Hundreds of object proposals tend to overlap with each other. Through clustering, the object proposals containing the similar objects will be assigned to the same group.	67
4.3	Hundreds of object proposals can be approximated by a compact set of fuzzy objects product the corresponding sparse codes.	67
4.4	The performance comparison of FOM, SCM and ROM on Oxford5K and Paris6K datasets. FOM gives the best performance.	74
4.5	Selected results of top-10 retrievals of our FOM scheme on the Oxford5K and Paris6K datasets. The bounding box in the reference image correspond to the estimated best-matched object proposal.	78
4.6	The framework of our hierarchical object prototype encoding.	80
4.7	The comparison between sphere k-means and SCS k-means.	84
4.8	Query objects visualization.	90
4.9	The mAP of two baseline algorithms on Groundhog day and NTU-VOI datasets.	91

4.10	The mAP comparison between K-means, Sphere K-means and SCS K-means on Groundhog Day and NTU-VOI datasets.	92
4.11	Comparison of the proposed frame-level object prototypes with the object representatives generated from K-medoids [1] and SMRS [2].	93
4.12	The mAP comparison between sphere k-means and TCS k-means on Groundhog Day and NTU-VOI datasets.	94
4.13	The performance comparison between sparse coding [3] and our dataset-level object prototype encoding.	95
4.14	Comparison of our dataset-level object prototype encoding with product quantization [4], optimized product quantization [5] and product quantization with coefficients [6].	96
4.15	The influence of parameters on the Groundhog Day.	97
4.16	Visualization of top-16 search results on the Groundhog dataset with query black clock and microphone.	97
4.17	Visualization of top-16 search results on the NTU-VOI dataset with query Kittyb and Maggi.	97
5.1	The overview of the proposed compressive quantization algorithm. .	100
5.2	We rotate and stretch the blue hypercube to minimize the distance from each object proposal to the corresponding vertex of the hyper- cube the object proposal is assigned to.	104
5.3	Examples of query objects.	111
5.4	mAP comparison between Baseline2 and Baseline3.	112
5.5	mAP comparison of the proposed compressive quantization (ours) with Baseline3 on the Groundhog Day dataset.	113

5.6	mAP comparison of the proposed compressive quantization (ours) with Baseline3 on the CNN2h dataset.	113
5.7	mAP comparison of the proposed compressive quantization (ours) with Baseline3 on Egocentric dataset.	113
5.8	mAP comparison between the proposed compressive quantization (ours) and Baseline2.	115
5.9	Memory comparison between the proposed compressive quantiza- tion (ours) and Baseline2.	115
5.10	Time comparison between the proposed compressive quantization (ours) and Baseline2.	116
5.11	Top-3 search results of <i>black clock</i> and <i>microphone</i>	117
5.12	Visualization of the reconstruction model.	121
5.13	Query Objects Visualization.	129
5.14	The reconstruction error decreases as the ratio of the query in- creases.	130
5.15	The performance comparison of the reconstruction model imple- mented by subspace projection and sparse autoencoder.	131
5.16	The performance of the reconstruction model implemented by k- sparse autoencoder.	132
5.17	The performance of the reconstruction model implemented by sparse coding.	132
5.18	Visualization of top-3 search results of the proposed ROSE method on three datasets.	135

List of Tables

1.1	Four types of search tasks.	2
1.2	Complexity and speedup of the proposed FOM and HOPE. M is the number of images or video frames in the database. N is the number of object proposals per image or video frame. D is each object proposal's feature dimension. K is the number of fuzzy objects per image and Z is the number of non-zero elements of each sparse code used in FOM. K_1 is the number of global object prototypes and K_2 is the number of local object prototypes of each video frame used in HOPE. Z_1 is the number of non-zero elements of each global sparse code and Z_2 is the number of non-zero elements of each local sparse code.	8
1.3	Complexity and speedup of the proposed FOM and HOPE. M is the number of video clips in the database. N is the number of object proposals per video clip. D is the feature dimension of each object proposal. K is the number of binary codes per video and L is the length of each binary code used in CQ. Z is the number of neurons in the hidden layer of the auto-encoder used in RM.	9
3.1	Comparisons with PQ and LSQ.	28
3.2	mAP comparisons with state-of-the-art methods using 3CNet. . . .	29

3.3	mAP comparisons with existing state-of-the-art methods using AlexNet base model on the CIFAR10 dataset.	31
3.4	mAP comparisons with existing state-of-the-art methods using AlexNet base model on the NUS-WIDE dataset.	33
3.5	Comparison with baselines.	43
3.6	Influence of M_r . When $M_r = 1$, the product residual product quantization degenerates into product quantization.	43
3.7	Comparison with state-of-the-art methods using 3CNet on the CIFAR10 dataset.	43
3.8	mAP comparisons with state-of-the-art methods using AlexNet on the CIFAR10 dataset.	44
3.9	mAP comparisons with existing state-of-the-art methods using AlexNet base model on the NUS-WIDE dataset.	46
3.10	Regularization	58
3.11	Comparison with existing hashing methods.	58
3.12	Comparison with deep video hashing methods.	58
4.1	The performance comparison of EOPM, FOM and the method using global features (GF) on the Oxford5K dataset.	72
4.2	The performance comparison of intra-image FOM and inter-image FOM on Oxford5K and Paris6K datasets.	75
4.3	The performance of Average Query Expansion on Oxford5K and Paris6K datasets.	75
4.4	Performance comparisons with state-of-the-art methods on Oxford5K, Paris6K and Sculptures6K datasets.	77
4.5	The complexity analysis of different schemes.	89
4.6	Efficiency comparison on the Groundhog Day.	96

5.1	The memory, time and mAP comparisons of Baseline1 and Baseline2 on three datasets.	111
5.2	Comparison of our method with k-means followed by other hashing methods.	118
5.3	Average frame ratios of 8 queries on three datasets.	129
5.4	Comparison of our ROSE method with the representative selection method and exhaustive search method.	133

List of Abbreviations

ANN	approximated nearest neighbor
ATL	asymmetric triplet loss
CEL	cross-entropy loss
EOPM	exhaustive object proposal matching
FOM	fuzzy object matching
HOPE	hierarchical object prototype encoding
IMI	inverted multi-indexing
LLC	locally linear encoding
LSH	locality-sensitive hashing
mAP	mean average precision
PCA	principle component analysis
PQ	product quantization
PQN	product quantization network
RPQN	residual product quantization network
RQ	residual quantization
SIFT	scale-invariant feature transform
SC	sparse coding
TL	triplet loss
TPQN	temporal product quantization network

List of Notations

$\frac{\partial L}{\partial x}$	the partial derivative of loss L with respect to x
$\langle \mathbf{x}, \mathbf{y} \rangle$	the inner-product between $\mathbf{x}$ and $\mathbf{y}$
$ a $	the absolute value of number a
$\ \mathbf{v}\ $	the norm of vector $\mathbf{v}$
$\alpha \in A$	α is an element of set A
$\lim_{x \rightarrow +\infty} f(x)$	the limitation of $f(x)$ when $x \rightarrow +\infty$
$\mathbb{I}(\cdot)$	the indicator function
$a \leftarrow b$	assign the value of b to a
$\mathbf{v} \in \mathbb{R}^d$	$\mathbf{v}$ is a d -dimension vector
$\mathbf{M} \in \mathbb{R}^{n \times m}$	$\mathbf{M}$ is a n -row and m -column matrix
$\operatorname{argmax}_x f(x)$	the value of x achieving the maximal $f(x)$

Chapter 1

Introduction

1.1 Background and Motivation

Thanks to the popularity of smart phones, now, many people have instant access to recording, editing and publishing images and videos. It further boosts the emergence of social network applications based on images and videos like Facebook, Snapchat, *etc.* Besides popularly used text-based search engines, content-based visual search provides another amazing way for query. Nevertheless, the explosive growth of daily uploaded images and videos from users generates an unprecedentedly huge corpus, leading to a tough challenge for image and video search.

Different from text-based search taking keywords as input, visual search takes a visual query q as input and seeks to return its relevant items from the database $\mathcal{X} = \{x_i\}_{i=1}^N$. The query q as well as an item from the database x_i can be a 2D image or a 3D video or a 1D feature vector. Basically, the search problem is a ranking problem. It ranks the relevance between the query q and each item x_i in the database based on a devised relevance function $s(q, x_i)$. The top items in the rank list are returned to the user as retrieved items. According to the type of the query and retrieved items, the search task can be categorized into four classes as

task	query	database
image-to-image	image	image
image-to-video	image	video
video-to-image	video	image
video-to-video	video	video

Table 1.1: Four types of search tasks.

shown in Table 1.1: image-to-image search, image-to-video search, video-to-image search and video-to-video search. Since video-to-image search is not widely used in the real applications, in this thesis, we explore three of above-mentioned tasks, including image-to-image search, image-to-video search and video-to-video search.

Effectiveness and efficiency are two key aspects for a retrieval system. In fact, the retrieval system design is in essence a trade-off between these two aspects. Effectiveness and efficiency also drive the progress of research on visual search in two directions. The first direction is to design or learn a more effective representation for a higher retrieval precision and recall [7, 8, 9, 10, 11, 12, 13, 14, 15, 16]. A good representation maintains a large distance between irrelevant images/videos in feature space and a close distance between relevant ones. Traditional image/video retrieval systems generate image/frame representation by aggregating hand-crafted local features [7, 8, 9]. With the progress of deep learning, the convolutional neural network provides an effective representation [10, 11, 12, 17, 18], which is trained by the semantic information and thus is robust to low-level image transformations.

In this thesis, we focus on the second direction, efficiency. We seek to achieve a high-speed search while preserving a satisfactory search accuracy. To boost the search efficiency, especially when dealing with a large-scale dataset, a compact representation is necessary. Generally, there are two main methods for compressing the high-dimension feature vector. They are hashing and quantization. Hashing

takes a real-value vector $\mathbf{x} \in \mathbb{R}^d$ as input and maps it into a hash code/bucket by

$$\mathbf{y} = [y_1, \dots, y_m, \dots, y_M] = \mathbf{h}(\mathbf{x}) = [h_1(\mathbf{x}), \dots, h_m(\mathbf{x}), \dots, h_M(\mathbf{x})], \quad (1.1)$$

where $h_m(\cdot)$ is a Hashing function and $y_m \in \{0, 1\}$. The distance between the query's feature $\mathbf{q}$ and a reference item $\mathbf{x}$ can be approximated by

$$\text{dist}(\mathbf{q}, \mathbf{x}) \approx \text{dist}(\mathbf{h}(\mathbf{q}), \mathbf{h}(\mathbf{x})). \quad (1.2)$$

Hash codes can be used in two manners, hash table lookup and hash code ranking. Hash table lookup strategy hashes the query's feature into a bucket, and look up the table to retrieve items in the dataset assigned to the same bucket. On the other hand, hash code ranking strategy computes the Hamming distance between the query's hash code and reference items' hash codes, and ranks reference items according to their Hamming distances to the query. One of the most widely used hashing method is locality sensitivity hashing (LSH) [19, 20, 21]. Nevertheless, LSH is data-independent, which ignores the data distribution and is sub-optimal to a specific dataset. To further improve search accuracy, some methods [22, 23, 24] learn the projection from the data, which cater better to a specific dataset and achieve higher accuracy than LSH. Traditional hashing methods are based on off-the-shelf visual features. They optimize the feature extraction and Hamming embedding independently. More recently, inspired by the progress of deep learning, some deep methods [25, 26, 27, 28] are proposed, which simultaneously conduct the feature learning and data compression in a unified neural network.

Nevertheless, hashing methods are only able to produce a few distinct distances, limiting its capability of describing the distance between data points. In parallel to hashing methods, another widely used data compression method in image retrieval is feature quantization. For example, product quantization (PQ) splits the feature

vector $\mathbf{x} \in \mathbb{R}^D$ into M sub-vectors $[\mathbf{x}_1, \dots, \mathbf{x}_m, \dots, \mathbf{x}_M]$ where $\mathbf{x}_m \in \mathbb{R}^{d/M}$. It quantizes each sub-vector $\mathbf{x}_m$ individually by

$$\mathbf{c}_{i_m}^m = q_m(\mathbf{x}_m), \quad (1.3)$$

where $i_m = \operatorname{argmin}_{i_m \in [1, K]} \|\mathbf{x}_m - \mathbf{c}_{i_m}^m\|_2$ and $\{\mathbf{c}_k^m\}_{k=1}^K$ are learned codewords for m -th subspace. After quantization, for each feature $\mathbf{x}$, we only store its index vector

$$\tau = [i_1, \dots, i_M], \quad (1.4)$$

which takes only $M \log(K)$ bits. The distance between the query's feature $\mathbf{q} = [\mathbf{q}_1, \dots, \mathbf{q}_M]$ and the feature of an item in the database $\mathbf{x}$ with the index vector $[i_1, \dots, i_M]$ can be approximated by

$$\operatorname{dist}(\mathbf{q}, \mathbf{x}) \approx \sum_{m=1}^M \operatorname{dist}(\mathbf{q}_m, \mathbf{c}_{i_m}^m), \quad (1.5)$$

which is efficiently obtained through looking up the table. PQ [4] and its optimized versions like optimized product quantization (OPQ) [5], CKmeans [29], additive quantization (AQ) [30] and composite quantization (CQ) [31, 32] are originally designed for an unsupervised scenario where no labeling data are provided. Supervised Quantization (SQ) [33] extends product quantization to a supervised scenario. Since SQ is based on the hand-crafted features or CNN features from the pretrained model, it might not be optimal with respect to the a specific dataset. In this thesis, the first challenge we seek to address is integrating product quantization in a convolutional neural network so that we could obtain a compact and discriminative representation for an image or a video in an end-to-end manner.

In spite of great success achieved in whole image retrieval, the object instance search is still challenging. In the object instance search task, the query is an

(a) Whole image retrieval.

(b) Object instance search.

Figure 1.1: Whole image retrieval and object instance search.

object instance and it usually occupies a small area in a reference image. The relevance between a query object instance and a reference image is not determined by the overall similarity between the query and the reference image. Figure 1.1 visualizes the difference between whole image retrieval and object instance search where queries are in the left column and their relevant images are in right columns. Green boxes highlight the locations of query objects in reference images.

In this case, the global representation of a reference image might not be effective to capture the relevance between the query object with the reference image. Inspired by the success of the object proposal scheme [34] in object detection, we utilize object proposals as potential object candidates for object instance search. We denote by $\mathbf{f}_q$ the feature of the query q and denote by $\{\mathbf{f}_x^i\}_{i=1}^T$ features of object proposals of a reference image x . The relevance between the query q and the

Figure 1.2: The scope of this thesis.

reference image x is determined by

$$s(q, x) = \max_{i \in [1, T]} \frac{1}{\text{dist}(\mathbf{f}_q, \mathbf{f}_x^i)} \quad (1.6)$$

In this framework, we need a few hundreds of object proposals for each image or video frame. However, exploring the benefits of hundreds of proposals demands the storage of all object proposals and high computational cost to match them. It might be infeasible when the dataset is large. How to efficiently utilize the large corpus of object proposals is the second challenge we seek to address in this thesis.

In this thesis, we tackle above two challenges and apply the proposed algorithms in three types of search problems including image-to-image, image-to-video and video-to-video search. Among them, image-to-image search has been widely exploited in the past decade, whereas image-to-video search and video-to-video search are less exploited. Since videos have become a bigger medium than images, video-based search becomes increasingly important and useful, which drives us to pay considerable attention to image-to-video and video-to-video tasks. Figure 1.2 illustrates the scope of this thesis.

To tackle the first challenge, incorporating product quantization in a convolutional neural network to learn a compact representation in an end-to-end manner, we construct a soft quantization function. It approximates the hard quantiza-

tion but is differentiable. Based on the constructed soft quantization function, we propose the product quantization network (PQN), which supports an end-to-end training. To make the learned representation compatible to the asymmetric similarity, we extend the original triplet loss to asymmetric triplet loss (ATL).

Inspired by the success of residual quantization in the traditional image retrieval task, we extend the PQN to residual product quantization network (RPQN) by combining the product quantization and residual quantization. The residual quantization not only enriches the codebook, but also triggers the residual learning. It improves the discriminating capability of the representation. Systematic experimental results conducted on two benchmark image datasets, CIFAR10 [35] and NUS-WIDE [36], demonstrate higher mean average precision of the proposed PQN and RPQN over other hashing and quantization methods.

Since both PQN and RPQN are optimized for fast image search, they might not be optimal for the video search task. By exploiting the temporal consistency inherited in the video through temporal pyramid, we extend PQN to temporal product quantization network (TPQN) for fast video search. Systematic experiments conducted on two benchmark video datasets, UCF101 [37] and HMDB51 [38] demonstrate the effectiveness of the proposed TPQN.

To tackle the second challenge brought by huge number of object proposals, we exploit the redundancy among object proposals. We observe that object proposals extracted from an identical image or video frame might overlap heavily, which we describe as spatial redundancy. Meanwhile, in a video, a frame and its neighboring frames might contain similar objects, which we describe as temporal redundancy.

In the task of object instance search in images, to exploit the spatial redundancy in object proposals, we propose fuzzy object matching (FOM) for fast object instance search in images. It factorizes the feature matrix of all the object proposals from an image into the product of a set of fuzzy objects and a sparse code

Method	Complexity	Speedup
Exhaustive Search	$\mathcal{O}(MND)$	1
FOM	$\mathcal{O}(MKD + MNZ)$	≈ 10
HOPE	$\mathcal{O}(K_1D + MK_2Z_1 + MNZ_2)$	≈ 100

Table 1.2: Complexity and speedup of the proposed FOM and HOPE. M is the number of images or video frames in the database. N is the number of object proposals per image or video frame. D is each object proposal’s feature dimension. K is the number of fuzzy objects per image and Z is the number of non-zero elements of each sparse code used in FOM. K_1 is the number of global object prototypes and K_2 is the number of local object prototypes of each video frame used in HOPE. Z_1 is the number of non-zero elements of each global sparse code and Z_2 is the number of non-zero elements of each local sparse code.

matrix. In this case, similarities between the query object and object proposals can be efficiently obtained through a sparse-matrix and vector multiplication. The best-matched object proposal also provides the location of the query object in the reference image. Experiments on three public datasets, Oxford5K [39], Paris6K [40] and Sculptures6K [41] show the excellent mean average precision of the proposed FOM. Meanwhile, it achieves around $10\times$ speedup over the baseline, exhaustive search. Moreover, utilizing the spatial redundancy and temporal redundancy in object proposals from frames of the same video, we extend FOM to hierarchical object prototype encoding (HOPE) for fast object instance search in video frames. We achieve state-of-the-art performance on two public datasets, Groundhog Day [42] and NTU-VOI [43], and achieve $100\times$ speedup over exhaustive search. Table 1.2 summarizes the complexity and the speedup achieved by FOM and HOPE.

As for the object instance search in video clips, since video clips are composed of individual frames, straightforwardly, we can solve it by the proposed methods for the task of object instance search in video frames such as HOPE. Nevertheless, HOPE exhaustively computes the similarities between the query and all the object proposal for a precise localization, which is unnecessary in the task of object instance search in video clips. To further speed up the search, we propose two improved methods to avoid exhaustive comparisons: compressive quantization (CQ)

Method	Complexity	Speedup
Exhaustive Search	$\mathcal{O}(MND)$	1
CQ	$\mathcal{O}(MKL)$	≈ 200
RM	$\mathcal{O}(MZD)$	≈ 200

Table 1.3: Complexity and speedup of the proposed FOM and HOPE. M is the number of video clips in the database. N is the number of object proposals per video clip. D is the feature dimension of each object proposal. K is the number of binary codes per video and L is the length of each binary code used in CQ. Z is the number of neurons in the hidden layer of the auto-encoder used in RM.

and reconstruction model (RM). CQ converts a large set of proposals’ features into a small set of binary codes. RM abstracts a great many object proposals through a reconstruction model. Experiments conducted on CNN2h [44] and Egocentric [45] datasets demonstrate the effectiveness of the proposed CQ and RM. Table 1.3 summarizes the complexity and the speedup achieved by CQ and RM.

1.2 Thesis Contribution

Our contributions lie in three parts:

- (i) Deep compact feature learning:
 - By constructing a soft quantization function, we incorporate product quantization in a convolutional neural network and propose product quantization network (PQN) which supports an end-to-end training.
 - We extend PQN to residual product quantization network (RPQN) by revisiting the residual quantization in convolutional neural network. Benefited from the residual learning triggered by the residual quantization, the search accuracy is boosted.
 - By exploiting the temporal consistency in videos through temporal pyramid, we extend the proposed PQN to temporal product quantization network (TPQN) for fast video search.

(ii) Fast object instance search in images and video frames:

- We propose fuzzy object matching (FOM) which exploits the spatial redundancy among the object proposals from the same image. It significantly speeds up the object instance search in images.
- To exploit the spatial and temporal redundancy in video frames, we propose hierarchical object prototype encoding (HOPE). It achieves state-of-the-art performance in object instance search in video frames.

(iii) Fast object instance search in video clips:

- We propose compressive quantization (CQ) to speed up the object instance search in video clips. It converts a large set of object proposals' features into a much smaller set of binary codes.
- We propose reconstruction model (RM) to improve the efficiency of object instance search in video clips. It abstracts a large number of object proposals through parameters of a compact reconstruction model.

1.3 Organization of the Thesis

This thesis is organized as follows:

In Chapter 2, we present a detailed literature review of existing works in approximated nearest neighbour search, object instance search and fast video search.

In Chapter 3, we present the details of methods proposed for deep compact feature learning. We introduce product quantization network (PQN) and residual product quantization network (RPQN) for fast image retrieval, and temporal product quantization network (TPQN) for fast video retrieval.

In Chapter 4, methods for fast object instance search in images and video frames are presented. In this chapter, we introduce fuzzy object matching (FOM)

for fast object instance search in images and hierarchical object prototype encoding (HOPE) for fast object instance search in video frames.

In Chapter 5, we present the details of the proposed methods for fast object instance in video clips. Compressive quantization (CQ) and reconstruction model (RM) are introduced.

Chapter 6 summarizes the previous chapters and presents concluding remarks on this thesis and recommends several future research directions.

To facilitate the reading, Figure 1.3 visualizes the structure of this thesis.

Figure 1.3: The structure of this thesis.

Chapter 2

Related Work

In this chapter, we review the existing works in approximate nearest neighbor search, object instance search and fast video search, respectively.

2.1 Approximate Nearest Neighbor Search

The problem of nearest neighbor search is finding nearest neighbor to a query under a certain similarity measure from a reference database. Exact nearest neighbor search is prohibitively costly under the circumstance that the search database is huge. The alternative approach, approximate nearest neighbor search, is more efficient and useful in practice. In the past decade, we have witnessed substantial progress in approximate nearest neighbor (ANN) search, especially in visual search applications. We review three related methods: hashing, product quantization and non-exhaustive search.

2.1.1 Hashing

Hashing [20, 22, 23, 24, 25, 26, 27, 28, 46, 47, 48, 49] aims to map a feature vector into a short code consisting of a sequence of bits, which enables a fast distance

computation mechanism as well as a light memory cost. A pioneering hashing method is locality sensitivity hashing (LSH) [20]. Nevertheless, it is independent to data distribution and might not be optimal to a specific dataset. To further improve the performance, some hashing methods [22, 23, 24] learn the projection from the data by minimizing the distortion error, which caters better to a specific dataset and achieves higher retrieval precision.

To further boost the performance of hashing methods when supervision information is available, some supervised and semi-supervised hashing methods [50, 51, 52, 53, 28] are proposed. According to the loss function, they can be categorized into label-based methods [50, 53, 28], pair-based methods [52, 54] and triplet-based methods [55]. To be specific, label-based methods are trained by classification loss. Pair-based methods are trained by pairwise loss, aiming to preserve the distance between two items in pairs after hashing. Triplet-based methods are trained by triplet loss which maintains the distance difference between positive pair and negative pair after hashing.

Note that the above-mentioned supervised and unsupervised methods first obtain real-value image features and then compress the features into binary codes. They conduct the representation learning and the feature compression separately and the mutual influence between them is ignored. Recently, motivated by the success of deep learning, some works [25, 26, 27, 28] propose deep hashing methods by incorporating hashing as a layer into a deep neural network. The end-to-end training mechanism of deep hashing simultaneously optimizes the representation learning and feature compression, achieving better performance than the traditional hashing methods.

2.1.2 Product Quantization

Since the hashing methods are only able to produce a few distinct distances, it has limited capability of describing the distance between data points. Another scheme termed product quantization (PQ) [4] decomposes the space into a Cartesian product of subspaces and quantizes each subspace individually. Each feature vector in the database is quantized into a short code and the search is conducted over PQ-codes efficiently through lookup tables. Note that, the distance between query and the points in the database is approximated by the asymmetric distance between the raw vector of the query and PQ-codes of points in database. Since there is no distortion error in the query side, it is a good estimation of the original Euclidean distance. Nevertheless, PQ simply divides an feature vector into sub-vectors, ignoring the data distribution. This might perform well for features in a specific structure like SIFT, but is might not be optimal in general cases. Observing this limitation, optimized product quantization (OPQ) learns a rotation matrix $\mathbf{R}$ from the dataset, minimizing the distortion errors caused by quantization. For fast similarity evaluation, the rotation matrix $\mathbf{R}$ used in OPQ is constrained to be orthogonal. But adding the orthogonal constraint leads to a larger quantization error. To achieve a smaller distortion error, AQ [30] and CQ [31] remove the orthogonal constraint and achieve higher retrieval precision. Note that production quantization and its optimized versions such as OPQ [5], AQ [30] and CQ [31] are originally designed for an unsupervised scenario where no label information is provided.

Wang *et al* [33] propose supervised quantization (SQ) by exploiting the label information. Nevertheless, SQ conducts feature extraction and quantization individually, whereas the interaction between these two steps are ignored. To simultaneously learn image representation and product quantization, deep quantization network (DQN) [56] adds a fully connected bottleneck layer in the convolutional

network. It optimizes a combined loss consisting of a similarity-preserving loss and a product quantization loss. Nevertheless, the codebook in DPQ is trained through k-means clustering and thus the supervised information is ignored. Recently, deep product quantization (DPQ) [57] is proposed where the codebook as well as the parameters are learned in an end-to-end manner. Different from original product quantization which determines the codeword assignment according to the distance between the original feature and codewords, DPQ determines the codeword assignment through a fully-connected layer whose parameters are learned from data. Nevertheless, the additional parameters in the cascade of fully-connected layers will make the network more prone to over-fitting.

2.1.3 Non-exhaustive Search

When the dataset is large, the exhaustive search is mostly infeasible to achieve satisfactory efficiency even if we adopt hashing-based binary codes or PQ-based indexing. Thus, many researchers have resorted to the non-exhaustive search.

2.1.3.1 Inverted Indexing

When dealing with billion-scale data, all the existing systems avoid infeasible exhaustive search by restricting the part of the database for each query based on an indexing structure. It partitions the feature space into a huge number of disjoint regions and the search process inspects only the points from regions closest to the query. The data points assigned to the same region are cached in a link-list and the query is only compared with heads of link-lists to find the candidate regions. After finding the regions, we compare the query with data points located in the regions exhaustively. One pioneering indexing method is introduced in [58]. It is based on inverted index which partitions the feature space into Voronoi regions obtained by K-means. Inspired by the success of product quantization, Babenko *et al.*

[59] propose IMI decomposes the feature space into several orthogonal subspaces and partitions each subspace into Voronoi regions independently. Benefited from the Cartesian product settings, the IMI space partition is extremely fine-grained. Therefore, it forms accurate lists while being memory and runtime efficient. Inspired by the success of optimize product quantization (OPQ), Babenko *et al.* [60] improves IMI and propose GNO-IMI. It discards the orthogonal constraint and generates a finer feature space partition.

2.1.3.2 Partition Tree

In million-scale regime, partition tree is a common method for ANN search. It recursively partition points in the database into cells up to a few adjoin data points in each tree leaf. In the search phase, a query is propagated down the tree in depth-first visiting order. In this case, only distances to a small number of points within visited leafs are evaluated. State-of-the-art partition trees use binary space splitting by a hyperplane $\langle \mathbf{w}, \mathbf{x} \rangle = b$ where $\mathbf{x}$ is a data point, $\mathbf{w}$ is the split direction and b is the threshold. Given a query $\mathbf{q}$, the sign of $\langle \mathbf{w}, \mathbf{q} \rangle - b$ determines the sub-tree to be visited first. When constructing a tree, we seek to use direction $\mathbf{w}$ leading to a large variance of data projection $Var\langle \mathbf{w}, \mathbf{x} \rangle$. This is due to large-variance direction tends to create a more balanced tree and make the bread-first visiting more efficient. The most popular partition-tree method is KD-tree [61]. It randomly selects a dimension and simply thresholds the scalar in that dimension to determine the visiting order in a very fast way. Nevertheless, KD-tree can only select the split direction from d coordinates basis, which might be sub-optimal if data variance along all of them are small. PCA-tree [62] improves KD-tree by splitting along the main component direction learn from to ensure the data variance is large. But the PCA-tree need conduct an additional projection operation compared with KD-tree and thus is slower. Trinary projection tree [63]

selects split direction from a restricted set, which is richer than KD-tree but not as rich as PCA-tree. Recently, inspired by the success of product quantization, PS-tree [64] achieves state-of-the-art performance by learning splitting directions in the product quantization form.

2.2 Object Instance Search

Different from the traditional image retrieval, the query object only occupies a small portion of a frame in the object instance search task. In this scenario, the relevance between the query object and the frame is not equivalent to the global similarity between the query object and the whole frame. Moreover, the object instance search task requires to precisely localize the object in the frames. In order to tackle this challenge, Razavian *et al.* [65] cropped both query image and the reference image into k patches. In their framework, the similarity between the query and the reference image is determined by k^2 times calculation of the cross-matching kernel, which is time consuming. In [66], the authors max-pooled all the proposal-level deep features from the reference image into a global feature. However, the global feature may be distracted when the query object only occupies a small area in the reference image and there exist dense clutters around it. Tolias *et al.* [67] and Mohedano *et al.* [68] uniformly sampled tens of regions in reference images. The best-matched region of the reference image is considered as the search result. However, tens of sampled regions are not enough to capture the small object in the reference image. Bhattacharjee *et al.* [69] utilized the object proposals as candidate regions of the query object in reference images and achieved state-of-the-art performance in small object instance search. Nevertheless, hundreds of object proposals brought huge amount of memory and computational cost. In order to speed up the search, Meng *et al.* [70] selected the “key objects” from

all the object proposals. To preserve the search precision, the selection ratio can not be too small, which limits its usefulness. Recently, Cao *et al.* [71] proposed a query-adaptive matching method. But it requires solving a quadratic optimization problem, which is computationally demanding.

2.3 Fast Video Search

Since a video contains considerably rich visual information, how to learn a compact representation that embodies the rich content inherited in a video is much more challenging than image hashing or quantization. Traditional video hashing methods [72, 73] follow a two-step pipeline which extracts the video feature followed by feature hashing. Nevertheless, the two-step pipeline ignores the interaction between feature learning and hashing, and thus might not yield an optimal hashing code. To overcome this drawback, Wu *et al.* [74] propose a deep video hashing method which jointly optimize frame feature learning and hashing. Similarly, Liong *et al.* [75] incorporate video feature learning and hashing function optimization in a single neural network. Recently, Liu *et al.* [76] propose a deep video hashing framework as well as a category mask to increase the discriminativity of the pursued hash codes.

Despite the problem of object instance search has been widely exploited on image datasets [77, 13, 78, 79, 67, 80], while few works have focused on video datasets. A pioneering work for object instance search in videos is Video Google [42], which treats each video keyframe independently and ranks the videos by their best-matched keyframe. Some following works [81, 82] also process the frames individually and ignore the redundancy across the frames in videos. Until recently, Meng *et al.* [70] create a pool of object proposals for each video and further select the keyobjects to speed up the search.

Chapter 3

Deep Compact Feature Learning

In this chapter, we explore the deep compact feature learning. By incorporating product quantization in a convolutional neural network, we propose product quantization network (PQN). It achieves excellent performance in fast image retrieval. Inspired by the recent progress of the residual learning, we further extend the proposed PQN to residual product quantization network (RPQN), achieving better performance in fast image retrieval. Moreover, we extend PQN to temporal product quantization network (TPQN) for fast video search.

3.1 Product Quantization Network

We attempt to incorporate the product quantization in a neural network and train it in an end-to-end manner. We propose a soft product quantization layer which is differentiable and the original product quantization is a special case of the proposed soft product quantization when $\alpha \rightarrow +\infty$. Meanwhile, inspired by the success of the triplet loss in metric learning and the triumph of the asymmetric similarity measurement in feature compression, we propose a novel asymmetric triplet loss to directly optimize the asymmetric similarity measurement in an end-to-end manner. Figure 3.1 visualizes the proposed product quantization network. CNN represents

Figure 3.1: The overview of the proposed product quantization network.

the convolutional neural network and SPQ represents the proposed soft product quantization layer. The asymmetric triplet loss takes as input a triplet consisting of the CNN feature of an anchor image (I), the SPQ feature of a positive sample (I_+) and the SPQ feature of a negative sample (I_-).

3.1.1 From Hard Quantization to Soft Quantization

Let us denote by $\mathbf{x} \in \mathbb{R}^d$ the feature of an image I , we divide the feature $\mathbf{x}$ into M subvectors $[\mathbf{x}_1, \dots, \mathbf{x}_m, \dots, \mathbf{x}_M]$ in the feature space where $\mathbf{x}_m \in \mathbb{R}^{d/M}$ is a subvector. The product quantization further approximates $\mathbf{x}$ by

$$\mathbf{q} = [q_1(\mathbf{x}_1), \dots, q_m(\mathbf{x}_m), \dots, q_M(\mathbf{x}_M)], \quad (3.1)$$

where $q_m(\cdot)$ is the quantizer for $\mathbf{x}_m$ defined as

$$q_m(\mathbf{x}_m) = \sum_k \mathbb{1}(k = k^*) \mathbf{c}_{mk}, \quad (3.2)$$

where $k^* = \underset{k}{\operatorname{argmin}} \|\mathbf{c}_{mk} - \mathbf{x}_m\|_2$, $\mathbb{1}(\cdot)$ is the indicator function and $\mathbf{c}_{mk}$ is the k -th codeword from the m -th codebook. The hard assignment makes it infeasible to

derive its derivative and thus it can not be incorporated in a neural network. This embarrassment motivates us to replace the hard assignment $\mathbb{1}(k = k^*)$ by the soft assignment $e^{-\alpha\|\mathbf{x}_m - \mathbf{c}_{mk}\|_2^2} / \sum_{k'} e^{-\alpha\|\mathbf{x}_m - \mathbf{c}_{mk'}\|_2^2}$ and obtain

$$\mathbf{s} = [s_1(\mathbf{x}_1), \dots, s_m(\mathbf{x}_m), \dots, s_M(\mathbf{x}_M)], \quad (3.3)$$

where $s_m(\cdot)$ is the soft quantizer for m -th subvector defined as

$$s_m(\mathbf{x}_m) = \sum_k \frac{e^{-\alpha\|\mathbf{x}_m - \mathbf{c}_{mk}\|_2^2} \mathbf{c}_{mk}}{\sum_{k'} e^{-\alpha\|\mathbf{x}_m - \mathbf{c}_{mk'}\|_2^2}}. \quad (3.4)$$

It is not difficult to observe that

$$\mathbb{1}(k = k^*) = \lim_{\alpha \rightarrow +\infty} \frac{e^{-\alpha\|\mathbf{x}_m - \mathbf{c}_{mk}\|_2^2}}{\sum_{k'} e^{-\alpha\|\mathbf{x}_m - \mathbf{c}_{mk'}\|_2^2}} \quad (3.5)$$

Therefore, when $\alpha \rightarrow +\infty$, the soft quantizer $s_m(\mathbf{x}_m)$ will be equivalent to the hard quantizer $q_m(\mathbf{x}_m)$. Since the soft quantization operation is differentiable and thus it can be readily incorporated into a network as a layer.

3.1.2 Soft Product Quantization Layer

Before we conduct soft product quantization in the network, we first pre-process the original feature $\mathbf{x} = [\mathbf{x}_1, \dots, \mathbf{x}_m, \dots, \mathbf{x}_M]$ through intra-normalization and conduct ℓ_2 -normalization on codewords $\{\mathbf{c}_{mk}\}_{m=1, k=1}^{M, K}$:

$$\mathbf{x}_m \leftarrow \mathbf{x}_m / \|\mathbf{x}_m\|_2 \quad (3.6)$$

$$\mathbf{c}_{mk} \leftarrow \mathbf{c}_{mk} / \|\mathbf{c}_{mk}\|_2 \quad (3.7)$$

The pre-processing step is motivated by two reasons: 1) intra-normalization and ℓ_2 -normalization can balance the contribution of each sub-vector and each codeword; 2) it simplifies the gradient computation.

3.1.2.1 Forward pass.

After intra-normalization on original features and ℓ_2 -normalization on the codewords, we can obtain $\|\mathbf{x}_m - \mathbf{c}_{mk}\|_2^2 = 2 - 2\langle \mathbf{x}_m, \mathbf{c}_{mk} \rangle$, where $\langle \cdot, \cdot \rangle$ denotes the inner product between two vectors. Based on the above property, we can rewrite Eq. (3.4) into:

$$s_m(\mathbf{x}_m) = \sum_k \frac{e^{2\alpha\langle \mathbf{x}_m, \mathbf{c}_{mk} \rangle} \mathbf{c}_{mk}}{\sum_{k'} e^{2\alpha\langle \mathbf{x}_m, \mathbf{c}_{mk'} \rangle}}. \quad (3.8)$$

3.1.2.2 Backward pass.

To elaborate the backward pass of the soft quantization layer, we introduce an immediate variable a_{mk} defined as

$$a_{mk} = \frac{e^{2\alpha\langle \mathbf{x}_m, \mathbf{c}_{mk} \rangle}}{\sum_{k'} e^{2\alpha\langle \mathbf{x}_m, \mathbf{c}_{mk'} \rangle}}. \quad (3.9)$$

Based on the above definition, Eq. (3.8) will be converted into

$$s_m(\mathbf{x}_m) = \sum_k a_{mk} \mathbf{c}_{mk}. \quad (3.10)$$

Through chain rule, we can obtain the derivative of loss with respect to $\mathbf{c}_{mk}$ by

$$\frac{\partial L}{\partial \mathbf{c}_{mk}} = a_{mk} \frac{\partial L}{\partial s_m(\mathbf{x}_m)} + \sum_{k'} \frac{\partial a_{mk'}}{\partial \mathbf{c}_{mk}} \left(\frac{\partial s_m(\mathbf{x}_m)}{\partial a_{mk'}} \right)^\top \frac{\partial L}{\partial s_m(\mathbf{x}_m)}, \quad (3.11)$$

where

$$\frac{\partial s_m(\mathbf{x}_m)}{\partial a_{mk'}} = \mathbf{c}_{mk'}, \quad (3.12)$$

and

$$\frac{\partial a_{mk'}}{\partial \mathbf{c}_{mk}} = \begin{cases} \frac{-e^{2\alpha\langle \mathbf{x}_m, \mathbf{c}_{mk'} \rangle} e^{2\alpha\langle \mathbf{x}_m, \mathbf{c}_{mk} \rangle} 2\alpha \mathbf{x}_m}{(\sum_{k''} e^{2\alpha\langle \mathbf{x}_m, \mathbf{c}_{mk''} \rangle})^2}, & k \neq k' \\ \frac{\sum_{k' \neq k} e^{2\alpha\langle \mathbf{x}_m, \mathbf{c}_{mk'} \rangle} e^{2\alpha\langle \mathbf{x}_m, \mathbf{c}_{mk} \rangle} 2\alpha \mathbf{x}_m}{(\sum_{k''} e^{2\alpha\langle \mathbf{x}_m, \mathbf{c}_{mk''} \rangle})^2}, & k = k' \end{cases} \quad (3.13)$$

By plugging Eq. (3.12) and Eq. (3.13) into Eq. (3.11), we can obtain $\frac{\partial L}{\partial \mathbf{c}_{mk}}$.

3.1.2.3 Initialization

We initialize the parameters of convolutional layers by fine-tuning a standard convolutional neural network without quantization, *e.g.*, Alexnet, on the specific dataset. Note that, we add an intra-normalization layer to fine-tune the network to make it compatible with our deep product quantization network. After the initialization of convolutional layers, we extract the features from the fine-tuned network and conduct k-means followed by ℓ_2 -normalization to obtain the initialized codewords $\{\mathbf{c}_{mk}\}_{k=1, m=1}^{K, M}$ in the soft product quantization layer.

3.1.2.4 Asymmetric Triplet Loss

We propose a novel asymmetric triplet loss to optimize the parameters of the network. We define (I, I_+, I_-) as a training triplet, where I_- and I_+ represent a relevant image and an irrelevant image with respect to the anchor image I . We denote by $\mathbf{x}_I$ as the feature of I before soft product quantization and denote by $\mathbf{s}_{I_+}$ and $\mathbf{s}_{I_-}$ the features of I_+ and I_- after soft product quantization. We define asymmetric similarity between I and I_+ as $\langle \mathbf{x}_I, \mathbf{s}_{I_+} \rangle$, where $\langle \cdot, \cdot \rangle$ denotes the inner-product operation. The proposed asymmetric triplet loss is defined as

$$l = \langle \mathbf{x}_I, \mathbf{s}_{I_-} \rangle - \langle \mathbf{x}_I, \mathbf{s}_{I_+} \rangle. \quad (3.14)$$

Intuitively, it aims to increase the asymmetric similarity between the pairs of relevant images and decrease that of pairs consisting of irrelevant images. It is a natural extension of original triplet loss on the condition of asymmetric distance. The difference is that, a training triplet used in original triplet loss consists of three features of the same type, whereas a training triplet used in the proposed asymmetric triplet loss consists of one feature without quantization and two features after quantization. In fact, our experiments show that a better performance is achieved by processing above loss through sigmoid function and a revised loss function is defined as Eq. (3.15). The better performance might be contributed by the fact that the sigmoid function can normalize the original loss so that the training will not be biased by some samples causing huge loss.

$$l = \frac{1}{1 + e^{\langle \mathbf{x}_I, \mathbf{s}_{I_+} \rangle - \langle \mathbf{x}_I, \mathbf{s}_{I_-} \rangle}}. \quad (3.15)$$

3.1.2.5 Encode and retrieval.

After training the proposed product quantization network, the reference images in the database will be encoded by hard product quantization. We define the layer before the soft product quantization layer as embedding layer. Given a reference image I of the database, we obtain its output from embedding layer $\mathbf{x} = [\mathbf{x}_1, \dots, \mathbf{x}_m, \dots, \mathbf{x}_M]$ and further obtain its product quantization code $\mathbf{b} = [b_1, \dots, b_m, \dots, b_M]$ where b_m is computed by

$$b_m = \underset{k}{\operatorname{argmax}} \langle \mathbf{x}_m, \mathbf{c}_{mk} \rangle, \quad (3.16)$$

where $\{\mathbf{c}_{mk}\}_{m=1, k=1}^{M, K}$ are codewords learned from our product quantization network. In the retrieval phase, we obtain the query feature from the embedding layer $\mathbf{q} = [\mathbf{q}_1, \dots, \mathbf{q}_m, \dots, \mathbf{q}_M]$. The relevance between the query image and a reference image represented by its product quantization code $\mathbf{b} = [b_1, \dots, b_m, \dots, b_M]$ is

computed by the asymmetric similarity $s(\mathbf{q}, \mathbf{b})$ defined as

$$s(\mathbf{q}, \mathbf{b}) = \sum_{m=1}^M \langle \mathbf{q}_m, \mathbf{c}_{mb_m} \rangle. \quad (3.17)$$

Since $\langle \mathbf{q}_m, \mathbf{c}_{mb_m} \rangle$ is computed only once for all the reference images in the database and thus obtaining $s(\mathbf{q}, \mathbf{b})$ only requires to sum up the pre-computed similarity scores in the look-up table, considerably speeding up the image retrieval process. Meanwhile, storing the product quantization code $\mathbf{b}$ only requires $M \log_2 K$ bits, which considerably reduces the memory cost.

3.1.2.6 Relation to existing methods.

DQN [56] is the first attempt of incorporating product quantization in the neural network. It alternatively optimizes codewords and other parameters of the network. It is worth noting that when updating codewords, it only minimizes the quantization errors through k-means. Therefore, when learning codewords, the supervision information is ignored and the solution might be sub-optimal.

SUBIC [83] integrates the one-hot block encoding layer in the deep neural network. It represents each image by a product of one-hot blocks, following the spirit of product quantization. Nevertheless, the sparse property limits its representation capability, making it perform not as well as ours.

DPQ [57] is another attempt of incorporating the product quantization into the neural network. It determines the codeword assignment through a cascade of two fully-connected layers. In contrast, our method determines the codeword assignment according to the similarity between original feature and the codewords. Note that, the additional parameters from these two fully-connected layers in DPQ not only increase the computation complexity in training the neural network but also are more prone to over-fitting. Our experiments show that our proposed PQN considerably outperforms DPQ.

3.1.3 Experiments

We evaluate the performance of our PQN on two public benchmark datasets, CIFAR-10 and NUS-WIDE. CIFAR-10 [35] is a dataset containing 60,000 color images in 10 classes, and each class has 6,000 images in size 32×32 . Different from CIFAR-10, NUS-WIDE [36] is a dataset for evaluating multi-class classification, in which one sample is assigned to one or multiple labels. We follow the settings in [26, 56] and use the subset of 195,834 images that are associated with the 21 most frequent concepts, where each concept consists of at least 5,000 images. We resize all images into 256×256 .

On the CIFAR-10 dataset, the performance reported by different baselines are based on different base convolutional neural networks, making it unfair to directly compare their reported retrieval accuracy. To make a fair comparison, we evaluate our method based on two types of convolutional neural networks. The first convolutional neural network we use is 3CNet which is also used by SUBIC [83] and DQN [57]. 3CNet is proposed in [27], which consists of $L = 3$ convolutional layers with 32, 32 and 64 filters of size 5×5 respectively, followed by a fully connected layer with $d = 500$ nodes. The second convolutional neural network we choose is AlexNet. It is worth noting that the baselines we compare may apply different models. For example, DQN [56] adopts AlexNet whereas other work [84, 28] adopt VGG-F model. These two models are similar in the architecture. To be specific, both the CNN-F and AlexNet consist of five convolutional layers and two fully connected layers. As shown in [85], the CNN-F generally performs better than Alexnet in image retrieval, therefore, the better performance of ours based on AlexNet than existing state-of-art methods based on CNN-F is not owing to better base network. In other words, our method can achieve better performance even with an inferior base network. On the NUS-WIDE dataset, we also adopt AlexNet as our base model. On both datasets, we report the performance of the proposed

Figure 3.2: The influence of parameters on PQN when using 3CNet on CIFAR-10 dataset.

method through mAP, which is a standard metric in evaluating the performance of retrieval algorithms.

3.1.3.1 CIFAR-10 using 3CNet

Following the experimental setup in SUBIC [83] and DPQ [57], the training is conducted on 50K image training set. The test set is split into 9K database images and 1K query images (100 per class).

Influence of M and K . In this section, we evaluate the influence of the number of subvectors M and the number of codewords per sub-codebook K on the retrieval precision of the proposed PQN. We vary M among $\{1, 2, 4, 8\}$, and vary $B = \log_2 K$ among $\{3, 6, 9, 12\}$. As shown in Figure 3.2(b), the proposed method achieves the best performance when $M = 4$. By default, we set $M = 4$ on CIFAR-10 Dataset. Note that when $M = 1$ and $M = 2$, the performance of the proposed PQN increases as B increases. This is expected since the larger B can partition the feature space into finer cells. Nevertheless, when $M = 4$, the

method	4bits	8bits	16bits	24bits	32bits
TL+Full	0.779				
TL+PQ	0.503	0.621	0.741	0.773	0.780
TL+LSQ	0.511	0.720	0.752	0.753	0.763
PQN(Ours)	0.574	0.729	0.778	0.782	0.786

Table 3.1: Comparisons with PQ and LSQ.

performance drops when B increases from 9 to 12. Meanwhile, when $M = 8$, there is also a performance drop when K increases from 6 to 9. The worse performance might be caused by over-fitting when both M and K are large.

Influence of α . α controls the quantization softness of the soft product quantization layer. We evaluate the performance of our method when α varies. We test the influence of α when $M = 4$ and K varies among $\{2^3, 2^6, 2^9, 2^{12}\}$. As shown in Figure 3.2(a), the performance of the proposed PQN is relatively stable when α increases from 1 to 80. Note that, when $\alpha = 1$, the performance is slightly worse than that when $\alpha = 5$. The worse performance is due to the fact a small α will make the quantization too soft and thus the soft quantization in training phase differs too much from the hard quantization in the testing phase. Meanwhile, we also observe a performance drop when α increases from 20 to 40. This drops might be caused by the fact that a huge α tends to push the input of soft-max function to the saturation region and lead to gradient vanishing.

Comparison with unsupervised PQ/LSQ. We compare with unsupervised PQ and LSQ [86] based on fine-tuned features trained through triplet loss. As shown in Table 3.1, ours considerably outperforms both TL+PQ and TL+LSQ. Meanwhile, we also show the performance of original features trained through triplet loss without quantization (TL+Full) in Table 3.1. The performance of ours is even better than that of features without quantization, this is owing to the regularization imposed by quantization, which suppresses over-fitting.

Effectiveness of asymmetric triplet loss. Meanwhile, in order to show the

Method	12 bits	24 bits	36 bits	48 bits
SUBIC [83]	0.635	0.672	0.682	0.686
DPQ [57]	0.673	0.692	0.695	0.693
Ours+CEL	0.737	0.771	0.768	0.762
Ours+ATL	0.741	0.782	0.787	0.786

Table 3.2: mAP comparisons with state-of-the-art methods using 3CNet.

effectiveness of the proposed asymmetric triplet loss, we compare with two alternatives, cross-entropy loss (CEL) and triplet loss (TL). To make a fair comparison, we only change the loss function and keep the other parts of the network unchanged. As shown in Figure 3.3, the proposed asymmetric loss consistently outperforms the cross-entropy loss and triplet loss when L varies among $\{12, 24, 36, 48\}$. For instance, when $L = 36$, our ATL achieves a 0.787 mAP whereas TL only achieves a 0.778 mAP and CEL only achieves a 0.768 mAP.

Compare with state-of-the-art methods. We compare our method with two state-of-the-art methods (SUBIC and DPQ), which adopt the same 3CNet as well as the same experimental settings. We change bit length L among $\{12, 24, 36, 48\}$. We set $M = 4$ and $K = 8, 64, 512, 4096$, respectively. Since SUBIC adopts cross-entropy loss, it is unfair to directly compare it with ours using asymmetric triplet loss. Therefore, we report the performance of our PQN based on the cross-entropy loss (CEL) as well as the proposed asymmetric triplet loss (ATL). As shown in Table 3.2, our method based on both CEL and ATL significantly outperform the existing state-of-the-art methods including SUBIC and DPQ. For instance, when $L = 24$, ours achieves a 0.771 mAP based on the cross-entropy loss and a 0.782 mAP using the proposed asymmetric triplet loss whereas SUBIC only achieves a 0.672 mAP and DPQ only achieves a 0.692 mAP.

Figure 3.3: Comparisons of our asymmetric triple loss (ATL) with cross-entropy loss (CEL) and triplet loss (TL).

3.1.3.2 CIFAR-10 using AlexNet

Following the experimental settings in [84, 28], we randomly sample 1000 images per class (10000 images in total) as the testing query images, and the remaining 50000 images are used as the training set as well as reference images in the database. We set $M = 4$ and vary K among $\{2^4, 2^6, 2^9, 2^{12}\}$, and thus the code length L varies among $\{16, 24, 36, 48\}$.

Comparisons with state-of-the-art methods. As shown in Table 3.3, ours consistently outperforms the existing state-of-the-art methods, especially when the bit length is small. For instance, when the bit length is 16, our method based on asymmetric triplet loss (Ours+ATL) achieves a 0.947 mAP whereas the second best method, DSDH, only achieves a 0.935 mAP. We visualize the retrieval result obtained from the proposed PQN in Figure 3.4.

Short code evaluation. As shown in Table 3.3, the mAP achieved by our method does not drop when the bit length decreases from 48 to 16. In contrast, the performance of other methods in Table 3.3 all turn worse due to decrease of

Method	16 bits	24 bits	36 bits	48 bits
DRSCH [87]	0.615	0.622	0.629	0.631
DSCH [87]	0.609	0.613	0.617	0.686
DSRH [88]	0.608	0.611	0.617	0.618
VDSH [89]	0.845	0.848	0.844	0.845
DPSH [90]	0.903	0.885	0.915	0.911
DTSH [84]	0.915	0.923	0.925	0.926
DSDH [28]	0.935	0.940	0.939	0.939
Ours + CE	0.939	0.941	0.941	0.940
Ours + ATL	0.947	0.947	0.946	0.947

Table 3.3: mAP comparisons with existing state-of-the-art methods using AlexNet base model on the CIFAR10 dataset.

Figure 3.4: Visualization of retrieval results from the proposed PQN. The left column contains the query images and the right columns contain the top-5 retrieved images.

Figure 3.5: Evaluation on the extremely short code.

the bit length. To fully exploit the potential of the proposed product quantization network on the CIAFR-10 dataset, we evaluate it by setting the code length L extremely small. We vary M among 1, 2 and 4, and meanwhile vary the code length (bit number) L within $\{4, 6, 8, 10, 12\}$. As shown in Figure 3.5, when code length is extremely small, *e.g.*, $L = 4$, the performance of PQN when $M = 1$ significantly outperforms that when $M = 2, 4$. Meanwhile, when $M = 1$, there is not significant performance drop when L decreases from 12 to 4. Note that, when $M = 1$, the proposed PQN achieves a 0.945 mAP when using only 4 bits per code. It considerably outperforms the existing state-of-art method DSDH [28] which only achieves 0.935 mAP using 16 bits.

3.1.3.3 NUS-WIDE

Following the experiment setup in [84, 28], we randomly sample 100 images per class (2100 images in total) as the test query set and the remaining images are used as database images. 500 database images per label are randomly sampled as training images. The mAP is calculated based on the top 5000 returned neighbors.

Method	12 bits	24 bits	36 bits	48 bits
SH + CNN [28]	0.621	0.616	0.615	0.612
ITQ + CNN [28]	0.719	0.739	0.747	0.756
LFH + CNN [28]	0.695	0.734	0.739	0.759
KSH + CNN [28]	0.768	0.786	0.790	0.799
SDH+ CNN [28]	0.780	0.804	0.815	0.824
FASTH+CNN [28]	0.779	0.807	0.816	0.825
CNNH [25]	0.611	0.618	0.625	0.608
NINH [26]	0.674	0.697	0.713	0.715
DHN [91]	0.708	0.735	0.748	0.758
DQN [56]	0.768	0.776	0.783	0.792
DPSH [90]	0.752	0.790	0.794	0.812
DTSH [84]	0.773	0.808	0.812	0.824
DSDH [28]	0.776	0.808	0.820	0.829
Ours	0.795	0.819	0.823	0.830

Table 3.4: mAP comparisons with existing state-of-the-art methods using AlexNet base model on the NUS-WIDE dataset.

Due to multi-label settings, the cross-entropy loss used in SUBIC [83] and the softmax loss in DPQ [57] are no longer feasible, which explains neither SUBIC [83] nor DPQ [57] conducts the experiments on NUS-WIDE dataset. Inspired by the success of label embedding proposed in [28], we also adopt a combined loss, which is a weighed sum of our asymmetric triplet loss and a mean square loss defined as

$$l = \frac{1}{1 + e^{\langle \mathbf{x}_I, \mathbf{s}_{I+} \rangle - \langle \mathbf{x}_I, \mathbf{s}_{I-} \rangle}} + \beta \|\mathbf{W}\mathbf{s}_I - \mathbf{y}_I\|_2^2, \quad (3.18)$$

where $\mathbf{W}$ is the parameter in an additional FC layer after the soft product quantization layer and $\mathbf{y}_I$ is the label of the sample I . We set $\beta = 10$ by default.

Comparisons with existing state-of-the-art methods. We compare our method with two types of baselines. The first type extracts the features from CNN and then convert the extracted features into binary codes. We directly copy the reported results in [28] which conducts experiments on several traditional hashing methods such as SH [22], ITQ [24], KSH [51], SDH [92], *etc.* The baselines of the second type are deep hashing/quantization methods, where the binary codes are

learned in an end-to-end manner. We compare several methods of the second type such as DQN [56], DPSH [90], DTSH [84], DSDH [28], *etc.* As shown in Table 3.4, the proposed PQN consistently outperforms these two types of baselines when code length L varies among $\{12, 24, 36, 48\}$. The advantage of our PQN over other methods is more obvious when the code length L is short. For instance, when $L = 12$, our PQN achieves a 0.795 mAP whereas the second best method, FASTH+CNN [28] only achieves a 0.779 mAP.

3.1.4 Summary

In this section, by incorporating product quantization in the neural network, we propose product quantization network (PQN) to learn a discriminative and compact image representation in an end-to-end manner. Meanwhile, we propose a novel asymmetric triplet loss, which directly optimizes the image retrieval based on asymmetric distance to train our network more effectively. Systematic experiments conducted on benchmark datasets demonstrate the state-of-the-art performance of the proposed PQN.

3.2 Residual Product Quantization Network

Compared with the popularity of product quantization [5, 31, 33, 56, 57], residual quantization [93] is rarely explored. It is due to that using codebook of the same scale, residual quantization tends to cause a larger quantization error than product quantization. If we adopt the off-the-shelf features like SIFT or pretrained CNN features, a larger quantization error tend to yield a lower retrieval precision. Nevertheless, in a deep learning network, there existing huge amount of redundancy in activations and the quantization error is less important. Meanwhile, we observe that the residual quantization naturally triggers residual learning [94], which can

boost the features' discriminability.

Based on above motivation, we revisit the residual quantization in a deep neural network. We demonstrate the advantage of residual quantization over product quantization when incorporated into a deep neural network. In fact, the residual quantization in our framework not only serves to minimize the distortion error from the quantization, but more importantly, it enhance the discriminability of the learned compact representation. In implementation, to overcome the discontinuity of the original hard quantization, we approximate it through a continuously differentiable soft quantization function which is the core element of the proposed residual product quantization network (RPQN).

3.2.1 Product Quantization and Residual Quantization

Since our residual product quantization is a hybrid of product quantization and residual quantization, we review these two methods, respectively. In fact, both product quantization and residual quantization can be seen as an extension of vector quantization (VQ) [95]. The original VQ maps a feature vector to the closest codeword learned through k-means clustering. To maintain a fine partition of feature space, especially in a high-dimension scenario, the number of codewords of VQ has to be considerably large, making the memory complexity of storing the codebook relatively high. Product quantization [4] and residual quantization [4] are two types of solutions to generating a huge number of codewords and meanwhile maintain a low memory complexity when storing the codebook.

3.2.1.1 Product Quantization

Product quantization (PQ) [4] decomposes the original feature space into a Cartesian product of M disjoint sub-space. Each sub-space is further partitioned into K clusters and data samples within a cluster can be quantized into the centroid of the

Figure 3.6: The proposed residue product quantization layer.

cluster. Given a feature vector $\mathbf{f}$, PQ divides it into M_p sub-vectors $[\mathbf{f}_1, \dots, \mathbf{f}_{M_p}]$ and then quantizes the feature vector into $\mathbf{q} = [Q_1(\mathbf{f}_1), \dots, Q_{M_p}(\mathbf{f}_{M_p})]$, where Q_{M_p} denotes the quantizer of the M_p -th subspace. Thanks to the product settings, PQ can divide the whole feature space into K^{M_p} cells using only $\mathcal{O}(Kd)$ memory cost where d is the dimension of the feature vector.

3.2.1.2 Residual Quantization

Different from PQ, residual quantization (RQ) [93] performs quantization to the whole feature space, and then recursively applies VQ models to the residuals of the previous quantization level, which is a stacked quantization model. In other words, different from PQ conducting VQ independently on several partitioned sub-spaces, RQ conducts VQ on multiple levels instead. For the first level, RQ simply applies VQ to quantize a feature vector $\mathbf{f}$ into one of K centroids through $q_0(\mathbf{f})$. The feature vector's residual vector $\mathbf{r}_1$ can be obtained by subtracting its corresponding centroid by $\mathbf{f} - q_0(\mathbf{f})$. After than, $\mathbf{r}_1$ will be further quantized into K centroids through $q_1(\mathbf{r}_1)$. By repeatedly quantizing the residuals $M_r - 1$ times, we can approximate $\mathbf{f}$ by $q_0(\mathbf{f}) + \sum_{m=1}^{M_r-1} q_m(\mathbf{r}_m)$. Through combining codewords in M_r levels, we obtain a considerably huge codebook consisting of K^{M_r} codewords.

Figure 3.7: The architecture of the proposed residual product quantization network.

Generally, residual quantization can not achieve comparable distortion error as product quantization, which limits its effectiveness in retrieval based on off-the-shelf features.

3.2.2 Residual Product Quantization Network

Figure 3.7 shows the architecture of the proposed residual product quantization network (RPQN). The network takes input a triplet consisting of an anchor image $\mathbf{I}$, a positive image $\mathbf{I}_+$ and a negative image $\mathbf{I}_-$. The whole architecture mainly consists of three main components: 1) The backbone network consisting of a series of convolutional layers followed fully-connected layers, 2) a residual product quantization layer, and 3) an asymmetric triplet loss. Since the backbone network is off-the-shelf, we mainly focus on introduction of the proposed residual product quantization layer (RPQL) and asymmetric triplet loss (ATL) in this section.

3.2.2.1 Residual Product Quantization Layer

Figure 3.6 illustrates the proposed residual product quantization layer. We denote by $\mathbf{x} \in \mathbb{R}^d$ the input feature vector and divide $\mathbf{x}$ into M_p sub-vectors $[\mathbf{x}_1, \dots, \mathbf{x}_m, \dots, \mathbf{x}_{M_p}]$ ($M_p = 2, M_r = 2$ in this example). For each sub-vector $\mathbf{x}_m$, it goes through the

following forward path:

- $\mathbf{q}_m^1 \leftarrow Q_1(\mathbf{x}_m, \alpha)$
- $\mathbf{r}_m^1 \leftarrow \mathbf{x}_m - \mathbf{q}_m^1$
- $\mathbf{q}_m^2 \leftarrow Q_2(\mathbf{r}_m^1, \alpha)$
- $\mathbf{q}_m \leftarrow \mathbf{q}_m^1 + \mathbf{q}_m^2$

$Q_1(\mathbf{x}_m, \alpha)$ and $Q_2(\mathbf{r}_m^1, \alpha)$ are soft quantization functions defined as

$$\begin{aligned} Q_1(\mathbf{x}_m, \alpha) &= \sum_{k=1}^K \frac{e^{\alpha \langle \mathbf{c}_m^{1,k}, \mathbf{x}_m \rangle} \mathbf{c}_m^{1,k}}{\sum_{k'=1}^K e^{\alpha \langle \mathbf{c}_m^{1,k'}, \mathbf{x}_m \rangle}}, \\ Q_2(\mathbf{r}_m^1, \alpha) &= \sum_{k=1}^K \frac{e^{\alpha \langle \mathbf{c}_m^{2,k}, \mathbf{r}_m^1 \rangle} \mathbf{c}_m^{2,k}}{\sum_{k'=1}^K e^{\alpha \langle \mathbf{c}_m^{2,k'}, \mathbf{r}_m^1 \rangle}} \end{aligned} \quad (3.19)$$

where $\langle \cdot, \cdot \rangle$ is the inner-product operation between two vectors, $\{\mathbf{c}_m^{1,k}\}_{k=1}^K$ represent the first-level codewords for quantizing the original input feature $\mathbf{x}_m$ and $\{\mathbf{c}_m^{2,k}\}_{k=1}^K$ denote the second-level codewords for quantizing the residuals $\mathbf{r}_m^1$. α is the coefficient to control the softness of the quantization and it is not difficult to observe that:

$$\lim_{\alpha \rightarrow +\infty} Q_1(\mathbf{x}_m, \alpha) = \mathbf{c}_m^{1,k_1^*}, \quad \lim_{\alpha \rightarrow +\infty} Q_2(\mathbf{r}_m^1, \alpha) = \mathbf{c}_m^{2,k_2^*}, \quad (3.20)$$

where

$$k_1^* = \operatorname{argmax}_k \langle \mathbf{c}_m^{1,k}, \mathbf{x}_m \rangle, \quad k_2^* = \operatorname{argmax}_k \langle \mathbf{c}_m^{2,k}, \mathbf{r}_m^1 \rangle, \quad (3.21)$$

From Eq. (3.20), we can find the original hard quantization is just a special case of the proposed soft quantization function when $\alpha \rightarrow +\infty$. In implementation, choosing a large α can approximate the hard quantization well. More importantly,

overcoming the discontinuity of original hard quantization, the proposed soft product quantization is continuously differentiable, which can be readily embed in a neural network. Below we derive the backward path of soft quantization function $Q_1(\mathbf{x}_m, \alpha)$ and that of $Q_2(\mathbf{r}_m^1, \alpha)$ can be derived in a similar manner.

Let's assume we have already obtained the derivative of the loss with respect to the output $\partial L / \partial Q_1(\mathbf{x}_m, \alpha)$ and the backward path seeks to derive the derivative of the loss with respect to input $\partial L / \partial \mathbf{x}_m$ and that with respect to each codeword $\partial L / \partial \mathbf{c}_m^{1,k}$. To facilitate the derivation, we introduce an intermediate variable a_k defined as

$$a_k = \frac{e^{\alpha \langle \mathbf{c}_m^{1,k}, \mathbf{x}_m \rangle}}{\sum_{k'=1}^K e^{\alpha \langle \mathbf{c}_m^{1,k'}, \mathbf{x}_m \rangle}}. \quad (3.22)$$

Intuitively, a_k denotes the soft assignment probability. Based on the above definition, we rewrite Eq. (3.19) into

$$Q_1(\mathbf{x}_m, \alpha) = \sum_{k=1}^K a_k \mathbf{c}_m^{1,k} \quad (3.23)$$

According to the chain rule, we can obtain $\frac{\partial L}{\partial \mathbf{x}_m}$ and $\frac{\partial L}{\partial \mathbf{c}_m^{1,k}}$ by

$$\begin{aligned} \frac{\partial L}{\partial \mathbf{x}_m} &= \left[\frac{\partial L}{\partial Q_1(\mathbf{x}_m, \alpha)} \right]^\top \sum_{k=1}^K \frac{\partial Q_1(\mathbf{x}_m, \alpha)}{\partial a_k} \frac{\partial a_k}{\partial \mathbf{x}_m} \\ &= \sum_{k=1}^K \left\langle \frac{\partial L}{\partial Q_1(\mathbf{x}_m, \alpha)}, \mathbf{c}_m^{1,k} \right\rangle \frac{\partial a_k}{\partial \mathbf{x}_m} \end{aligned} \quad (3.24)$$

$$\begin{aligned} \frac{\partial L}{\partial \mathbf{c}_m^{1,k}} &= \sum_{k'=1}^K \left\{ a_{k'} \frac{\partial \mathbf{c}_m^{1,k'}}{\partial \mathbf{c}_m^{1,k}} \frac{\partial L}{\partial Q_1} + \left[\frac{\partial L}{\partial Q_1} \right]^\top \frac{\partial Q_1}{\partial a_{k'}} \frac{\partial a_{k'}}{\partial \mathbf{c}_m^{1,k}} \right\} \\ &= a_k \frac{\partial L}{\partial Q_1(\mathbf{x}_m, \alpha)} + \sum_{k'=1}^K \left\langle \frac{\partial L}{\partial Q_1(\mathbf{x}_m, \alpha)}, \mathbf{c}_m^{1,k'} \right\rangle \frac{\partial a_{k'}}{\partial \mathbf{c}_m^{1,k}} \end{aligned} \quad (3.25)$$

where $\frac{\partial a_k}{\partial \mathbf{x}_m}$ and $\frac{\partial a_{k'}}{\partial \mathbf{c}_m^{1,k}}$ are computed by

$$\frac{\partial a_k}{\partial \mathbf{x}_m} = \frac{\alpha e^{\alpha \langle \mathbf{c}_m^{1,k}, \mathbf{x}_m \rangle} \sum_{k'=1}^K e^{\alpha \langle \mathbf{c}_m^{1,k'}, \mathbf{x}_m \rangle} (\mathbf{c}_m^{1,k} - \mathbf{c}_m^{1,k'})}{(\sum_{k'=1}^K e^{\alpha \langle \mathbf{c}_m^{1,k'}, \mathbf{x}_m \rangle})^2} \quad (3.26)$$

$$\frac{\partial a_{k'}}{\partial \mathbf{c}_m^{1,k}} = \begin{cases} \frac{-\alpha e^{\alpha \langle \mathbf{c}_m^{1,k} + \mathbf{c}_m^{1,k'}, \mathbf{x}_m \rangle} \mathbf{x}_m}{(\sum_{k''=1}^K e^{\alpha \langle \mathbf{c}_m^{1,k''}, \mathbf{x}_m \rangle})^2}, & k' \neq k \\ \frac{\sum_{k'' \neq k} \alpha e^{\alpha \langle \mathbf{c}_m^{1,k} + \mathbf{c}_m^{1,k''}, \mathbf{x}_m \rangle} \mathbf{x}_m}{(\sum_{k''=1}^K e^{\alpha \langle \mathbf{c}_m^{1,k''}, \mathbf{x}_m \rangle})^2}, & k' = k \end{cases} \quad (3.27)$$

3.2.2.2 Asymmetric Triplet Loss

The asymmetric triplet loss we used is an extension of original triplet loss. It directly takes input of a triplet consists of original feature of anchor image $\mathbf{x}_I$ and the quantization feature of the positive image $\mathbf{q}_{I+}$ and the quantization feature of the negative image $\mathbf{q}_{I-}$. It aims to maximize the similarity margin between the positive pair and the negative pair defined as $\langle \mathbf{x}_I, \mathbf{q}_{I+} \rangle - \langle \mathbf{x}_I, \mathbf{q}_{I-} \rangle$. We define the asymmetric triplet loss as

$$\mathcal{L}_{ATL} = [\langle \mathbf{x}_I, \mathbf{q}_{I-} \rangle - \langle \mathbf{x}_I, \mathbf{q}_{I+} \rangle + m]_+, \quad (3.28)$$

where m is a predefined margin. Intuitively, the asymmetric triplet loss directly optimize the asymmetric similarity, which makes the learned features more adaptive to asymmetric similarity measurement.

3.2.2.3 Indexing and Retrieval

After training the network, in the indexing phase, we encode each reference image I in the database through hard-assignment residual product quantization. Let us define the layer before residual product quantization layer as embedding layer. We denote by $\mathbf{x}$ the output of embedding layer when input is I . $\mathbf{x}$ is partitioned into

$[\mathbf{x}_1, \dots, \mathbf{x}_m, \dots, \mathbf{x}_{M_p}]$ and each sub-vector $\mathbf{x}_m$ is indexed as follows:

- $i_m^1 \leftarrow \operatorname{argmax}_i \langle \mathbf{x}_m, \mathbf{c}_m^{1i} \rangle$
- $\mathbf{r}_m^1 \leftarrow \mathbf{x}_m - \mathbf{c}_m^{1i_m^1}$
- $i_m^2 \leftarrow \operatorname{argmax}_i \langle \mathbf{r}_m^1, \mathbf{c}_m^{2i} \rangle$

In the indexing phase, for each reference image I , we only need to store indices of their corresponding codewords $\{i_m^1\}_{m=1}^{M_p}$ and $\{i_m^2\}_{m=1}^{M_p}$, taking $2M_p \log_2 K$ bits in total.

In the retrieval phase, we utilize the asymmetric similarity scores to rank the reference images in the dataset. Let denote by $\mathbf{x}_q$ as the output of embedding layer when the input is query image q . Let denote by $[i_1^1, \dots, i_{M_p}^1]$ and $[i_1^2, \dots, i_{M_p}^2]$ indices in two-level residual product quantization of the reference image I . The asymmetric similarity is defined as

$$\operatorname{sim}_{\text{asym}}(q, I) = \sum_{m=1}^{M_p} [\langle \mathbf{x}_q, \mathbf{c}_m^{1i_m^1} \rangle + \langle \mathbf{x}_q, \mathbf{c}_m^{2i_m^2} \rangle]. \quad (3.29)$$

Therefore, for each query, we only need to compute its similarity with all the codewords for only once and then similarity between the query and each reference image can be efficiently obtained through $2M_p$ table look-ups.

3.2.3 Experiments

We evaluate the proposed RPQN on two public benchmark datasets, CIFAR-10 and NUS-WIDE. CIFAR-10 [35] contains 60000 color images in 10 classes, and each class has 6000 images in size 32×32 . NUS-WIDE [36] is built for multi-class classification task, where one sample is assigned to one or multiple labels. Following [26, 56], we use a subset of 195834 images associated with the 21 most frequent concepts.

Figure 3.8: The influence of M_p . Figure 3.9: The influence of α .

On the CIFAR-10 dataset, the performance reported by different baselines are based on different backbones. To make a fair comparison, we evaluate our method on two backbones, 3CNet and AlexNet. 3CNet is used in SUBIC [83] and DQN [57]. It consists of 3 convolutional layers with 32, 32 and 64 filters of size 5×5 respectively, followed by a fully connected layer with $d = 500$ nodes. Some baselines such as DQN [56] adopt AlexNet whereas some works [84, 28] adopts VGG-F model. These two models are similar. Both of them consist of five convolutional layers followed by two fully connected layers. We choose AlexNet as our base model since there is no pre-trained CNN-F model available in pytorch. Even though [85] shows that CNN-F performs better than AlexNet in image retrieval, the proposed

Method	8 bits	16 bits	24 bits	32 bits
FULL	0.779			
FULL + PQ	0.621	0.741	0.773	0.780
FULL + RQ	0.610	0.732	0.753	0.763
RPQN (ours)	0.739	0.768	0.799	0.798

Table 3.5: Comparison with baselines.

M_r	12 bits	24 bits	36 bits	48 bits
1	0.741	0.782	0.787	0.786
2	0.758	0.799	0.796	0.797

Table 3.6: Influence of M_r . When $M_r = 1$, the product residual product quantization degenerates into product quantization.

RPQN based on AlexNet still achieves a better performance than baselines based on CNN-F. On the NUS-WIDE dataset, we adopt AlexNet as backbone. On both datasets, we report the performance through mAP.

3.2.3.1 CIFAR-10 using 3CNet

Following the experimental setup in SUBIC [83] and DPQ [57], the training is conducted on 50K image training set. The test set is split into 9K database images and 1K query images (100 per class).

To validate the effectiveness of end-to-end training of the proposed RPQN, we compare it with several baselines based on unquantized features or features obtained from unsupervised quantization methods. The first baseline is achieved by training an Alexnet using the triplet loss and use the activations of fully-connected

Method	12 bits	24 bits	36 bits	48 bits
SUBIC [83]	0.635	0.672	0.682	0.686
DPQ [57]	0.673	0.692	0.695	0.693
PQN+ATL	0.741	0.782	0.787	0.786
RPQN+CEL	0.726	0.772	0.771	0.782
RPQN+ATL	0.758	0.799	0.796	0.797

Table 3.7: Comparison with state-of-the-art methods using 3CNet on the CIFAR10 dataset.

Method	16 bits	24 bits	36 bits	48 bits
DRSCH [87]	0.615	0.622	0.629	0.631
DSCH [87]	0.609	0.613	0.617	0.686
DSRH [88]	0.608	0.611	0.617	0.618
DMDH [96]	0.704	-	-	0.732
VDSH [89]	0.845	0.848	0.844	0.845
DPSH [90]	0.903	0.885	0.915	0.911
DTSH [84]	0.915	0.923	0.925	0.926
DSDH [28]	0.935	0.940	0.939	0.939
Ours	0.950	0.949	0.949	0.948

Table 3.8: mAP comparisons with state-of-the-art methods using AlexNet on the CIFAR10 dataset.

layer without quantization as the feature for retrieval. We define this baseline as FULL shown in Table 3.5. We further do the original unsupervised product quantization and residual quantization using the obtained FULL feature. We define these two baselines as FULL+PQ and FULL+RQ, respectively. As shown in Table 3.5, our RPQN consistently outperform FULL+PQ and FULL+RQ, which validates the effectiveness of the end-to-end training. Meanwhile, ours also outperforms FULL when bit number is larger than 24, which shows the advantage of residual learning triggered by the proposed RPQN.

We then evaluate the influence of residual level M_r on the performance of our RPQN. Note that, when $M_r = 1$, the residual product quantization will be equivalent to the product quantization. We compare the case when $M_r = 1$ and that when $M_r = 2$. Since the code length $L = M_p M_r \log_2 K$, to achieve an identical code length, we set $M_p = 4$ when $M_r = 1$, and set $M_p = 2$ when $M_r = 2$. As shown in Table 3.6, the performance when $M_r = 2$ consistently outperforms that when $M_r = 1$, which verify the advantage of residual quantization over product quantization. We also evaluate the influence of M_p . We fix $M_r = 2$ and vary M_p among $\{1, 2, 4\}$ and vary K among $\{8, 64, 512, 4096\}$. As shown in Figure 3.8, when $M_p = 2$, it achieves the highest mAP and meanwhile it only takes a half bits of that when $M_p = 4$. By default, we set $M_p = 2$.

Note that, a too small α in Eq. (3.19) will lead to a high distortion between hard quantization and soft quantization, leading to a relatively bad performance. Meanwhile, a too large α will push the input of the softmax function in soft-quantization to the saturation region. To demonstrate the sensitivity of the proposed method with respect to α we evaluate the performance of our RPQN when α varies among $\{1, 2, 5, 10\}$. To be specific, we conduct the experiments by fixing the subspace number $M_p = 2$ and residual quantization level $M_r = 2$ and vary the number of codewords per subspace per level K among $\{8, 64, 512, 4096\}$. As shown in Figure 3.9, the mAP of our RPQN first increases and then drops as α increases. But it is considerably stable. By default, we set $\alpha = 2$ in all experiments.

To further demonstrate the effectiveness of our method, we compare it with existing state-of-the-art methods. We change bit length L among $\{12, 24, 36, 48\}$. We choose a 2-level residual quantization ($M_r = 2$), set $M_p = 2$ and vary K among $\{8, 64, 512, 4096\}$, respectively. Since SUBIC adopts cross-entropy loss, it is unfair to directly compare it with ours using asymmetric triplet loss. Therefore, we report the performance of our RPQN based on the cross-entropy loss (CEL) as well as the proposed asymmetric triplet loss (ATL). As shown in Table 3.7, our method based on ATL outperforms the existing state-of-the-art methods including SUBIC, DPQ and our PQN. For instance, when $L = 24$, ours achieves a 0.799 mAP using the proposed asymmetric triplet loss whereas SUBIC only achieves a 0.672 mAP, DPQ only achieves a 0.692 mAP and PQN achieves a 0.782 mAP.

3.2.3.2 CIFAR-10 using AlexNet

Following the experimental settings in [84, 28], we randomly sample 1000 images per class (10000 images in total) as the testing query images, and the remaining 50000 images are used as the training set as well as reference images in the database. We choose a 2-level residual quantization, set $M_p = 2$ and increase K from 2^4 to

Method	12 bits	24 bits	36 bits	48 bits
SH + CNN [28]	0.621	0.616	0.615	0.612
ITQ + CNN [28]	0.719	0.739	0.747	0.756
LFH + CNN [28]	0.695	0.734	0.739	0.759
KSH + CNN [28]	0.768	0.786	0.790	0.799
SDH+ CNN [28]	0.780	0.804	0.815	0.824
FASTH+CNN [28]	0.779	0.807	0.816	0.825
CNNH [25]	0.611	0.618	0.625	0.608
NINH [26]	0.674	0.697	0.713	0.715
DHN [91]	0.708	0.735	0.748	0.758
DQN [56]	0.768	0.776	0.783	0.792
DPSH [90]	0.752	0.790	0.794	0.812
DTSH [84]	0.773	0.808	0.812	0.824
DSDH [28]	0.776	0.808	0.820	0.829
DMDH [96]	-	-	-	0.789
Ours	0.789	0.822	0.829	0.831

Table 3.9: mAP comparisons with existing state-of-the-art methods using AlexNet base model on the NUS-WIDE dataset.

2^{12} , and thus the code length L increases from 16 to 48. To demonstrate the excellent performance of the proposed RPQN, we compare it with several state of art methods like DRSCH [87], DSCH [87], DSRH [88], VDSH [89], DPSH [90], DTSH [84] and DSDH [28]. As shown in Table 3.8, ours consistently outperforms the existing state-of-the-art methods, especially when the bit length is small. For instance, when the bit length is 16, ours achieves a 0.950 mAP whereas the second best method, DSDH, only achieves a 0.935 mAP.

3.2.3.3 NUS-WIDE

Following the experiment setup in [84, 28], we randomly sample 100 images per class (2100 images in total) as the test query set and the remaining images are used as database images. 500 database images per label are randomly sampled as training images. The mAP is calculated based on the top 5000 returned neighbors. Due to multi-label settings, the cross-entropy loss used in SUBIC [83] and the softmax loss in DPQ [57] are no longer feasible, which explains neither SUBIC [83]

nor DPQ [57] conducts the experiments on NUS-WIDE dataset. Inspired by the success of label embedding proposed in [28], we also adopt a combined loss, which is a weighed sum of our asymmetric triplet loss and a mean square loss defined as

$$\mathcal{L}_{hybrid} = \mathcal{L}_{ATL} + \beta \|\mathbf{W}\mathbf{s}_I - \mathbf{y}_I\|_2^2, \quad (3.30)$$

where $\mathbf{W}$ is the parameter in an additional FC layer after the soft product quantization layer and $\mathbf{y}_I$ is the ground-truth label. β is a constant controlling the weights of two losses. By default we set $\beta = 5$.

We compare our methods with two types of baselines. The baselines of the first type first extract the features from CNN and then convert the extracted features into binary codes. We directly copy the reported results in [28] which conducts experiments on several traditional hashing method such as SH [22], ITQ [24], KSH [51], SDH [92], *etc.* The baselines of the second type are deep hashing/quantization methods, where the binary codes are learned in an end-to-end manner. We compare several methods of the second type such as DQN [56], DPSH [90], DTSH [84], DSDH [28], *etc.* As shown in Table 3.9, the proposed RPQN consistently outperforms these two types of baselines when the code length L varies among $\{12, 24, 36, 48\}$ and our advantage is especially apparent when bit number is small. For instance, when $L = 24$, our RPQN achieves a 0.822 mAP whereas the second best method, DSDH [28] only achieves a 0.808 mAP.

3.2.4 Summary

In this paper, we revisit an underestimated feature compression method, residual quantization. By incorporating it in a deep neural network, it not only achieves feature compression but also trigger the residual learning which boosts the features' discriminability. In implementation, to make the quantization differentiable, we

construct a soft quantization function, which is an approximation of original hard quantization. It is continuously differentiable and make the whole network trainable in an end-to-end manner. Moreover, we propose a novel asymmetric triplet loss, which directly optimizes the feature’s discriminability in asymmetric metric. Systematic experiments conducted on two public benchmark datasets demonstrate the state-of-the-art performance of the proposed RPQN.

3.3 Temporal Product Quantization Network

In this section, we explore the video-to-video retrieval task. Video-to-video search has multiple applications such as copy detection, particular event detection, person re-identification. In this task, the query is a video clip, which possesses much more representation capability than still images. For instance, the video query could contain the appearance of the object in multiple views and meanwhile it could embody the dynamic visual changes which is important to describe an action. Nevertheless, due to rich visual content inherited in video instances, designing a discriminative and compact video representation for an effective and efficient video-to-video search is more challenging than image search.

To capture the relevance between videos, some works tackle the video-to-video search by frame-based matching. Nevertheless, a video might contain a huge number of frames, making the frame-based matching very slow. To boost the efficiency, some works aggregate the local features of all frames of the video into a global feature, and the video-to-video search is based on the Euclidean distances between the aggregated global features of the query video and reference videos. However, the global representation ignores the temporal consistency among the frames and thus might be sub-optimal for video retrieval.

To exploit the temporal relations among frames, Lea *et al.* [97] utilize 3D CNN

with temporal convolution and some others [98] adopt RNN. Nevertheless, both 3D CNN and RNN are hard to train and require a huge amount of training data. Alternative methods [99, 100] use structured representations to model the temporal consistency inherited in the action. Inspired the temporal pyramid used in [99, 100], we propose temporal product quantization network to boost the efficiency in video-to-video search and exploit the temporal consistency in videos.

Despite two mainstream approaches, hashing [21, 23, 24, 91, 88, 27, 26] and quantization [4, 5, 29, 30, 31, 33], have achieved great success in fast image retrieval, there are few works exploring video retrieval in literature. Since a video contains considerably rich visual information, how to learn a compact representation that embodies the rich content inherited in a video is much more challenging than image hashing or quantization. Traditional video hashing methods [72, 73] follow a two-step pipeline which extracts the video feature followed by feature hashing. Nevertheless, the two-step pipeline ignores the interaction between feature learning and hashing, and thus might not yield an optimal hashing code. To overcome this drawback, Wu *et al.* [74] propose a deep video hashing method which jointly optimize frame feature learning and hashing. Similarly, Liong *et al.* [75] incorporate video feature learning and hashing function optimization in a single neural network. Recently, Liu *et al.* [76] propose a deep video hashing framework as well as a category mask to increase the discriminativity of the pursued hash codes. In this section, we incorporate product quantization in a deep neural network. To make it differentiable for end-to-end training, we approximate discontinuous quantization by constructing a continuously differential function and the original quantization is a special case of the constructed function. We implement the constructed approximation function as a product quantization layer, which is the core component of the proposed temporal product quantization network. Meanwhile, we exploit the temporal consistency inherited in the video through temporal pyramid and extend

the original product quantization to our temporal product quantization. Moreover, interestingly, we find that quantization serves as a regularization approach, which improves the model’s generalization capability and boost the retrieval precision.

We exploit the temporal consistency inherited in the video through temporal pyramid and extend the original product quantization to our temporal product quantization. Moreover, interestingly, we find that quantization serves as a regularization approach, which improves the model’s generalization capability and boost the retrieval precision. We incorporate frame feature learning, frame feature aggregation and video feature product quantization in a single network, supporting an end-to-end training. To exploit the video’s temporal consistency, we extend the original product quantization to a temporal product quantization, achieving a better retrieval accuracy. We discover that the quantization can serve as a regularization method which boosts the generalization capability of a network and achieves a higher accuracy in retrieval.

3.3.1 Convolutional and pooling layers

Given a video V , we uniformly sample N frames and define the set of sampled frames as $I_G = \{f_1, \dots, f_N\}$. We feed frames $\{f_i\}_{i=1}^N$ in parallel into the backbone network consisting of convolutional and pooling layers and obtain a set of frame-level features $\{F(f_i)|f_i \in I_G\}$. We uniformly partition the frames in I_G into T intervals $\{I_t|t \in [1, T] \cap I_t \subset I_G\}$. Features of frames in each interval I_t are maxpooled into an interval-level feature:

$$\mathbf{x}_t = \text{maxpool}(\{F(f)|f \in I_t\}), \quad t = 1, \dots, T + 1 \quad (3.31)$$

where $I_{T+1} = I_G$ accounts for the global set. All the interval-level features will be concatenated into a global feature $\mathbf{x} = [\mathbf{x}_1, \dots, \mathbf{x}_{T+1}]$, which is the input of the pro-

Figure 3.10: The architecture of the proposed temporal product quantization network (TPQN).

posed temporal product quantization layer. Intuitively, $\mathbf{x}$ is a two-level temporal pyramid pooling feature which exploits the temporal consistency inherited in the videos. But more importantly, the pyramid structure provides a natural feature partition for product quantization, which we will discuss in the later section.

3.3.2 Temporal Product Quantization Layer

For each interval-level feature $\mathbf{x}_t$, following the standard product quantization, we equally split it into M sub-vectors as $\mathbf{x}_t = [\mathbf{x}_{t1}, \dots, \mathbf{x}_{tM}]$. Each $\mathbf{x}_{tm}$ goes through the following pipeline consisting of 4 steps:

- (i) $\mathbf{y}_{tm} = \text{relu}(\mathbf{W}_{tm}\mathbf{x}_{tm} + \mathbf{b}_{tm})$
- (ii) $\bar{\mathbf{y}}_{tm} = \frac{\mathbf{y}_{tm}}{\|\mathbf{y}_{tm}\|_2}$
- (iii) $\mathbf{z}_{tm} = q_{tm}(\bar{\mathbf{y}}_{tm})$
- (iv) $\bar{\mathbf{z}}_{tm} = \beta \frac{\mathbf{z}_{tm}}{\|\mathbf{z}_{tm}\|_2}$

where $\mathbf{W}_{tm}$ and $\mathbf{b}_{tm}$ in step 1 are parameters of a fully-connected layer used to further boost the feature's discriminability. β in step 4 is a scaling scalar to speed up the convergence and quantizer $q_{tm}(\bar{\mathbf{y}}_{tm})$ is defined as

$$q_{tm}(\bar{\mathbf{y}}_{tm}) = \mathbf{c}_{tm}^{k^*}, \quad (3.32)$$

where

$$k^* = \arg \max_{k \in [1, K]} \langle \bar{\mathbf{y}}_{tm}, \mathbf{c}_{tm}^k \rangle. \quad (3.33)$$

In Eq. (3.33), $\{\mathbf{c}_{tm}^1, \dots, \mathbf{c}_{tm}^K\}$ are codewords for $q_{tm}(\cdot)$. Originally, the codewords are learned through unsupervised k-means, which minimizing the distortion errors between codewords and the data points. Nevertheless, when incorporated into a

neural network in a supervised scenario, we seek to learn the codewords through back-propagation.

Let us derive the back-propagation of gradients in the third step of the pipeline since that of all other steps are trivial. We define L as the training loss, which we will introduce it in details lately. Let's assume we have already computed $\partial L / \partial \mathbf{z}_{tm}$ from the fourth step in the pipeline. It is straightforward to derive that the $\partial L / \partial \mathbf{c}_{tm}^{k*} = \partial L / \partial \mathbf{z}_{tm}$. Intuitively, it back-propagates the gradients of the output to the codewords it is assigned to and leave the other codewords unchanged. Nevertheless, there is no continuous function describing the relationship between input $\bar{\mathbf{y}}_{tm}$ and output $\mathbf{z}_{tm}$, making it infeasible to back-propagate $\partial L / \partial \mathbf{z}_{tm}$ to $\partial L / \partial \bar{\mathbf{y}}_{tm}$. In other words, the gradient from the later layers can not be back-propagated to earlier layers to update their parameters.

To overcome the non-differentiable problem, we generalize the quantization defined in Eq. (3.32) by constructing a continuously differential approximation function defined as $f_{tm}(\bar{\mathbf{y}}_{tm}, \alpha)$ which satisfies $\lim_{\alpha \rightarrow +\infty} f_{tm}(\bar{\mathbf{y}}_{tm}, \alpha) = q_{tm}(\bar{\mathbf{y}}_{tm})$, where α is a scalar controlling the consistency between $f_{tm}(\bar{\mathbf{y}}_{tm}, \alpha)$ and $q_{tm}(\bar{\mathbf{y}}_{tm})$. In practice, we can choose a large α to achieve a good approximation. There are multiple choices to construct the function $f_{tm}(\bar{\mathbf{y}}_{tm}, \alpha)$. Below we specify two instances:

$$\begin{aligned} f_{tm}^1(\bar{\mathbf{y}}_{tm}, \alpha) &= \sum_{k=1}^K \frac{\langle \bar{\mathbf{y}}_{tm}, \mathbf{c}_{mt}^k \rangle^\alpha}{\sum_{k'=1}^K \langle \bar{\mathbf{y}}_{tm}, \mathbf{c}_{mt}^{k'} \rangle^\alpha} \mathbf{c}_{mt}^k, \\ f_{tm}^2(\bar{\mathbf{y}}_{tm}, \alpha) &= \sum_{k=1}^K \frac{e^{\alpha \langle \bar{\mathbf{y}}_{tm}, \mathbf{c}_{mt}^k \rangle}}{\sum_{k'=1}^K e^{\alpha \langle \bar{\mathbf{y}}_{tm}, \mathbf{c}_{mt}^{k'} \rangle}} \mathbf{c}_{mt}^k \end{aligned} \quad (3.34)$$

Since our experiments show that $f_{tm}^1(\bar{\mathbf{y}}_{tm}, \alpha)$ outperforms $f_{tm}^2(\bar{\mathbf{y}}_{tm}, \alpha)$, we select $f_{tm}^1(\bar{\mathbf{y}}_{tm}, \alpha)$ as the default approximation function. Note that, $\lim_{\alpha \rightarrow +\infty} f_{tm}^1(\bar{\mathbf{y}}_{tm}, \alpha) = q_{tm}(\bar{\mathbf{y}}_{tm})$ might not satisfy if $\langle \bar{\mathbf{y}}_{tm}, \mathbf{c}_{mt}^k \rangle < 0$. To tackle this issue, we clip $\langle \bar{\mathbf{y}}_{tm}, \mathbf{c}_{mt}^k \rangle$

to ensure it is non-negative. Below we derive the gradient back-propagation for $f_{tm}^1(\bar{\mathbf{y}}_{tm}, \alpha)$. To facilitate the derivation, we introduce an intermediate variable a_{tm}^k defined as

$$a_{tm}^k = \langle \bar{\mathbf{y}}_{tm}, \mathbf{c}_{tm}^k \rangle^\alpha / \sum_{k'=1}^K \langle \bar{\mathbf{y}}_{tm}, \mathbf{c}_{tm}^{k'} \rangle^\alpha \quad (3.35)$$

and obtain

$$\mathbf{z}_{tm} = f_{tm}^1(\bar{\mathbf{y}}_{tm}, \alpha) = \sum_{k=1}^K a_{tm}^k \mathbf{c}_{tm}^k. \quad (3.36)$$

Since $\mathbf{z}_{tm}$ is a function of $\bar{\mathbf{y}}_{tm}$ and $\{\mathbf{c}_{tm}^k\}_{k=1}^K$, it is straightforward to obtain

$$\begin{aligned} dL &= \sum_{t=1}^{T+1} \sum_{m=1}^M \left(d\mathbf{z}_{tm} \right)^\top \frac{\partial L}{\partial \mathbf{z}_{tm}} \\ &= \sum_{t=1}^{T+1} \sum_{m=1}^M \left(\frac{\partial \mathbf{z}_{tm}}{\partial \bar{\mathbf{y}}_{tm}} d\bar{\mathbf{y}}_{tm} + \sum_{k=1}^K \frac{\partial \mathbf{z}_{tm}}{\partial \mathbf{c}_{tm}^k} d\mathbf{c}_{tm}^k \right)^\top \frac{\partial L}{\partial \mathbf{z}_{tm}}. \end{aligned} \quad (3.37)$$

Meanwhile, according the definition of $\mathbf{z}_{tm}$ in Eq. (3.36), we obtain

$$\begin{aligned} \frac{\partial \mathbf{z}_{tm}}{\partial \bar{\mathbf{y}}_{tm}} &= \sum_{k=1}^K \frac{\partial \mathbf{z}_{tm}}{\partial a_{tm}^k} \left(\frac{\partial a_{tm}^k}{\partial \bar{\mathbf{y}}_{tm}} \right)^\top = \sum_{k=1}^K \mathbf{c}_{tm}^k \left(\frac{\partial a_{tm}^k}{\partial \bar{\mathbf{y}}_{tm}} \right)^\top, \\ \frac{\partial \mathbf{z}_{tm}}{\partial \mathbf{c}_{tm}^k} &= \sum_{k''=1}^K \frac{\partial \mathbf{z}_{tm}}{\partial a_{tm}^{k''}} \left(\frac{\partial a_{tm}^{k''}}{\partial \mathbf{c}_{tm}^k} \right)^\top + a_{tm}^k \frac{\partial \mathbf{c}_{tm}^k}{\partial \mathbf{c}_{tm}^k} \\ &= \sum_{k''=1}^K \mathbf{c}_{tm}^{k''} \left(\frac{\partial a_{tm}^{k''}}{\partial \mathbf{c}_{tm}^k} \right)^\top + a_{tm}^k \mathbf{I}. \end{aligned} \quad (3.38)$$

By plugging Eq. (3.38) into Eq. (3.37), we further obtain

$$\begin{aligned} dL &= \sum_{t=1}^{T+1} \sum_{m=1}^M \left\{ \sum_{k=1}^K \mathbf{c}_{tm}^k \left(\frac{\partial a_{tm}^k}{\partial \bar{\mathbf{y}}_{tm}} \right)^\top d\bar{\mathbf{y}}_{tm} \right. \\ &\quad \left. + \sum_{k=1}^K \left[\sum_{k''=1}^K \mathbf{c}_{tm}^{k''} \left(\frac{\partial a_{tm}^{k''}}{\partial \mathbf{c}_{tm}^k} \right)^\top + a_{tm}^k \mathbf{I} \right] d\mathbf{c}_{tm}^k \right\}^\top \frac{\partial L}{\partial \mathbf{z}_{tm}}. \end{aligned} \quad (3.39)$$

As the loss L is also the function of $\bar{\mathbf{y}}_{tm}$ and $\mathbf{c}_{tm}$, therefore,

$$dL = \sum_{t=1}^{T+1} \sum_{m=1}^M \left[\left(d\bar{\mathbf{y}}_{tm} \right)^\top \frac{\partial L}{\partial \bar{\mathbf{y}}_{tm}} + \sum_{k=1}^K \left(d\mathbf{c}_{tm}^k \right)^\top \frac{\partial L}{\partial \mathbf{c}_{tm}^k} \right] \quad (3.40)$$

Comparing Eq. (3.40) and Eq. (3.39), it is not difficult to obtain

$$\begin{aligned} \frac{\partial L}{\partial \bar{\mathbf{y}}_{tm}} &= \sum_{k=1}^K \frac{\partial a_{tm}^k}{\partial \bar{\mathbf{y}}_{tm}} \left(\mathbf{c}_{tm}^{k''} \right)^\top \frac{\partial L}{\partial \mathbf{z}_{tm}}, \\ \frac{\partial L}{\partial \mathbf{c}_{tm}^k} &= \sum_{k''=1}^K \frac{\partial a_{tm}^{k''}}{\partial \mathbf{c}_{tm}^k} \left(\mathbf{c}_{tm}^{k''} \right)^\top \frac{\partial L}{\partial \mathbf{z}_{tm}} + a_{tm}^k \frac{\partial L}{\partial \mathbf{z}_{tm}} \end{aligned} \quad (3.41)$$

where

$$\begin{aligned} \frac{\partial a_{tm}^k}{\partial \bar{\mathbf{y}}_{tm}} &= \frac{\alpha \sum_{k'=1}^K \langle \bar{\mathbf{y}}_{tm}, \mathbf{c}_{mt}^{k'} \rangle^{\alpha-1} \langle \bar{\mathbf{y}}_{tm}, \mathbf{c}_{mt}^k \rangle^{\alpha-1} \mathbf{c}_{mt}^k \langle \bar{\mathbf{y}}_{tm}, \mathbf{c}_{mt}^{k'} \rangle}{\left(\sum_{k'=1}^K \langle \bar{\mathbf{y}}_{tm}, \mathbf{c}_{mt}^{k'} \rangle^\alpha \right)^2} \\ &\quad - \frac{\alpha \sum_{k'=1}^K \langle \bar{\mathbf{y}}_{tm}, \mathbf{c}_{mt}^{k'} \rangle^{\alpha-1} \langle \bar{\mathbf{y}}_{tm}, \mathbf{c}_{mt}^k \rangle^{\alpha-1} \mathbf{c}_{mt}^{k'} \langle \bar{\mathbf{y}}_{tm}, \mathbf{c}_{mt}^k \rangle}{\left(\sum_{k'=1}^K \langle \bar{\mathbf{y}}_{tm}, \mathbf{c}_{mt}^{k'} \rangle^\alpha \right)^2} \\ \frac{\partial a_{tm}^{k''}}{\partial \mathbf{c}_{tm}^k} &= \frac{\alpha \mathbb{I}(k = k'') \mathbf{c}_{mt}^{k''} \langle \bar{\mathbf{y}}_{tm}, \mathbf{c}_{mt}^{k''} \rangle^{\alpha-1} \sum_{k'=1}^K \langle \bar{\mathbf{y}}_{tm}, \mathbf{c}_{mt}^{k'} \rangle^\alpha}{\left(\sum_{k'=1}^K \langle \bar{\mathbf{y}}_{tm}, \mathbf{c}_{mt}^{k'} \rangle^\alpha \right)^2} \\ &\quad - \frac{\alpha \mathbf{c}_{mt}^k \langle \bar{\mathbf{y}}_{tm}, \mathbf{c}_{mt}^k \rangle^{\alpha-1} \langle \bar{\mathbf{y}}_{tm}, \mathbf{c}_{mt}^{k''} \rangle^\alpha}{\left(\sum_{k'=1}^K \langle \bar{\mathbf{y}}_{tm}, \mathbf{c}_{mt}^{k'} \rangle^\alpha \right)^2} \end{aligned} \quad (3.42)$$

$\mathbb{I}(k = k') = 1$ only if $k = k'$ otherwise 0.

3.3.2.1 Regularization

Interestingly, the quantization mechanism works as a regularization, effectively suppressing over-fitting. Different from traditional regularization method putting constraints on the weights of the network such as ℓ_1/ℓ_2 regularization, quantization, instead, puts constraints on the activations. We will show that even though the quantized features can be seen as an approximation of the original features with inevitable distortion, it achieves a much better retrieval precision than that using

Figure 3.11: The influence of T .

the original features.

3.3.2.2 Indexing and Retrieval

After training the neural network, we will use the learned codewords but adopt hard quantization for indexing. To be specific, given a reference video V^r in the database, let $\mathbf{x}_t^r = [\mathbf{x}_{t1}^r, \dots, \mathbf{x}_{tm}^r, \dots, \mathbf{x}_{tM}^r]$ denote the max-pooled feature of the intervals I_t where $\mathbf{x}_{tm}^r$ is the m -th sub-vector. For each $\mathbf{x}_{tm}^r$, we will obtain its index i_{tm}^r through the following:

$$(i) \quad \mathbf{y}_{tm}^r = \text{relu}(\mathbf{W}_{tm}\mathbf{x}_{tm}^r + \mathbf{b}_{tm})$$

$$(ii) \quad \bar{\mathbf{y}}_{tm}^r = \frac{\mathbf{y}_{tm}^r}{\|\mathbf{y}_{tm}^r\|_2}$$

$$(iii) \quad i_{tm}^r = \arg \max_k \langle \bar{\mathbf{y}}_{tm}^r, \mathbf{c}_{tm}^k \rangle$$

Let us define K as the number of codewords used by each quantizer $q_{tm}(\cdot)$, it takes $\log_2(K)$ bits to store i_{tm}^r and $(T+1)M\log_2(K)$ bits to store all indices $[i_{11}^r, \dots, i_{T+1M}^r]$ for the video V . Through combination, we can obtain $K^{(T+1)M}$ possible codewords, which has relatively strong representation capability. The advantage of temporal quantization over original product quantization is two-fold: 1) it not only exploits the temporal consistency which is inherited in the video;

Figure 3.12: The influence of M .

2) it increases the number of codewords from K^M to $K^{(T+1)M}$, leading to a finer partition of the feature space.

In the retrieval phase, given a query video V^q , we can directly obtain its concatenated max-pooled feature $[\mathbf{x}_{11}^q, \dots, \mathbf{x}_{T+1M}^q]$ through the trained backbone network, further post-process each $\mathbf{x}_{tm}^q$ through $\{\mathbf{W}_{tm}, \mathbf{b}_{tm}\}$ followed by ℓ_2 normalization and obtain the final feature $[\bar{\mathbf{y}}_{11}^q, \dots, \bar{\mathbf{y}}_{T+1M}^q]$. Note that we do not conduct quantization on the query's feature due to the asymmetric similarity measurement. The similarity between the query V^q and a reference video in the database V^r is computed by

$$\text{sim}(V^q, V^r) = \sum_{t=1}^{T+1} \sum_{m=1}^M \langle \bar{\mathbf{y}}_{tm}^q, \mathbf{c}_{tm}^{i_r^r} \rangle. \quad (3.43)$$

Computing Eq. (3.43) is considerably efficient due to the asymmetric distance evaluation approach proposed in [4]. To be specific, we only need to compute the similarity between query video's feature with each codeword $\langle \bar{\mathbf{y}}_{tm}^q, \mathbf{c}_{tm}^k \rangle$ for one time and then store the similarity in a look-up table. Computing the similarity between query video V^q and reference video V^r defined in Eq. (3.43) only need $M(T+1)$ times similarity table look-ups.

	UCF101			HMDB51		
Quantization?	$T = 1$	$T = 2$	$T = 3$	$T = 1$	$T = 2$	$T = 3$
NO	0.8547	0.8642	0.8622	0.4628	0.4927	0.4945
YES	0.8639	0.9035	0.9023	0.5352	0.6012	0.5801

Table 3.10: Regularization

	UCF101				HMDB			
	6 bits	12 bits	18 bits	36 bits	6 bits	12 bits	18 bits	36 bits
LSH	0.090	0.220	0.326	0.521	0.068	0.116	0.149	0.231
SH	0.234	0.448	0.613	0.777	0.155	0.255	0.344	0.440
ITQ	0.282	0.588	0.728	0.828	0.206	0.360	0.449	0.521
SPBE	0.266	0.532	0.717	0.805	0.209	0.327	0.408	0.512
KSH	0.450	0.752	0.842	0.878	0.325	0.436	0.532	0.572
DSH	0.513	0.807	0.856	0.882	0.331	0.457	0.546	0.586
Ours	0.626	0.864	0.888	0.904	0.535	0.563	0.591	0.601

Table 3.11: Comparison with existing hashing methods.

3.3.3 Experiments

Dataset we use for evaluating our method are two public benchmark datasets for action recognition: UCF101 [37] and HMDB51 [38]. **UCF101** dataset consists of 101 categories containing 13320 total realistic videos. **HMDB51** dataset contains 6766 clips divided into 51 categories, each containing a minimum of 101 clips. Both UCF101 and HMDB51 provide 3 training/testing splits. Following [76], we use the third split of UCF101 and first split of the HMDB51. The training data not only is used for training the network but also serves as the reference videos in the database for retrieval. The testing data are the query videos. A reference video is related to the query video if they share the same semantic label.

Backbone network we adopt is an old-fashioned network, BN-Inception [101],

	SUBIC [83]			DHCM [76]			TPQN (Ours)		
	64	128	256	64	128	256	12	18	36
UCF101	0.324	0.432	0.449	0.759	0.817	0.843	0.864	0.888	0.904
HMDB51	0.192	0.247	0.298	0.356	0.367	0.368	0.563	0.591	0.601

Table 3.12: Comparison with deep video hashing methods.

Figure 3.13: The influence of α .

but we remove its last fully-connected layer and softmax layer. To suppress overfitting, we add a dropout layer with ratio $r = 0.8$ after the last layer of the backbone network. Even though there are many more advanced deep learning architectures available, we select BN-Inception due to the limitation of our computing resources. Laterly, we will show that the performance achieved by ours using BN-Inception is considerably better than another work [76] using a deeper ResNet50 as the backbone. Despite that a temporal stream network taking optical flow as input can achieve higher performance in action retrieval, we do not do that since the baselines we compare only take the RGB frames as input.

Training details. On both testing datasets, we set N , the number of sampled frames per video, as 9. The batch size is set to be 128. The initial learning rate is 0.001 and the learning rate will be divided by 10 after every 30 epochs and the training process finishes in 120 epochs. We use SGD as the optimizer and set the momentum as 0.9. The loss function used in training the model is standard cross-entropy loss.

3.3.3.1 Ablation Study

Influence of T . We vary T among $\{1, 2, 3\}$. Note that when $T = 1$, it will be equivalent to a one-level global max-pooling. On both datasets, we fix $M = 1$ and increase the number of codewords K from 32 to 2048. As shown in Figure 3.11, when $T = 2, 3$, it consistently outperform that when $T = 1$. Nevertheless, a higher T will take more bits, leading to a higher memory and computation cost. To balance the precision and efficiency, we set $T = 2$ by default on both datasets. Meanwhile, from Figure 3.11 we can also observe that as K increases, the mAP increases in the early stage and then drops. The increase of mAP is due to that a higher K is capable of representing richer information. In contrast, the mAP decreases when $K > 128$ is caused by over-fitting. We will show in section 3.3.3.2 that the quantization can serve as a regularization mechanism which suppresses the over-fitting. How to select K is dependent on the scale of the dataset. As a rule of thumb, we pick a larger K for a larger dataset. As shown in Figure 3.11, when T varies among $\{1, 2, 3\}$, it consistently achieves the best performance when $K = 128$ on UCF101 dataset and when $K = 64$ on HMDB51 dataset.

Influence of M . A larger M brings a richer codebook as well as a greater complexity. We vary M among $\{1, 2, 4\}$. On both datasets, we set $T = 2$ and increase K from 32 to 2048. In this scenario, the number of bits required for representing a video is $3M \log_2 K$. As shown in Figure 3.12, when K is small, $M = 2, 4$ achieves much better performance than $M = 1$, this is expected since a small K will have a limited capability of representing and therefore need a larger M to enrich the codewords. On the contrary, when K is large, the advantage of $M = 2, 4$ over $M = 1$ is not so obvious. Meanwhile, in consistency with the previous experimental results shown in Figure 3.11, when M varies among $\{1, 2, 4\}$, it continuously achieves the highest precision on UCF101 dataset when $K = 128$ and on HMDB51 when $K = 64$.

Influence of α . α controls the consistency between the approximation function $f_{tm}^1(\cdot, \alpha)$ and original quantization function $q_{tm}(\cdot)$. We fix $T = 2$ and $M = 2$ and test our TPQN by increasing α from 2 to 10 and vary K among $\{32, 64, 128\}$. As shown in Figure 3.13, the performance, when α is within the range $[5, 10]$, the performance of TPQN is considerably stable. Nevertheless, we can observe a significant performance drop when α decreases from 5 to 2. This drop is due to the fact that a small α brings a large inconsistency between the approximation function $f_{tm}^1(\cdot, \alpha)$ in the training phase and the hard quantization in the indexing phase. Meanwhile, on the HMDB51 dataset, we also observe a slight performance drop when α increases from 7 to 10. This performance drop is caused by the fact that a large α will tend to make the training unstable. By default, we set $\alpha = 5$ on both datasets.

3.3.3.2 Complexity Analysis and Regularization

In this section, we analyze the complexity reduction brought by quantization and also show the regularization functionality of the quantization. We will show that our TPQN not only significantly boosts the efficiency but also serves as a regularization mechanism, improving the precision. To make a fair comparison, we compare with a baseline by directly removing the quantization operation, *i.e.*, the third step in temporal product quantization layer and keep other parts of TPQN fixed. As for our TPQN, we set $M = 2$ and $K = 64$ on both datasets.

Without quantization, the video features are high-dimension real-value vectors. For instance, when $T = 2$, the feature dimension is $D(T + 1) = 1024 \times 3 = 3072$ and it takes $3072 \times 4 = 12288$ bytes to store the feature in a float-type array. In contrast, using the quantization, it only takes $(T + 1)M \lceil \log_2 K / 8 \rceil = 6$ bytes, achieving an approximate $2000\times$ memory reduction.

Meanwhile, we find the quantization also significantly improves the retrieval

precision as shown in Table 3.10. This is due to the regularization mechanism brought by the quantization, which suppresses overfitting and improves the generalization capability of the trained model. For instance, when $T = 2$, our TPQN achieves 0.9035 mAP on UCF101 dataset and 0.6012 mAP on HMDB51 dataset. In contrast, after removing the quantization, it only achieves 0.8642 mAP on UCF101 dataset and 0.4927 mAP on HMDB51 dataset.

3.3.3.3 Comparison with state-of-the-art methods

To further demonstrate the effectiveness of the proposed TPQN, we compare it with the state-of-the-art methods. The compared methods can be categorized into two types. The first type of methods are based on a two-step process: video feature extraction followed by hashing. To make a fair comparison, we directly use the features without quantization for hashing. We implement multiple hashing methods including LSH [21], SH [23], ITQ [24], SPBE [102], KSH [51], SDH [50]. Among them, LSH [21], SH [23], ITQ [24] and SPBE [102] are unsupervised hashing methods, whereas KSH [51] and SDH [50] are supervised hashing methods. As shown in Table 3.11, our method consistently outperforms other methods on both UCF101 and HMDB51 datasets, especially when bit length is small. For instance, on UCF101 dataset, our method achieves a 0.864 mAP when bit length is only 12 whereas the second best DSH only achieves a 0.807 mAP using the same bit length. On HMDB51 dataset, our method achieves a 0.535 mAP when bit length is only 6 whereas the second best DSH only achieves a 0.331 mAP.

The second type of methods are deep video hashing methods. We mainly compare with two most recent methods, SUBIC [83] and Deep Hashing with Category Mask (DHCM) [76]. Note that even if SUBIC and DHCM are based on a deeper ResNet50 backbone, ours still consistently outperforms both of them using a shallower backbone BN-Inception as shown in Table 3.12. To be specific, on the UCF101

Figure 3.14: Visualization of retrieval results from the proposed TPQN.

dataset, we achieve a 0.904 mAP using only 36 bits, whereas DHCM only achieves a 0.843 using 256 bits. Meanwhile, on the HMDB51 dataset, we achieve a 0.563 mAP using only 12 bits, whereas DHCM only achieves a 0.368 using 256 bits. Figure 3.14 visualizes a retrieval result from the proposed TPQN.

3.3.4 Summary

In this section, we propose a temporal product quantization network (TPQN). It is the first work incorporating frame feature learning, frame feature aggregation and video feature product quantization in a single network, which supports an end-to-end training. Through constructing an approximated function, it makes the product quantization continuously differentiable and makes back-propagation feasible. Meanwhile, by exploiting the temporal consistency inherited in the video, we extend the original product quantization to temporal product quantization. Moreover, we discover that, the quantization mechanism not only reduces the memory and computation complexity in video retrieval but also serves as a regularization approach, boosting the generalization capability of the model and improving the retrieval precision.

Chapter 4

Fast Object Instance Search in Images and Video Frames

Despite that significant progress has been achieved in whole image retrieval, object instance search is relatively less exploited. In the task of object instance search in images, the query is an object instance and it seeks to retrieve all the reference images in the database containing the query instance. One of challenges in the object instance search is that the query instance might only occupy a small portion in its relevant reference items. Therefore, directly comparing the query’s feature with an reference item’s global feature might not precisely capture their relevance. To tackle this challenge, a straightforward solution is to crop the reference images into multiple candidate regions using object proposals. The relevance between the query object and a reference image can be obtained by the the similarity between the query and its best-matched proposal. Meanwhile, the best-matched object proposal of the reference image provides the user with the location of the query instance. Nevertheless, utilizing object proposals significantly increases computational cost. In this chapter, we exploit the redundancy among the huge number of object proposals to speed up the object instance search in images and video

Figure 4.1: Local convolutional features of each proposal are extracted from the last convolutional layer of CNN and they are further pooled or aggregated into a global feature.

frames.

4.1 Fuzzy Object Matching

Given a set of reference images, the ultimate task of object instance search is to identify a subset of reference images containing the similar instances of objects to the query. Sometimes, users are also interested in the location of the object instance within the underlying reference image. In order to achieve this goal, we design an effective representation scheme which simultaneously considers the effectiveness and efficiency of the matching phase. We then discuss the proposed fuzzy objects encoding and matching in details.

4.1.1 Fuzzy Object Encoding

Given an image I , a set of object proposals $\mathcal{P} = \{p_i\}_{i=1}^n$ are generated by Edge Boxes [34]. The extracted object proposals will represent the potential object candidates in the reference image. The flow of feature extraction for each object proposal p_i is illustrated in Figure 4.1. Particularly, after being resized into a fixed scale, each object proposal p_i will be fed into the convolutional neural network (CNN) [103] and a tensor $\mathbf{T}_i \in \mathbb{R}^{w \times h \times d}$ will be obtained from last convolutional layer of CNN. The tensor $\mathbf{T}_i$ is further split into a set of local features $\mathcal{T}_i = \{\mathbf{t}_{im} \in$

$\mathbb{R}^d\}_{m=1}^{wh}$. Finally, $\mathbf{p}_i$, the global deep feature for p_i is generated by pooling or aggregating the local convolutional features in $\mathcal{T}_i$. In this paper, four types of aggregation or pooling methods: max-pooling, sum-pooling, bilinear-pooling [104] and vector of locally aggregated descriptors (VLAD) [105] are implemented for local convolutional features pooling/aggregation to generate the object proposal representation and their performance are evaluated respectively in the experiment section.

We denote by $\mathbf{P} = [\mathbf{p}_1, \dots, \mathbf{p}_n] \in \mathbb{R}^{d \times n}$ the ℓ_2 -normalized d -dimensional features of n object proposals from the reference image. Given a query represented by ℓ_2 -normalized feature $\mathbf{q}$, the similarity scores of all the object proposals from the reference image can be computed by

$$\mathbf{s} = \mathbf{q}^\top \mathbf{P}. \quad (4.1)$$

In order to efficiently obtain the similarity scores for the object proposals, we group the features of all the proposals $\{\mathbf{f}_i\}_{i=1}^n$ from the same image into t clusters $\{\mathcal{C}_l\}_{l=1}^t$ by k-medoids clustering [1] as shown in Figure 4.2, which achieves better performance than k-means clustering in our experiments. Given the ℓ_2 -normalized centroid of cluster $\mathcal{C}_l$, a fuzzy object $\mathbf{o}_l$ is obtained by

$$\begin{aligned} \mathbf{c}_l &= \frac{1}{|\mathcal{C}_l|} \sum_{\mathbf{f} \in \mathcal{C}_l} \mathbf{f}, \\ \mathbf{o}_l &= \mathbf{c}_l / \|\mathbf{c}_l\|_2. \end{aligned} \quad (4.2)$$

We treat the set of fuzzy objects $\mathbf{O} = [\mathbf{o}_1, \dots, \mathbf{o}_t] \in \mathbb{R}^{d \times t}$ as a dictionary and further learn a matrix consisting of sparse codes $\mathbf{H} = [\mathbf{h}_1, \dots, \mathbf{h}_n] \in \mathbb{R}^{t \times n}$ such that

$$\mathbf{P} \approx \mathbf{O}\mathbf{H}. \quad (4.3)$$

Figure 4.2: Hundreds of object proposals tend to overlap with each other. Through clustering, the object proposals containing the similar objects will be assigned to the same group.

Figure 4.3: Hundreds of object proposals can be approximated by a compact set of fuzzy objects product the corresponding sparse codes.

We visualize this process in Figure 4.3. While this process is similar to [3] in spirit, the fundamental difference is that the proposed encoding process being at the image level instead of the dataset level, offers a more compact object description scheme. The image-specific fuzzy objects encoding scheme is designed to extract a sparse yet discriminative object proposal representation. It can successfully address the problem of handling a huge number of object proposals without loss of search precision, as will be shown in experiments.

Different from sparse coding used in [3] which learns the dictionary by solving the ℓ_1 -norm penalty optimization problem, the fuzzy objects are simply learned by k-medoids clustering. At the same time, we reconstruct the feature of the object proposal using only the fuzzy objects which are close to the object proposal in the feature space. This process is closely related to Locality-constrain Linear Coding (LLC) [106]. However, LLC is employed to extract features for classification. In contrast, we use it to achieve efficient object instance search. Given the feature of the object proposal $\mathbf{p}_i$, we use its z nearest neighbours in fuzzy objects set $\mathbf{O}$ as the local bases $\mathbf{O}_i$ and obtain the codes by solving the linear system

$$\min_{\mathbf{h}_i \in \mathbb{R}^z} \|\mathbf{p}_i - \mathbf{h}_i \mathbf{O}_i\|_2 \quad s.t., \quad \mathbf{1}^\top \mathbf{h}_i = 1. \quad (4.4)$$

The solution of Eq. (4.4) can be derived analytically by

$$\begin{aligned} \hat{\mathbf{h}}_i &= (\mathbf{O}_i - \mathbf{p}_i \mathbf{1}^\top)(\mathbf{O}_i - \mathbf{p}_i \mathbf{1}^\top)^\top \backslash \mathbf{1}, \\ \mathbf{h}_i &= \hat{\mathbf{h}}_i / (\mathbf{1}^\top \hat{\mathbf{h}}_i). \end{aligned} \quad (4.5)$$

4.1.2 Fuzzy Objects Matching

After obtaining the fuzzy object sets $\mathbf{O}$ and sparse codes $\mathbf{H}$, the estimated similarity scores between the query object and all the object proposals from the reference

image is computed by

$$\tilde{\mathbf{s}} = \mathbf{q}^\top \mathbf{O}\mathbf{H}. \quad (4.6)$$

The relevant between the query object and the reference image is then determined by the maximum item in the estimated similarity scores vector $\tilde{\mathbf{s}}$:

$$R(q, I) = \max_{l=1, \dots, n} \tilde{s}(l). \quad (4.7)$$

The reference images in the dataset will further be ranked according to their relevance scores.

The complexity of computing $\mathbf{q}^\top \mathbf{O}\mathbf{H}$ is only $\mathcal{O}(dt + zn)$, where z is the number of non-zero elements in each sparse code. Due to the information redundancy observed in the generated proposals from an image, the number of clusters t is chosen to be much smaller compared with the total number of proposals n . Typically $t \sim 0.1n$ works excellent in our experiments. In addition, the codes are very sparse and the typical choices of z is less than $0.01d$. This yields a significant reduction in computation and memory cost. We will show that the object instance search precision obtained by the FOM is comparable with exhaustive matching the query object with all the object proposals but with much less memory and computational cost.

4.1.3 Fuzzy Objects Refinement

In order to further improve the accuracy of object instance search, we refine the features of all the fuzzy objects of all the reference images according to their neighborhood information. In our neighborhood refinement scheme, every fuzzy object $\mathbf{o}_l$ is treated as a pseudo query and m most similar fuzzy objects $\{\mathbf{o}_l^k\}_{k=1}^m$ in the dataset are retrieved to refine the feature of the fuzzy object. The refined fuzzy object $\tilde{\mathbf{o}}_l$ is computed by

$$\tilde{\mathbf{o}}_l = \frac{(\mathbf{o}_l + \sum_{k=1}^m \mathbf{o}_l^k)}{m+1}. \quad (4.8)$$

The proposed neighborhood refinement follows the spirit of the average query expansion [107]. However, the average query expansion only refines the feature of the query which is conducted online when the query comes. In contrast, the proposed neighborhood refinement scheme is to refine the features of the fuzzy objects which can be carried out offline and does not effect the search time. It is worth noting that our neighborhood refinement can work in parallel with average query expansion to further boost object instance search precision. In the experiment section, we will evaluate the influence of neighborhood refinement and average query expansion, respectively.

In fact, $\mathbf{o}_l^k$ can be generated from different reference images in the dataset. As $\tilde{\mathbf{o}}_l$ generated from Eq.(4.8) fuses the information from different reference images, we term $\tilde{\mathbf{o}}_l$ as **inter-image fuzzy object**. In contrast, the fuzzy object $\mathbf{o}_l$ in Eq.(4.2) are generated from fusing the features of proposals from the same reference image, therefore, we term $\mathbf{o}_l$ as **intra-image fuzzy object**. After $\tilde{\mathbf{o}}_l$ is generated, the approximated similarity between query and all proposals will be computed by

$$\tilde{\mathbf{s}} = \mathbf{q}^\top \tilde{\mathbf{O}} \mathbf{H}, \quad (4.9)$$

where $\tilde{\mathbf{O}} = [\tilde{\mathbf{o}}_1, \dots, \tilde{\mathbf{o}}_t]$ represent the refined fuzzy objects. Here, the neighbourhood refinement only adjusts $\mathbf{O}$ for each reference image and keeps the codes $\mathbf{H}$ unchanged.

The location of each object proposal of each reference image is stored in the database. We will localize the query object in the reference image using the location of the estimated best-matched object proposal.

4.1.4 Experimental Results

In this section, we carry out comprehensive experiments on three benchmark datasets, *i.e.*, Oxford5K [39], Paris6K [40] and Sculptures6k [41]. Object search performance of the proposed framework is evaluated by mean average precision (mAP) which is widely used in evaluating the effectiveness of the image retrieval system.

In this work, the CNN model we used is the 16-layer VGG network [103] pre-trained on the ImageNet dataset. Each proposal is resized to the size 448×448 prior to being fed into the network. The 512-dimensional local features are extracted from the last convolutional layer conv5_3. In this scenario, the spatial size of conv5_3 layer is $w \times h = 28 \times 28$.

We evaluate our FOM scheme using features from four different aggregation methods: max-pooling, sum-pooling, bilinear-pooling and VLAD. To be more specific, the cluster number of VLAD is set to be 64 and we conduct intra-normalization proposed in [105] to post-process VLAD features. In the VLAD and bilinear pooling aggregations, the dimension of the local convolutional feature is reduced into 64 by principle component analysis (PCA). Finally, all the global features generated from max-pooling, sum-pooling, bilinear-pooling and VLAD are further processed by PCA and whitening followed by ℓ_2 normalization and the dimensions of four types of features are fixed as 512.

4.1.4.1 Compared with Exhaustive Object Proposals Matching

In Table 4.1, we evaluate the performance of the proposed Fuzzy Objects Matching (FOM) in comparison with Exhaustive Object Proposals Matching (EOPM) which directly compares the query object with all the object proposals of all the reference images as Eq.(4.1). We also show the performance from the method using global feature to represent the whole image. It is worth noting that the fuzzy objects we

Method	EOPM						FOM				GF
#	10	20	40	80	160	300	5	10	20	40	1
Max	49.5	57.7	62.8	68.3	71.9	73.9	68.8	72.1	73.2	74.5	31.5
Sum	57.5	62.9	67.3	71.1	75.1	77.0	65.1	69.7	72.2	74.4	40.9
Bilinear	50.8	56.4	63.2	68.9	72.5	74.5	59.7	65.4	67.8	71.5	37.6
VLAD	57.1	62.1	67.0	71.2	74.3	76.5	66.2	70.7	72.7	75.1	37.8

Table 4.1: The performance comparison of EOPM, FOM and the method using global features (GF) on the Oxford5K dataset.

evaluated in this section is the intra-image fuzzy objects without neighborhood refinement. Since our intra-image FOM is an estimation of EOPM, the performance of EOPM should be better than that of the intra-image FOM.

We can observe from Table 4.1 that the proposed FOM scheme using tens of fuzzy objects (FOs) can achieve comparable mAP with EOPM scheme using 300 object proposals (OPs) in object search. Moreover, comparing the mAP of EOPM using 20 object proposals with the mAP of FOM using 20 fuzzy objects, we find the FOM is much more effective than the EOPM when the memory and computational cost are comparable. Besides, the mAP of the method using global features is much lower than EOPM and FOM, which verifies the effectiveness of the object proposals.

The larger number of the fuzzy objects brings the higher precision as well as higher memory and computational cost. To balance the effectiveness and efficiency, we choose the default number of fuzzy objects as 20. Therefore, for each image, it requires 20×512 dimensions to store the fuzzy objects, $300 \times 3 \times 2$ dimensions to store the value and indices of non-zero elements in $\mathbf{H}$ and 4×300 dimensions to cache the locations of 300 object proposals. In total, for each reference image, it demands only around 13k dimensions to store all the information required by the FOM. In contrast, given 300 object proposals, the dimensions required by the EOPM for each reference image is around 150k, which is much larger than that from the FOM.

4.1.4.2 Compared with Other Fast Strategies

In this section, we compare the proposed FOM method with other two alternative fast strategies. They are Representative Object Matching and Sparse Coding Matching.

For each reference image, Representative Object Matching (ROM) scheme selects representative proposals from all the object proposals by k-medoids [1]. To be more specific, the features of n object proposals in the matrix $\mathbf{P} \in \mathbb{R}^{d \times n}$ are grouped into t clusters by k-medoids algorithm and the medoids of the clusters are selected to be the representatives of the n object proposals. In this scenario, we only need to compare the query object with the selected object proposals and both the computational and memory cost are reduced from $\mathcal{O}(nd)$ to $\mathcal{O}(td)$.

Sparse Coding Matching (SCM) [3] factorizes matrix $\mathbf{P}$ by sparse coding. The dictionary $\mathbf{D} \in \mathbb{R}^{d \times t}$ is learned by solving the following ℓ_1 -penalized optimization:

$$\min_{\mathbf{D}} \|\mathbf{P} - \mathbf{DC}\|_F^2 + \lambda \|\mathbf{C}\|_{1,1}. \quad (4.10)$$

The sparsity of codes $\mathbf{C}$ is strictly controlled by orthogonal matching pursuit and the number of non-zero elements in each column of $\mathbf{C}$ is fixed as z . Computing $\mathbf{q}^\top \mathbf{DC}$ only requires $\mathcal{O}(td+nz)$ time complexity which is as efficient as the proposed FOM.

We define complexity ratio as the computational complexity of fast strategies over the computational complexity of exhaustive object proposals matching. In this case, the complexity ratio of ROM is t/n and the complexity ratio of FOM and SCM is $t/n + z/d$.

Figure 4.4 compares the performance of FOM with other two fast strategies using max-pooling features on Oxford5K and Paris6K datasets respectively. In the implementation, we fix the number of non-zero elements z in both FOM and

Figure 4.4: The performance comparison of FOM, SCM and ROM on Oxford5K and Paris6K datasets. FOM gives the best performance.

SCM as 3 and changes the number of atoms t from 5 to 20. We can observe that the performance of FOM is much better than ROM. At the same time, the proposed FOM is also better than SCM, which is attributed to the locality-constrained property of the proposed FOM.

4.1.4.3 Intra-image Fuzzy Objects Versus Inter-image Fuzzy Objects

To validate the performance of the proposed neighborhood refinement, we compare the performance of intra-image FOM and inter-image FOM using max-pooling, sum-pooling, bilinear-pooling and VLAD features. The results are shown in Table 4.2.

In the experiment, the number of proposals n to generate fuzzy objects is set as 300, the number of fuzzy objects t is set as 20 and the number of neighborhoods m in Eq.(4.8) is set as 20. As can be seen from the Table 4.2, the performance of inter-image FOM outperforms that from intra-image FOM in all of four features on these datasets. For example, on the Oxford5K dataset, inter-image FOM improves the mAP of Max-pooling features from 73.2 to 83.1. On the Paris6K dataset, inter-image FOM improves the mAP of VLAD feature from 83.1 to 89.0.

Dataset	Oxford5K		Paris6K	
Feature	Intra	Inter	Intra	Inter
Max	73.2	83.1	74.6	83.8
Sum	72.2	76.4	77.6	83.5
Bilinear	67.8	74.8	74.3	80.7
VLAD	72.7	80.4	83.1	89.0

Table 4.2: The performance comparison of intra-image FOM and inter-image FOM on Oxford5K and Paris6K datasets.

Dataset	Oxford5K		Paris6K	
Feature	Inter	Inter+AQE	Inter	Inter+AQE
Max	83.1	88.9	83.8	90.7
Sum	76.4	84.8	83.5	88.5
Bilinear	74.8	82.3	80.7	85.4
VLAD	80.4	88.7	89.0	92.5

Table 4.3: The performance of Average Query Expansion on Oxford5K and Paris6K datasets.

4.1.4.4 Average Query Expansion

We conduct average query expansion (AQE) [107] to further improve the performance of the FOM scheme. For each query q , s fuzzy objects closest to q in the feature space are retrieved. We further compute the mean of feature of query q and features of the s fuzzy objects. The generated mean vector will serve as the new feature of the query in order to re-rank the reference images in the database. Table 4.3 shows the performance of the proposed search system with and without AQE. In our implementation, we set s as 20, which is equal to the number of fuzzy objects. As it can be seen, AQE can effectively improve the performance of the system. For example, on the Oxford5K dataset, it improves the mAP of max-pooling feature by 5.8. On the Paris6K dataset, it improves the mAP of VLAD feature by 3.5.

4.1.4.5 Comparison with State-of-the-art Methods

The first part of Table 4.4 shows the object instance search performance from methods using hand-crafted local features. [8] achieved 41.8 mAP by improved Fisher vector encoding method on the Oxford5K dataset and [105] achieved 55.5 mAP on the Oxford5K dataset by aggregating SIFT features using VLAD. In contrast, ours can achieve 88.9 mAP using max-pooling features. In [108], the authors proposed selective matching kernel scheme and achieved excellent performance on both oxford5K and Paris6K datasets with high memory cost. Ours outperforms them with less memory and computation cost.

The second part of Table 4.4 compares our method with other methods using CNN features. Neural Codes [10] adopt finetuned deep features and achieved 55.7 mAP on Oxford5K dataset. SPoc [11] achieved 65.7 mAP on Oxford5K dataset by sum-pooling the local convolutional feature. [12] aggregated the local convolutional features using VLAD. Note that [10, 11, 12] all represented the whole image as a global feature and thus might be incapable of handling the scenarios when there are multiple objects in the reference image. We can observe in the Table 4.4 that the methods in [10, 11, 12] are not competitive with that from [65, 67, 71].

In [109, 65], the authors cropped both reference images and query images into patches and the relevance between the query and reference images are conducted by cross-matching between patches. Our methods outperform both of them, for example, we achieved 88.1 mAP on the Oxford5K dataset, whereas the mAP of [65] is only 84.4. Moreover, we should point out that, the cross-matching scheme in [109] is much less efficient than ours. For example, it need conduct 1024 comparisons between 32 patches from the query and 32 patches from the reference image. In contrast, we only need 20 comparisons between the query and 20 fuzzy objects from the reference image. In [67, 68], the authors conducted re-ranking process through spatial search in reference images and achieved much better per-

Method	Dimension	Oxford5K	Paris6K	Sculptures6K
Hand-crafted Local Features				
Perronnin <i>et al.</i> [8]	2k	41.8	-	-
Arandjelovic <i>et al.</i> [105]	32k	55.5	-	-
Arandjelovic <i>et al.</i> [41]	-	-	-	45.4
Mikulik <i>et al.</i> [110]	64k	74.2	74.9	-
Tao <i>et al.</i> [79]	-	77.8	-	-
Tolias <i>et al.</i> [108]	64k	80.4	77.0	-
CNN Features				
Babenko <i>et al.</i> [10]	256	55.7	-	-
Babenko <i>et al.</i> [11]	256	65.7	-	-
Ng <i>et al.</i> [12]	128	59.3	59.0	-
Razavian <i>et al.</i> [109]	15k	68.0	79.5	42.3
Razavian <i>et al.</i> [65]	32k	84.4	85.3	67.4
Tolias <i>et al.</i> [67]	-	77.3	86.5	-
Mohedano <i>et al.</i> [68]	-	78.8	84.8	-
Cao <i>et al.</i> [71]	-	78.1	87.4	60.8
Our FOM Scheme				
Max _{Inter} +AQE	13k	88.9	90.7	65.1
Sum _{Inter} +AQE	13k	84.8	88.5	65.8
Bilinear _{Inter} +AQE	13k	82.3	85.4	59.4
VLAD _{Inter} +AQE	13k	88.7	92.5	73.1

Table 4.4: Performance comparisons with state-of-the-art methods on Oxford5K, Paris6K and Sculptures6K datasets.

formance than global representation methods [10, 12, 11]. A most recent work [71] achieved 78.1 mAP on the Oxford5k dataset , 87.4 mAP on the Paris6K dataset and 60.8 mAP on the Sculptures6K dataset. In contrast, our FOM with VLAD feature achieved 88.7 mAP on the Oxford5K dataset, 92.5 mAP on the Paris6K dataset and 73.1 mAP on the Sculptures6K dataset. Some visual results of our FOM scheme using VLAD features are shown in Figure 4.5. We show the estimated best-matched object proposal to validate the performance of our scheme in object localization.

(a) Oxford5K

(b) Paris6K

(c) Sculptures6K

Figure 4.5: Selected results of top-10 retrievals of our FOM scheme on the Oxford5K and Paris6K datasets. The bounding box in the reference image correspond to the estimated best-matched object proposal.

4.1.5 Summary

In this chapter, we propose the Fuzzy Objects Matching (FOM) scheme in order to achieve efficient and effective instance search. We conduct comprehensive experiments using object proposal representation generated from four different aggregation methods on three public datasets. The state-of-the-art precision and the efficient implementation verifies the outstanding performance of our method.

4.2 Hierarchical Object Prototype Encoding

Driven by the emergence of large-scale data sets and fast development of computation power, features based on convolutional neural networks (CNN) have proven to perform remarkably well on visual search [10, 11, 111, 112]. In this work, we exploit CNN features to represent object proposals. Given a query object represented by $\mathbf{q} \in \mathbb{R}^d$, directly computing the similarity between the query and m object proposals $\mathbf{P}_i \in \mathbb{R}^{d \times m}$ from the frame f_i is unaffordable because both the number of object proposals m and the number of the frames n are large. Therefore, we design two types of sphere k-means methods, *i.e.*, spatially-constrained sphere (SCS) k-means and temporally-constrained sphere (TCS) k-means to accelerate the object instance search without sacrificing the search accuracy.

In order to exploit the self-similarity property in the spatial domain, for the frame f_i , SCS k-means learns frame-level object prototypes $\mathbf{C}_i \in \mathbb{R}^{d \times t_1} (t_1 \ll m)$ and factorizes $\mathbf{P}_i$ into $\mathbf{C}_i \mathbf{H}_i$ where each column of $\mathbf{H}_i$ is the frame-level sparse code for the object proposal. Similarly, to exploit the self-similarity property in temporal domain, TCS k-means further factorizes all the frame-level object prototypes $[\mathbf{C}_1, \mathbf{C}_2, \dots, \mathbf{C}_n]$ into $\mathbf{G}[\mathbf{E}_1, \mathbf{E}_2, \dots, \mathbf{E}_n]$, where $\mathbf{G} \in \mathbb{R}^{t_2} (t_2 \ll nt_1)$ denotes the dataset-level object prototypes and $\mathbf{E}_i$ denotes the dataset-level sparse codes for the frame-level object prototypes from frame f_i . Therefore, the object proposals

Figure 4.6: The framework of our hierarchical object prototype encoding.

$\mathbf{P}_i$ will be hierarchically decomposed into $\mathbf{G}\mathbf{E}_i\mathbf{H}_i$. As shown in Figure 4.6, the features of the object proposals from frame f_i are first decomposed into the product of frame-level object prototypes $\mathbf{C}_i$ and the frame-level sparse codes $\mathbf{H}_i$ illustrated by dashed lines. The frame-level object prototypes of all the frames $[\mathbf{C}_1, \dots, \mathbf{C}_n]$ are further decomposed into the product of dataset-level object prototypes $\mathbf{G}$ and the dataset-level sparse codes $[\mathbf{E}_1, \dots, \mathbf{E}_n]$ depicted by solid lines.

In this scenario, rather than directly storing the features of all the object proposals of all the frames $[\mathbf{P}_1, \dots, \mathbf{P}_n] \in \mathbb{R}^{d \times nm}$, we only need to store the dataset-level object prototypes $\mathbf{G} \in \mathbb{R}^{d \times t_2}$, the dataset-level sparse codes $\{\mathbf{E}_i\}_{i=1}^n$ and the frame-level sparse codes $\{\mathbf{H}_i\}_{i=1}^n$. Since $t_2 \ll nm$ and both $\mathbf{E}_i$ and $\mathbf{H}_i$ are sparse, the proposed HOPE model leads to a dramatic reduction in terms of memory and computational cost. In fact, as demonstrated in our experiments, the memory cost of storing the HOPE model is less than 1% of that of storing $\{\mathbf{P}_i\}_{i=1}^n$.

4.2.1 Problem Statement

Given a query represented by the ℓ_2 -normalized feature $\mathbf{q} \in \mathbb{R}^d$ and a dataset consisting of n frames $\{f_i\}_{i=1}^n$, the object instance search is to retrieve all the relevant frames and localize the query object in the relevant frames. In most cases, the interested object normally takes up a small area in the whole frame. To capture the relevance of the frame with the query, for each frame f_i , we extract m object proposals serving as potential regions of the query object in the frame. We denote by $\mathbf{p}_{ij} \in \mathbb{R}^d$ the ℓ_2 -normalized feature of the j -th object proposal from the i -th frame and denote by $\mathbf{P}_i = [\mathbf{p}_{i1}, \dots, \mathbf{p}_{im}] \in \mathbb{R}^{d \times m}$ the ℓ_2 -normalized features of all the object proposals from the i -th frame. In this scenario, the relevance score of the frame f_i with the query is determined by the similarity $S(\mathbf{q}, \mathbf{p}_{ij})$ between the query and the best-matched object proposal in the frame:

$$R(f_i) = \max_j S(\mathbf{q}, \mathbf{p}_{ij}) = \max_j \mathbf{q}^\top \mathbf{p}_{ij}. \quad (4.11)$$

Here, the best-matched object proposal is used to localize the query object in the relevant frame.

In order to obtain the relevance scores of all the frames, we need to compare the query with all the object proposals in all the frames, which is equivalent to calculating

$$\mathbf{s}^0 = \mathbf{q}^\top [\mathbf{P}_1, \dots, \mathbf{P}_n]. \quad (4.12)$$

It should be emphasized that computing $\mathbf{s}^0$ in Eq. (4.12) requires $\mathcal{O}(nmd)$ complexity. In this case, the key problem of the object instance search is how to efficiently obtain $\mathbf{s}^0$.

4.2.2 Frame-level Object Prototype Encoding

As mentioned in Section 4.2, the proposals from the same frame tend to overlap with each other which brings a great amount of redundancy. Redundancy means that there exists self-similarity among the object proposals. To exploit the self-similarity property, we propose to factorize the features $\mathbf{P}_i$ of object proposals from the frame f_i using the frame-level object prototypes $\mathbf{C}_i$. The formulation is given by

$$\mathbf{P}_i \rightarrow \mathbf{C}_i \mathbf{H}_i, \quad (4.13)$$

where $\mathbf{C}_i = [\mathbf{c}_{i1}, \dots, \mathbf{c}_{it_1}] \in \mathbb{R}^{d \times t_1}$ is generated by our spatially-constrained sphere k-means and $\mathbf{H}_i = [\mathbf{h}_{i1}, \dots, \mathbf{h}_{im}] \in \mathbb{R}^{t_1 \times m}$ consists of frame-level sparse codes of the object proposals generated from soft-assignment coding. We will introduce how to solve $\mathbf{C}_i$ and $\mathbf{H}_i$ in details.

Spatially-constrained Sphere (SCS) K-means is an extension of sphere k-means [6] by taking the spatial information of the object proposals into consideration. The spatial relationship can provide valuable information. Intuitively, if two object proposals are heavily overlapped, they are inclined to share the objects. For convenience, we drop the index i in $\mathbf{P}_i$, $\mathbf{C}_i$, $\mathbf{H}_i$, which identifies a specific frame because the following spatially-constrained sphere k-means is applied to each frame independently. We denote by $\mathcal{B} = \{b_1, \dots, b_m\}$ the bounding boxes of the object proposals. We develop the spatially-constrained sphere (SCS) k-means which iteratively updates the frame-level object prototypes (ℓ_2 -normalized centroids of the clusters) and the assignment of the object proposals by

$$\begin{aligned}
Assign : A_u &= \arg \max_k \mathbf{p}_u^\top \mathbf{c}_k + \frac{\lambda_s}{|\mathcal{A}_k|} \sum_{v \in \mathcal{A}_k} IoU(b_u, b_v), \\
Update : \mathbf{c}_k &= \sum_{v \in \mathcal{A}_k} \mathbf{p}_v / \left\| \sum_{v \in \mathcal{A}_k} \mathbf{p}_v \right\|_2,
\end{aligned} \tag{4.14}$$

where A_u denotes the cluster index which u -th object proposal is assigned to, $\mathbf{c}_k$ denotes the k -th frame-level object prototypes, $\mathcal{A}_k$ denotes the set of object proposals assigned to the k -th cluster and $IoU(\cdot, \cdot)$ computes the intersection over union ratio of two bounding boxes. λ_s is the parameter controlling the weight of the spatial constraint. When $\lambda_s = 0$, the SCS k-means will degrade to the original sphere k-means. The iteration stops when the frame-level object prototypes do not change or the maximum iteration number is reached. Actually, our experiments show that SCS k-means converges within 20 iterations.

The spatial information is especially useful to handle the situation in which multiple different but similar instances appear in the frame. Figure 4.7 visualizes the clustering results of the object proposals from the sphere k-means and the spatially-constrained sphere k-means. It can be observed that the sphere k-means groups the object proposals containing different types of cars into the same clustering (red bounding boxes). In contrast, the spatially-constrained sphere k-means groups the object proposals containing different cars into different clusterings (red bounding boxes and purple bounding boxes).

Given an object proposal $\mathbf{p} \in \mathbb{R}^d$ and the frame-level object prototypes generated from spatially-constrained sphere k-means $\mathbf{C} \in \mathbb{R}^{d \times t_1}$, the soft-assignment coding (SAC) first finds z closest object prototypes of $\mathbf{p}$ which we denote by $\hat{\mathbf{C}} = [\mathbf{c}_{k_1}, \dots, \mathbf{c}_{k_z}] \in \mathbb{R}^{d \times z}$ and further obtains the codes $\mathbf{h} \in \mathbb{R}^{t_1}$ by

Figure 4.7: The comparison between sphere k-means and SCS k-means.

$$\hat{\mathbf{h}}(i) = \begin{cases} \exp(\beta \mathbf{c}_i^\top \mathbf{p}), & i = k_1, \dots, k_z \\ 0, & \text{otherwise} \end{cases} \quad (4.15)$$

$$\mathbf{h} = \hat{\mathbf{h}} / \hat{\mathbf{h}}^\top \mathbf{1},$$

where β controls the softness of the assignment. The sparsity of the codes from SAC is strictly controlled by z . The soft-assignment coding was traditionally used as a feature extraction method for image classification [113] while we use it as a sparse coding method to achieve high efficiency for object instance search.

Complexity Analysis in Search Phase. Both the SCS k-means and the soft-assignment coding can be conducted offline, which does not affect the search time. In the search phase, the similarity scores for all the object proposals of all the frames can be efficiently achieved by

$$\mathbf{s}^1 = \mathbf{q}^\top [\mathbf{C}_1 \mathbf{H}_1, \dots, \mathbf{C}_n \mathbf{H}_n]. \quad (4.16)$$

We denote by z_1 the number of non-zeros elements in each column of $\mathbf{H}_i$. Computing $\mathbf{s}^1$ in Eq. (4.16) only requires $\mathcal{O}(nt_1d + nmz_1)$ complexity. This is because we can first compute $[\mathbf{x}_1, \dots, \mathbf{x}_n] = \mathbf{q}^\top [\mathbf{C}_1, \dots, \mathbf{C}_n]$ taking $\mathcal{O}(nt_1d)$ complexity and

then compute $\mathbf{s}^1 = [\mathbf{x}_1 \mathbf{H}_1, \dots, \mathbf{x}_n \mathbf{H}_n]$ which is a sparse matrix-vector multiplication (SpMV) problem [114] taking $\mathcal{O}(nmz_1)$ complexity. In fact, $t_1 \ll m$ and $z_1 \ll d$, which means $nt_1d + nmz_1 \ll nmd$. Therefore, computing $\mathbf{s}^1$ is much more efficient than directly computing $\mathbf{s}^0$ in Eq. (4.12). In addition, to store the sparse matrix $\mathbf{H}_i$, we only need to store its non-zero elements. Therefore, the memory cost for storing $\{\mathbf{C}_i\}_{i=1}^n$ and $\{\mathbf{H}_i\}_{i=1}^n$ is only $\mathcal{O}(nt_1d + nmz_1)$ which is much less than that of storing $\{\mathbf{P}_i\}_{i=1}^n$. However, when n is huge considering the fact that a video consists of enormous frames, the computational cost of calculating $\mathbf{s}^1$ as well as the memory cost of storing $\{\mathbf{C}_i\}_{i=1}^n$ and $\{\mathbf{H}_i\}_{i=1}^n$ are still considerable. It motivates us to exploit the self-similarity across the frames further.

4.2.3 Dataset-level Object Prototype Encoding

It is well known that the consecutive frames are very similar to each other. To further speed up the object instance search in videos, we propose to encode the frame-level object prototypes again using the self-similarity property across multiple different frames. We denote by $[\mathbf{C}_1, \dots, \mathbf{C}_n] \in \mathbb{R}^{d \times nt_1}$ the frame-level object prototypes of all the frames in the dataset. The dataset-level prototype encoding factorizes $[\mathbf{C}_1, \dots, \mathbf{C}_n]$ into $\mathbf{G}\mathbf{E}$, where $\mathbf{G} = [\mathbf{g}_1, \dots, \mathbf{g}_{t_2}] \in \mathbb{R}^{d \times t_2}$ comprises dataset-level object prototypes generated from the proposed temporally-constrained sphere k-means and $\mathbf{E} = [\mathbf{E}_1, \dots, \mathbf{E}_n] = [\mathbf{e}_{11}, \dots, \mathbf{e}_{1t_1}, \mathbf{e}_{21}, \dots, \mathbf{e}_{nt_1}] \in \mathbb{R}^{t_2 \times nt_1}$ is composed of the dataset-level sparse codes generated from the soft-assignment coding. In what follows, we show the whole encoding flow of the proposed HOPE model.

Algorithm 1 Hierarchical Object Prototype Encoding

Input: Object proposals from n frames $\{\mathbf{P}_i\}_{i=1}^n$, the number of frame-level object prototypes per frame t_1 , the number of dataset-level object prototypes t_2 , the number of non-zero elements in each frame-level code z_1 , the number of non-zero elements in each dataset-level code z_2 .

Output: Dataset-level object prototypes $\mathbf{G}$, dataset-level codes $\{\mathbf{E}_i\}_{i=1}^n$, frame-level codes $\{\mathbf{H}_i\}_{i=1}^n$.

```

1: for  $i = 1 \dots n$ 
2:    $\mathbf{C}_i \leftarrow \text{SCSKmeans}(\mathbf{P}_i, t_1)$  using Eq. (4.14)
3:    $\mathbf{H}_i \leftarrow \text{SAC}(\mathbf{P}_i, \mathbf{C}_i, z_1)$  using Eq. (4.15)
4: end
5:  $\mathbf{G} \leftarrow \text{TCSKmeans}([\mathbf{C}_1, \dots, \mathbf{C}_n], t_2)$  using Eq. (4.17)
6: for  $i = 1 \dots n$ 
7:    $\mathbf{E}_i \leftarrow \text{SAC}(\mathbf{G}, \mathbf{C}_i, z_2)$  using Eq. (4.15)
8: end
9: return  $\mathbf{G}, \{\mathbf{E}_i\}_{i=1}^n, \{\mathbf{H}_i\}_{i=1}^n$ 

```

Temporally-constrained Sphere (TCS) K-means. In a video, the temporal relationship among the frames can also provide valuable information. One common sense is that the consecutive frames will be prone to contain the similar objects. We thus utilize the temporal information to weakly supervise the clustering.

We divide the frames into multiple groups according to their temporal location. For example, given a long video consisting of M frames, we can equally divide the M frames into τ groups. According to their group assignment, the frame-level object prototypes will be divided into multiple temporal chunks $\{\mathcal{S}_1, \dots, \mathcal{S}_\tau\}$. Sometimes, the dataset also provides the shot information, *i.e.*, it indicates which frame is from which shot. In this case, we can directly arrange the frame-level object prototypes from the same video shot in the same temporal chunk. The temporally-constrained sphere k-means iteratively updates the dataset-level object prototypes and the assignment of the frame-level object prototypes by

$$\begin{aligned}
Assign : A_u &= \arg \max_k \mathbf{c}_u^\top \mathbf{g}_k + \lambda_t \frac{|\mathcal{A}_k \cap \mathcal{S}_{\mathbf{c}_u}|}{|\mathcal{A}_k|}, \\
Update : \mathbf{g}_k &= \sum_{v \in \mathcal{A}_k} \mathbf{c}_v / \left\| \sum_{v \in \mathcal{A}_k} \mathbf{c}_v \right\|_2,
\end{aligned} \tag{4.17}$$

where $\mathcal{S}_{\mathbf{c}_u}$ denotes the temporal chunk containing the u -th frame-level object prototype $\mathbf{c}_u$. $\mathcal{A}_k \cap \mathcal{S}_{\mathbf{c}_u}$ denotes the set of the frame-level object prototypes assigned to the k -th cluster and $\mathcal{S}_{\mathbf{c}_u}$. λ_t is the parameter controlling the weight of the temporal constraint. The temporally-constrained sphere k-means will reduce to the original sphere k-means when λ_t is 0. The iteration stops when the dataset-level object prototypes do not change or the maximum iteration number is reached. Our experiments show that the iteration can converge within 100 iterations.

Complexity Analysis in Search Phase. Utilizing the generated dataset-level object prototypes $\mathbf{G}$, the dataset-level sparse codes $\{\mathbf{E}_i\}_{i=1}^n$ and the frame-level sparse codes $\{\mathbf{H}_i\}_{i=1}^n$, the similarity scores for all the proposals of all the frames can be obtained by

$$\mathbf{s}^2 = \mathbf{q}^\top \mathbf{G} [\mathbf{E}_1 \mathbf{H}_1, \dots, \mathbf{E}_n \mathbf{H}_n]. \tag{4.18}$$

We denote by z_2 the number of non-zero elements in each column of $\mathbf{E}_i$ and denote by z_1 the number of non-zero elements in each column of $\mathbf{H}_i$. We decompose computing $\mathbf{s}^2$ into following three steps:

$$\begin{aligned}
1) : \mathbf{x} &\leftarrow \mathbf{q}^\top \mathbf{G} \quad \text{Complexity : } \mathcal{O}(t_2 d) \\
2) : [\mathbf{y}_1, \dots, \mathbf{y}_n] &\leftarrow [\mathbf{x} \mathbf{E}_1, \dots, \mathbf{x} \mathbf{E}_n] \quad \text{Complexity : } \mathcal{O}(nt_1 z_2) \\
3) : \mathbf{s}^2 &\leftarrow [\mathbf{y}_1 \mathbf{H}_1, \dots, \mathbf{y}_n \mathbf{H}_n] \quad \text{Complexity : } \mathcal{O}(nm z_1)
\end{aligned} \tag{4.19}$$

Both step 2) and step 3) in Eq. (4.19) are the sparse matrix-vector multiplication (SpMV) problem [114] which take $\mathcal{O}(nt_1z_2)$ and $\mathcal{O}(nmz_1)$, respectively. In total, the complexity of computing $\mathbf{s}^2$ is $\mathcal{O}(t_2d + nt_1z_2 + nmz_1)$. Since $t_2 \ll nt_1$ and $z_2 \ll d$, computing $\mathbf{s}^2$ is much more efficient than computing $\mathbf{s}^1$. Meanwhile, the total memory cost of storing $\mathbf{G}$, $\{\mathbf{E}_i\}_{i=1}^n$ and $\{\mathbf{H}_i\}_{i=1}^n$ is $\mathcal{O}(t_2d + nt_1z_2 + nmz_1)$, which is much less than that of storing $\{\mathbf{C}_i\}_{i=1}^n$ and $\{\mathbf{H}_i\}_{i=1}^n$ in computing $\mathbf{s}^1$.

4.2.4 Non-exhaustive Search

The above analysis shows that computing $\mathbf{s}^2$ takes $\mathcal{O}(t_2d + nt_1z_2 + nmz_1)$ complexity. In the practical cases, $t_1 \ll m$ and $t_2d \ll nmz_1$. Therefore, the most time-consuming part of computing $\mathbf{s}^2$ is multiplying the sparse codes $\mathbf{h}_{ij}$ for each object proposal. To further improve the efficiency, we propose the non-exhaustive search scheme to avoid comparing the query with all the object proposals of all the frames. Since the frame-level object prototypes are the ℓ_2 -normalized centroids of the clusters of object proposals, they represent different types of objects in the frame. In other words, they can be utilized as the representatives of the object proposals. Therefore, the relevance score of the frame with the query can be more efficiently calculated by the best-matched frame-level object prototype:

$$R(f_i) = \max_{j=1, \dots, t_1} (\mathbf{q}^\top \mathbf{c}_{ij}) \approx \max_{j=1, \dots, t_1} (\mathbf{q}^\top \mathbf{G} \mathbf{e}_{ij}). \quad (4.20)$$

We rank all the frames in the dataset through the frame-level relevance scores computed by Eq. (4.20). A candidate list of frames $\mathcal{F} = [f_{r1}, \dots, f_{rs}]$ will be obtained by selecting a portion of frames having high rankings. In this case, we only need to conduct spatial localization of the query object in the candidate list $\mathcal{F}$ by

$$\mathbf{s}^3 = \mathbf{q}^\top \mathbf{G} [\mathbf{E}_{r1} \mathbf{H}_{r1}, \dots, \mathbf{E}_{rs} \mathbf{H}_{rs}]. \quad (4.21)$$

Table 4.5: The complexity analysis of different schemes.

Scheme	Complexity
$\mathbf{s}^0$ in Eq. (4.12)	$\mathcal{O}(mnd)$
$\mathbf{s}^1$ in Eq. (4.16)	$\mathcal{O}(nt_1d + nmz_1)$
$\mathbf{s}^2$ in Eq. (4.18)	$\mathcal{O}(t_2d + nt_1z_2 + nmz_1)$
$\mathbf{s}^3$ in Eq. (4.21)	$\mathcal{O}(t_2d + nt_1z_2 + \alpha nmz_1)$

The best-matched object proposal of each frame will serve as the detected region of the query object in the frame. Computing $\mathbf{s}^3$ only requires $\mathcal{O}(t_2d + nt_1z_2 + \alpha nmz_1)$ complexity, where α is the ratio of the frames for further localizing the query object. It is much more efficient than computing $\mathbf{s}^2$ as $\alpha \ll 1$. Overall, the frame-level object prototypes $\mathbf{C}_i$ serve two roles: (1) the representatives of object proposals to efficiently determine the relevance score of the frame. (2) the prototypes to encode the features of object proposals to speed up the object localization in the relevant frames. In addition, the frame-level codes $\{\mathbf{H}_i\}_{i=1}^n$ are only used for object localization in the frame and irrelevant with the relevance scores of the frames. Table 4.5 summarizes the complexity of different schemes. m is the number of object proposals per frame, n is the number of frames. d is the feature dimension. $t_1 \ll m$ is the number of frame-level object prototypes per frame. $t_2 \ll nt_1$ is the number of dataset-level object prototypes. z_1 and z_2 denote the number of non-zero elements in frame-level codes and dataset-level codes, respectively.

4.2.5 Experiments

4.2.5.1 Settings and Datasets

In this paper, we adopt Edge Boxes [34] to generate object proposals serving as the potential regions of the object instance. For each proposal, we extract its feature by max-pooling the last convolutional layer of VGG-16 CNN model [103] pre-trained on the Imagenet dataset. The max-pooled 512-dimensional features are further post-processed by principle component analysis (PCA) and whitening

Figure 4.8: Query objects visualization.

in order to suppress the burstiness [115] but the dimension of the feature is kept as 512. The effectiveness of the proposed method is evaluated by mean average precision (mAP).

We conduct the systematic experiments on the Groundhog Day [42] and NTU-VOI [43] datasets. Groundhog Day dataset contains 5640 keyframes and six types of small query objects: Red clock, Microphone, Black clock, Frame sign, Phil sign and Digital clock. NTU-VOI dataset contains 37340 frames and we use five types of query objects: Ferrari, Kittyb, Kittyg, Maggi and Plane to evaluate the proposed method. Figure 4.8 visualizes the query objects.

4.2.5.2 Effectiveness of Object Proposals

In this section, we evaluate the effectiveness of the object proposals. The first algorithm (Baseline I) extracts features for the whole frame and represents each frame with a global feature. In this scenario, the frames are ranked according to their cosine similarity to the query object. The second algorithm (Baseline II) exhaustively compares the query object with all the object proposals of all the frames, illustrated in Eq. (4.12). In Baseline II, the best-matched proposal of each frame, *i.e.*, the proposal with the highest similarity scores, determines the relevance between the query and the frame. In other words, the frames are ranked according to the similarity scores of their best-matched proposals.

Figure 5.1 shows the object instance search performance of the Baseline I and Baseline II. It can be observed that the performance of Baseline II is much better

Figure 4.9: The mAP of two baseline algorithms on Groundhog day and NTU-VOI datasets.

than Baseline I. At the same time, the mAP of baseline II improves as the number of object proposals n increases. However, the increase rate is slower and slower. It makes sense as in most cases, hundreds of object proposals are enough to capture the location of the object instance in the frames. To balance the effectiveness and efficiency, we set the default number of object proposals as 300. We will use Baseline II algorithm as the reference to further evaluate the proposed hierarchical object prototype encoding method.

4.2.5.3 Frame-level Object Prototype Encoding

The proposed hierarchical object prototype encoding consists of two levels. We first evaluate the effectiveness of the frame-level object prototypes encoding and do not conduct the dataset-level object prototype encoding. We adopt $\mathbf{s}^1$ scheme in Eq. (4.16) where we set the number of nonzero elements in frame-level sparse codes z_1 as 3.

Figure 4.10 compares the performance of the frame-level object prototypes generated from k-means, sphere k-means and spatially-constrained sphere k-means, respectively. In our implementation, the object prototypes are generated from 300 object proposals and λ_s is fixed as 3 on both datasets. It can be observed that, in

Figure 4.10: The mAP comparison between K-means, Sphere K-means and SCS K-means on Groundhog Day and NTU-VOI datasets.

general, the sphere k-means is slightly better than k-means. In comparisons, the proposed spatially-constrained sphere k-means is better than the sphere k-means. Interestingly, when the number of frame-level object prototypes per frame t_1 is over 30, the performance achieved by the frame-level object prototypes is even better than baseline II. Although a larger number of frame-level object prototypes can bring higher mAP, the default number of the object prototypes is set as 30 to balance the accuracy and efficiency.

Comparison with Representative Selection. We compare the performance of the frame-level object prototypes with the object representatives selected from K-medoids [1] and the SMRS method [2] which select representative object proposals from hundreds of object proposals for each frame. In the representative selection scheme, we only need to compare the query object with the selected representative objects and the exhaustive search can be avoided. The fundamental difference of the proposed frame-level object prototypes from the object representatives is that the object representatives are selected from the original data samples whereas the object prototypes are generated from the centroids of the clusters which are more robust. From Figure 4.11, we can see that the performance of the proposed object prototypes scheme is much better than that from representative

Figure 4.11: Comparison of the proposed frame-level object prototypes with the object representatives generated from K-medoids [1] and SMRS [2].

objects generated from SMRS and K-medoids.

4.2.5.4 Effectiveness of HOPE

In this section, we evaluate the whole HOPE model. We adopt the non-exhaustive search scheme in Eq. (4.21) where we set α as 0.05. We denote by Baseline III the implementation in Section 4.2.5.3 where the frame-level object prototypes are generated from the SCS k-means, the number of frame-level object prototypes $t_1 = 30$ and the number of non-zero elements in frame-level sparse codes $z_1 = 3$. We fix the settings of frame-level prototype encoding in HOPE model exactly same as Baseline III and focus on testing dataset-level object prototype encoding. On the Groundhog Day dataset, we equally divide the 5640 keyframes into 50 groups to generate temporal chunks required by the TCS k-means. On the NTU-VOI dataset, we directly use the shot information provided by the dataset to obtain temporal chunks. On both datasets, we set the $\lambda_t = 2$.

We first compare the performance of the dataset-level object prototypes using the sphere k-means and the TCS k-means, where the number of non-zero elements in the codes z_2 is set as 3. As depicted in Figure 4.12, the performance of dataset-level object prototypes generated from the TCS k-means is significantly better

Figure 4.12: The mAP comparison between sphere k-means and TCS k-means on Groundhog Day and NTU-VOI datasets.

than that of generated from the sphere k-means. Surprisingly, Figure 4.12 shows that the mAP of from our HOPE model is much better than both Baseline II and Baseline III, which can be attributed to the denoising effect of the HOPE model. It is worth noting that 800 dataset-level object prototypes can achieve 0.70 mAP on the Groundhog day dataset. In this scenario, storing the HOPE model ($\{\mathbf{H}_i\}_{i=1}^n, \{\mathbf{E}_i\}_{i=1}^n, \mathbf{G}$) only requires 28.64 MB whereas the Baseline II costs 3.22 GB to store $\{\mathbf{P}_i\}_{i=1}^n$. On the NTU-VOI dataset, 100 dataset-level object prototypes can achieve 0.84 mAP, which only requires 176.47MB to store the HOPE model whereas the Baseline II costs 21.36 GB. On both datasets, the memory cost from HOPE is less than 1% of that of Baseline II.

Comparison with Sparse Coding. We compare our object prototype encoding with sparse coding method proposed in [3] to further validate the effectiveness of the HOPE. To make a fair comparison, both the sparse coding and the dataset-level object prototype encoding are conducted on the exactly same frame-level object prototypes generated from SCS K-means and we fix z_2 as 3 for both sparse coding and dataset-level object prototype encoding. From Figure 4.13, we can see the proposed dataset-level object prototype encoding is much better than that of sparse coding used in [3].

Figure 4.13: The performance comparison between sparse coding [3] and our dataset-level object prototype encoding.

Comparison with Quantization. In order to further validate the effectiveness of the dataset-level object prototype encoding, we compare it with product quantization (PQ) [4], optimized product quantization (OPQ) [5], and product quantization with coefficients (α -PQ) [6]. In the quantization scheme, we quantize all the frame-level object prototypes into the dataset-level quantizers and the relevance score of the frame-level object prototypes can be efficiently obtained from the look-up table. In the implementation of PQ, OPQ and α -PQ, we vary m , the number of sub-groups, from 2 to 16. We first set the number of clusters of each sub-group as 256 which is the standard settings of PQ. Then we set the number of clusters of each sub-group as 800 on the Groundhog day dataset and 100 on the NTU-VOI dataset which are the same settings as our dataset-level object prototype encoding. We can observe from Figure 4.14 that the proposed dataset-level object prototype encoding significantly outperforms PQ, OPQ and α -PQ.

Efficiency Comparison. We compare the memory and time cost of the proposed prototype encoding (ours) with Sparse Coding (SC) and Product Quantization (PQ) in Table 4.6. In implementation, we set $z_2 = 3$ and $t_2 = 800$ for ours and SC. In PQ, we set the number of sub-codebooks m as 4 and the sub-codebook size is 800. As we can see from Table 4.6 that ours significantly outperforms SC

Figure 4.14: Comparison of our dataset-level object prototype encoding with product quantization [4], optimized product quantization [5] and product quantization with coefficients [6].

and PQ in precision with comparable memory and time cost.

Method	Memory	Search Time	mAP
Ours	28.6MB	218 ms	0.70
SC	28.6MB	218 ms	0.55
PQ	27.0MB	212 ms	0.62

Table 4.6: Efficiency comparison on the Groundhog Day.

Parameters Sensitivity. Figure 4.15 shows the influence of the parameters. The mAP is relatively stable when $\lambda_t \in [1, 5]$. The bigger z_2 brings better mAP but higher memory and computation cost. Particularly, when $z_2 = 1$, the soft-assignment becomes hard-assignment and it achieves worse mAP than the soft-assignment when $z_2 > 1$. To balance the precision and efficiency, we set $z_2 = 3$. When $\beta = \infty$, the soft-assignment will become hard-assignment, so we should not set β to be too large. We set the default value of β as 3.

Search Results Visualization. Finally, we visualize the search results from the proposed HOPE model in Figure 4.16 and Figure 4.17. We can see that the query object only occupies a very small area surrounded by very dense clutter but our method can accurately find the frames containing the query object and precisely localize it in the frames.

Figure 4.15: The influence of parameters on the Groundhog Day.

Figure 4.16: Visualization of top-16 search results on the Groundhog dataset with query black clock and microphone.

Figure 4.17: Visualization of top-16 search results on the NTU-VOI dataset with query Kittyb and Maggi.

4.2.6 Summary

In this section, we leverage the object proposals to improve the quality of object instance search in videos. To address the large memory and computational cost of using object proposals, we propose to formulate the search into a hierarchical sparse coding problem. By utilizing the spatial and temporal self-similarity property of object proposals, we present SCS K-means and TCS K-means to learn the frame-level object prototypes and dataset-level object prototypes, respectively. Experimental results on two video datasets demonstrate that our approach can achieve even better performance than exhaustive search using all object proposals within a fraction of complexity and significantly outperform other state-of-the-art fast methods.

Chapter 5

Fast Object Instance Search in Video Clips

In this chapter, we address the object instance search in video clips. Since a video clip is composed of several frames, the object instance search in video clips can be converted into object instance search in frames and solved by the proposed methods in the Chapter 4. But in this task, we are not interested in the temporal locations of the object instance in videos, providing more opportunities to further speed up the search.

5.1 Compressive Quantization

We propose to utilize the spatio-temporal redundancy of the video-data to enable fast object-to-video (*i.e.*, point-to-set) search. For each video clip in the database, we can represent it as a collection of object proposals generated from its frames. However, instead of storing all the object proposals from one video, we propose compressive quantization to compress the set of object proposals into only a few binary codes, where each binary code is a compact representative of the object proposals. As a compact representation of a set of object proposals, our compressive

Figure 5.1: The overview of the proposed compressive quantization algorithm.

quantization targets at **(1)** quantizing the set of object proposals into fewer codewords and **(2)** hashing the codewords into binary codes. Nevertheless, instead of following a two-step optimization, we propose a joint optimization framework that can simultaneously optimize the distortion errors from quantization and hashing. It directly compresses M object proposals into k binary codes, where $k \ll M$.

As our compressive quantization simultaneously performs vector quantization and hashing, it is different from either conventional vector quantization or hashing methods. On one hand, different from conventional vector quantization [116], our compressive quantization constrains the codewords to be binary codes to achieve higher compression ratio. On the other hand, unlike hashing methods [21, 23, 117, 118] mapping one real-value vector to one binary code, our compressive quantization summarizes many real-value vectors into only a few binary codes, leading to a more compact representation of the set compared with traditional hashing methods.

As illustrated in Figure 5.1, using the proposed compressive quantization, a video consisting of $m \times t$ object proposals can be compactly represented by k ($\ll m \times t$) binary codes. To measure the relevance of a video with the query, we only need to compare the binary code of the query with k binary codes representing the video. It not only reduces the number of distance computations ($m \times t \rightarrow k$) but also enables fast search leveraging binary codes, which boosts the efficiency in

both time and memory.

5.1.1 A Baseline by Quantization + Hashing

We denote by $\mathcal{V}_i$ the i -th video in the database containing m frames $\{f_i^1, \dots, f_i^m\}$. For each frame f_i^j , we crop t object proposals. Therefore, the video $\mathcal{V}_i$ will be cropped into a set of $m \times t$ object proposals $\mathcal{S}_i = \{\mathbf{o}_i^1, \dots, \mathbf{o}_i^{m \times t}\}$. Given a query $\mathbf{q}$, the distance between $\mathcal{V}_i$ and $\mathbf{q}$ can be determined by the best-matched object proposal in $\mathcal{S}_i$:

$$D(\mathbf{q}, \mathcal{V}_i) = \min_{\mathbf{o} \in \mathcal{S}_i} \|\mathbf{q} - \mathbf{o}\|_2. \quad (5.1)$$

To obtain $D(\mathbf{q}, \mathcal{V}_i)$ in Eq. (5.1), it requires to compare the query with all the object proposals in $\mathcal{S}_i$. Nevertheless, it ignores the redundancy among the object proposals in $\mathcal{S}_i$ and leads to heavy computation and memory cost.

Quantization. To exploit the redundancy among the object proposals for reducing the memory cost and speeding up the search, one straightforward way is to quantize $m \times t$ object proposals in $\mathcal{S}_i$ into k codewords $\mathcal{Z}_i = \{\mathbf{z}_i^1, \dots, \mathbf{z}_i^k\}$. In this case, we only need to compare the query $\mathbf{q}$ with codewords rather than all the object proposals in $\mathcal{S}_i$. One standard vector quantization method is k-means, which seeks to minimize the quantization errors Q_i :

$$Q_i = \sum_{u=1}^k \sum_{v \in C_i^u} \|\mathbf{o}_i^v - \mathbf{z}_i^u\|_2^2, \quad (5.2)$$

where C_i^u denotes the set of indices of the object proposals in $\mathcal{S}_i$ that are assigned to the u -th cluster. Then the distance between $\mathbf{q}$ and $\mathcal{V}_i$ can be calculated by

$$D(\mathbf{q}, \mathcal{V}_i) = \min_{\mathbf{z} \in \mathcal{Z}_i} \|\mathbf{q} - \mathbf{z}\|_2. \quad (5.3)$$

Since $|\mathcal{Z}_i| \ll |\mathcal{S}_i|$, computing Eq. (5.3) is much faster than computing Eq. (5.1).

Hashing. In practice, due to the number of videos n in the dataset is huge and the codewords are represented by high-dimensional real-value vectors, it is still time consuming to compare the query with all the codewords. To further speed up the search, we seek to map the codewords to Hamming space by hashing. One popular hashing method is ITQ [117], which learns the binary codes $[\mathbf{b}_1^1, \mathbf{b}_1^2, \dots, \mathbf{b}_n^k] \in \{-1, 1\}^{l \times nk}$ and the transform matrix $\mathbf{R} \in \mathbb{R}^{d \times l}$ through minimizing the distortion error H :

$$H = \sum_{i=1}^n \sum_{u=1}^k \|\mathbf{z}_i^u - \mathbf{R}^\top \mathbf{b}_i^u\|_2^2, \quad \text{s.t.} \quad \mathbf{R}^\top \mathbf{R} = \mathbf{I}. \quad (5.4)$$

We denote by $\mathcal{B}_i = \{\mathbf{b}_i^1, \dots, \mathbf{b}_i^k\}$ the set of binary codes for the video $\mathcal{V}_i$ and denote by $\mathbf{b}_q = \text{sign}(\mathbf{R}\mathbf{q})$ the binary code of the query $\mathbf{q}$. In this scenario, the distance between $\mathbf{q}$ and the video $\mathcal{V}_i$ can be determined by the Hamming distance ($\|\cdot\|_H$) between the query's binary code $\mathbf{b}_q$ and the nearest binary code in $\mathcal{B}_i$:

$$D(\mathbf{q}, \mathcal{V}_i) = \min_{\mathbf{b} \in \mathcal{B}_i} \|\mathbf{b}_q - \mathbf{b}\|_H. \quad (5.5)$$

Limitations. The above two-step optimization sequentially minimizes the quantization errors of object proposals caused by quantization and the distortion errors of the codewords caused by hashing. Nevertheless, the above two-step optimization can only yield a suboptimal solution for the following reasons: **(1)** The distortion error H caused by the hashing is not taken into consideration when optimizing quantization errors $\{Q_i\}_{i=1}^n$. **(2)** The quantization errors $\{Q_i\}_{i=1}^n$ are not considered when minimizing the distortion error H . This motivates us to jointly minimize the total errors caused by quantization and hashing in order to achieve a higher search precision.

5.1.2 Compressive Quantization

One straightforward way to jointly optimize the quantization and hashing is to optimize a weighted sum of the quantization errors caused by the quantization and the distortion errors caused by hashing given by:

$$J = \sum_{i=1}^n Q_i + \lambda H, \quad (5.6)$$

where λ is the parameter controlling the weight of distortion error H with respect to the quantization errors $\{Q_i\}_{i=1}^n$. However, the choice of λ is heuristic and dataset dependent. Meanwhile, the accumulated distortion error from two separate operations is not equivalent to a simple summation of errors from each operation.

The above limitations motivate us to propose the compressive quantization scheme which directly minimizes the total distortion error J :

$$\begin{aligned} J = & \sum_{i=1}^n \sum_{u=1}^k \sum_{v \in C_i^u} \|\mathbf{o}_i^v - \alpha \mathbf{R}^\top \mathbf{b}_i^u\|_2^2, \\ \text{s.t. } & \mathbf{R}^\top \mathbf{R} = \mathbf{I}, \quad \mathbf{b}_i^u \in \{-1, +1\}^l, \end{aligned} \quad (5.7)$$

where C_i^u denotes the set of indices of the object proposals assigned to u -th cluster of $\mathcal{S}_i$; $\mathbf{b}_i^u$ denotes the binary codewords representing C_i^u ; $\alpha \in \mathbb{R}^+$ denotes the scaling factor and $\mathbf{R} \in \mathbb{R}^{d \times d}$ denotes the rotation matrix.

Intuitively, we fix the binary codewords as the vertices of the hypercube. As illustrated in Figure 5.2, we rotate and stretch the hypercube to minimize the sum of the squares of the Euclidean distance from each object proposal to the corresponding vertex of the hypercube. Note that, in current formulation, we set the rotation matrix $\mathbf{R}$ as the square matrix, *i.e.*, we fix the code length l to be equal to the original feature's dimension d . Later on, we will discuss how to extend the

Figure 5.2: We rotate and stretch the blue hypercube to minimize the distance from each object proposal to the corresponding vertex of the hypercube the object proposal is assigned to.

current algorithms to the scenario when $l \neq d$.

It is easy to see the above optimization problem is highly non-convex and difficult to solve. But it is tractable to solve the problem by updating one variable with others fixed:

- (i) Fix $\{C_i^u\}_{i=1, u=1}^{n, k}$, $\mathbf{R}$ and α , update $\{\mathbf{b}_i^u\}_{i=1, u=1}^{n, k}$.
- (ii) Fix $\{C_i^u\}_{i=1, u=1}^{n, k}$, $\{\mathbf{b}_i^u\}_{i=1, u=1}^{n, k}$ and α , update $\mathbf{R}$.
- (iii) Fix $\{C_i^u\}_{i=1, u=1}^{n, k}$, $\mathbf{R}$ and $\{\mathbf{b}_i^u\}_{i=1, u=1}^{n, k}$, update α .
- (iv) Fix $\mathbf{R}$, $\{\mathbf{b}_i^u\}_{i=1, u=1}^{n, k}$ and α , update $\{C_i^u\}_{i=1, u=1}^{n, k}$.

5.1.2.1 Fix $\{C_i^u\}_{i=1, u=1}^{n, k}$, $\mathbf{R}$ and α , update $\{\mathbf{b}_i^u\}_{i=1, u=1}^{n, k}$.

When C_i^u , $\mathbf{R}$ and α are fixed, the optimal value of $\mathbf{b}_i^u$ can be obtained by solving the below optimization problem:

$$\begin{aligned}
 \mathbf{b}_i^u &= \underset{\mathbf{b} \in \{-1, 1\}^{d \times 1}}{\operatorname{argmin}} \sum_{v \in C_i^u} \|\mathbf{o}_i^v - \alpha \mathbf{R}^\top \mathbf{b}\|_2^2 \\
 &= \underset{\mathbf{b} \in \{-1, 1\}^{d \times 1}}{\operatorname{argmin}} \sum_{v \in C_i^u} \|\alpha^{-1} \mathbf{x}_i^v - \mathbf{b}\|_2^2,
 \end{aligned} \tag{5.8}$$

where $\mathbf{x}_i^v = \mathbf{R}\mathbf{o}_i^v$. We denote by $b_i^u(t)$ the value of t -th bit of the above optimized $\mathbf{b}_i^u$ and denote by $x_i^u(t)$ the value of t -th dimension of $\mathbf{x}_i^u$. Below we will prove that

$$b_i^u(t) = \text{sign}\left(\sum_{v \in C_i^u} x_i^v(t)\right). \quad (5.9)$$

Note that $b_i^u(t)$ has only two possible values $\{-1, +1\}$. We denote by $l_{+1}(t) = \sum_{v \in C_i^u} (\alpha^{-1}x_i^v(t) - 1)^2$ the distortion error when the t -th bit of $\mathbf{b}_i^u$ is set to be $+1$ and denote by $l_{-1}(t) = \sum_{v \in C_i^u} (\alpha^{-1}x_i^v(t) + 1)^2$ the distortion error when the t -th bit of $\mathbf{b}_i^u$ is set to be -1 . We can further obtain

$$l_{+1}(t) - l_{-1}(t) = -2\alpha^{-1} \sum_{v \in C_i^u} x_i^v(t). \quad (5.10)$$

Since $\alpha > 0$,

$$\min\{l_{+1}(t), l_{-1}(t)\} = \begin{cases} l_{-1}(t), & \text{if } \sum_{v \in C_i^u} x_i^v(t) < 0 \\ l_{+1}(t), & \text{if } \sum_{v \in C_i^u} x_i^v(t) > 0 \end{cases} \quad (5.11)$$

Thereafter,

$$b_i^u(t) = \begin{cases} -1, & \text{if } \sum_{v \in C_i^u} x_i^v(t) < 0 \\ +1, & \text{if } \sum_{v \in C_i^u} x_i^v(t) > 0 \end{cases} \quad (5.12)$$

5.1.2.2 Fix $\{C_i^u\}_{i=1, u=1}^{n, k}$, $\{\mathbf{b}_i^u\}_{i=1, u=1}^{n, k}$ and α , update $\mathbf{R}$.

We arrange the object proposals of all the videos $\{\mathbf{o}_1^1, \mathbf{o}_1^2, \dots, \mathbf{o}_n^{mt}\}$ as columns in matrix $\mathbf{O} \in \mathbb{R}^{d \times nmt}$. We denote by $\mathbf{B} = [\mathbf{b}_1^{A(\mathbf{o}_1^1)}, \mathbf{b}_1^{A(\mathbf{o}_1^2)}, \dots, \mathbf{b}_n^{A(\mathbf{o}_n^m)}]$ the corresponding binary codes of the object proposals, where $A(\mathbf{o}_i^v)$ denotes the index of the cluster which $\mathbf{o}_i^v$ is assigned to. When $\{C_i^u\}_{i=1, u=1}^{n, k}$, $\{\mathbf{b}_i^u\}_{i=1, u=1}^{n, k}$ and α are fixed, optimizing $\mathbf{R}$ is an orthogonal procrustes problem [119] solved by

$$\mathbf{R} = \mathbf{V}\mathbf{U}^\top, \quad (5.13)$$

where $\mathbf{U}$ and $\mathbf{V}$ are obtained by the singular value decomposition (SVD) of $\mathbf{OB}^\top$. Note that computing $\mathbf{OB}^\top$ takes $\mathcal{O}(nmtdl)$ complexity. Since mt , the number of object proposals per video, is relatively large, this step is considerably time consuming. Below we optimize our algorithm to speed up the process of computing $\mathbf{OB}^\top$. We rewrite $\mathbf{OB}^\top$ into:

$$\begin{aligned} \mathbf{OB}^\top &= \sum_{i=1}^n \sum_{v=1}^m \mathbf{o}_i^v \mathbf{b}_i^{A(\mathbf{o}_i^v)^\top} \\ &= \sum_{i=1}^n \sum_{u=1}^k \sum_{v \in C_i^u} \mathbf{o}_i^v \mathbf{b}_i^{u^\top} \\ &= \left[\sum_{v \in C_1^1} \mathbf{o}_1^v, \dots, \sum_{v \in C_i^u} \mathbf{o}_i^v \right] [\mathbf{b}_1^1, \dots, \mathbf{b}_n^k]^\top \\ &= [\mathbf{y}_1^1, \dots, \mathbf{y}_n^k] [\mathbf{b}_1^1, \dots, \mathbf{b}_n^k]^\top, \end{aligned} \quad (5.14)$$

where $\mathbf{y}_i^u = \sum_{v \in C_i^u} \mathbf{o}_i^v$ has already been obtained in Eq. (5.9). Therefore, we can calculate $\mathbf{OB}^\top$ through $[\mathbf{y}_1^1, \dots, \mathbf{y}_n^k] [\mathbf{b}_1^1, \dots, \mathbf{b}_n^k]^\top$, which takes only $\mathcal{O}(nkdl)$ complexity. Considering the number of codewords (k) per video is significantly smaller than the number of object proposals (mt) per video, the complexity is largely reduced.

5.1.2.3 Fix $\{C_i^u\}_{i=1, u=1}^{n, k}$, $\mathbf{R}$ and $\{\mathbf{b}_i^u\}_{i=1, u=1}^{n, k}$, update α .

We directly compute the first-order and second-order derivative of J with respect to α :

$$\begin{aligned}
\frac{\partial J}{\partial \alpha} &= 2 \sum_{i=1}^n \sum_{u=1}^k \sum_{v \in C_i^u} (-\mathbf{R}^\top \mathbf{b}_i^u)^\top (\mathbf{o}_i^v - \alpha \mathbf{R}^\top \mathbf{b}_i^u), \\
\frac{\partial^2 J}{\partial^2 \alpha} &= 2 \sum_{i=1}^n \sum_{u=1}^k \sum_{v \in C_i^u} (-\mathbf{R}^\top \mathbf{b}_i^u)^\top (-\mathbf{R}^\top \mathbf{b}_i^u).
\end{aligned} \tag{5.15}$$

Since $\partial^2 J / \partial^2 \alpha = 2 \sum_{i=1}^n \sum_{u=1}^k \sum_{v \in C_i^u} \|\mathbf{R}^\top \mathbf{b}_i^u\|_2^2 > 0$, by setting $\partial J / \partial \alpha = 0$, we can obtain the optimal value of the scaling coefficient α which minimizes J :

$$\begin{aligned}
\alpha &= \frac{\sum_{i=1}^n \sum_{u=1}^k \mathbf{b}_i^u{}^\top \sum_{v \in C_i^u} \mathbf{R} \mathbf{o}_i^v}{\sum_{i=1}^n \sum_{u=1}^k \sum_{v \in C_i^u} \mathbf{b}_i^u{}^\top \mathbf{R} \mathbf{R}^\top \mathbf{b}_i^u} \\
&= \frac{\sum_{i=1}^n \sum_{u=1}^k \text{sign}(\sum_{v \in C_i^u} \mathbf{R} \mathbf{o}_i^v)^\top \sum_{v \in C_i^u} \mathbf{R} \mathbf{o}_i^v}{\sum_{i=1}^n \sum_{u=1}^k \sum_{v \in C_i^u} \mathbf{b}_i^u{}^\top \mathbf{b}_i^u} \\
&= \frac{\sum_{i=1}^n \sum_{u=1}^k \|\sum_{v \in C_i^u} \mathbf{R} \mathbf{o}_i^v\|_1}{nmt}.
\end{aligned} \tag{5.16}$$

5.1.2.4 Fix $\mathbf{R}$, $\{\mathbf{b}_i^u\}_{i=1, u=1}^{n,k}$ and α , update $\{C_i^u\}_{i=1, u=1}^{n,k}$.

When $\mathbf{R}$, $\{\mathbf{b}_i^u\}_{i=1, u=1}^{n,k}$ and α are fixed, the optimal $\{C_i^u\}_{i=1, u=1}^{n,k}$ can be obtained by simply assigning each object proposal $\mathbf{o}_i^v$ to its nearest vertex of hypercube by

$$A(\mathbf{o}_i^v) = \underset{u}{\operatorname{argmin}} \|\mathbf{o}_i^v - \alpha \mathbf{R}^\top \mathbf{b}_i^u\|_2. \tag{5.17}$$

5.1.2.5 Initialization and Implementation Details

We initialize $\{C_i^u\}_{i=1, u=1}^{n,k}$ by conducting k-means clustering for $\{\mathcal{S}_i\}_{i=1}^n$ and initialize the rotation matrix $\mathbf{R}$ using the centroids of $\{C_i^u\}_{i=1, u=1}^{n,k}$ through ITQ. The whole optimization process is summarized in Algorithm 2.

Algorithm 2 Compressive Quantization

Input: n sets of object proposals $\{\mathcal{S}_i\}_{i=1}^n$ generated from videos $\{\mathcal{V}_i\}_{i=1}^n$, number of codewords k per video.

Output: The rotation matrix $\mathbf{R}$, the binarized codewords $\{\mathbf{b}_i^u\}_{i=1,u=1}^{n,k}$.

```

1: Initialize the clusters  $\{C_i^u\}_{i=1,u=1}^{n,k}$ ,  $\mathbf{R}$ ,  $\alpha$ .
2: repeat
3:   for  $t = 1 \dots T$ 
4:     Update  $\{\mathbf{b}_i^u\}_{i=1,u=1}^{n,k}$  using Eq. (5.9).
5:     Update  $\mathbf{R}$  using Eq. (5.13).
6:   end
7:   Update  $\alpha$  using Eq. (5.16).
8:   Update  $\{C_i^u\}_{i=1,u=1}^{n,k}$  using Eq. (5.17).
9: until converge
10: return  $\mathbf{R}$ ,  $\{\mathbf{b}_i^u\}_{i=1,u=1}^{n,k}$ .
```

Our algorithm iteratively updates $\{C_i^u\}_{i=1,u=1}^{n,k}$, $\mathbf{R}$, $\{\mathbf{b}_i^u\}_{i=1,u=1}^{n,k}$ and α . Note that there is an inner loop within the outer loop, we set the number of iterations of the inner loop $T = 10$. The aim of the inner loop is to ensure that $\mathbf{R}$ and $\{\mathbf{b}_i^u\}_{i=1,u=1}^{n,k}$ are able to reach their local optimality before updating $\{C_i^u\}_{i=1,u=1}^{n,k}$ and α . Since each update of $\{C_i^u\}_{i=1,u=1}^{n,k}$, $\mathbf{R}$, $\{\mathbf{b}_i^u\}_{i=1,u=1}^{n,k}$ and α can strictly reduce the distortion error J , our algorithm can ensure the convergence. Our experiments show that it can converge within 50 iterations of outer loops on three datasets. Like other alternative-optimization algorithms, our algorithm only achieves a local optima and is influenced by the initialization.

Discussion: In the above formulation, we fix $\mathbf{R}$ as a square matrix, *i.e.*, we constrain the length of code l to the dimension of the original feature d . However, we still can deal with the situation when $l \neq d$ with pre-processing operations.

Particularly, when $l < d$, we will pre-process the original features using principle component analysis (PCA) to reduce their dimensionality from the d to l . After that, we can conduct the proposed compressive composition using the reduced features. It will obtain l -bit binary codes by the square rotation matrix $\mathbf{R} \in \mathbb{R}^{l \times l}$. On the other hand, if $l > d$, we project the original feature $\mathbf{o}$ to l -dimension feature

$\hat{\mathbf{o}}$ by a random orthogonal matrix $\mathbf{P} \in \mathbb{R}^{l \times d}$:

$$\hat{\mathbf{o}} = \mathbf{P}\mathbf{o}, \quad \text{s.t.} \quad \mathbf{P}^\top \mathbf{P} = \mathbf{I}. \quad (5.18)$$

After projection, the Euclidean distance between two points in original feature space $\|\mathbf{o}_1 - \mathbf{o}_2\|_2$ can be preserved in the projected feature space since

$$\|\hat{\mathbf{o}}_1 - \hat{\mathbf{o}}_2\|_2^2 = (\mathbf{o}_1 - \mathbf{o}_2)^\top \mathbf{P}^\top \mathbf{P} (\mathbf{o}_1 - \mathbf{o}_2) = \|\mathbf{o}_1 - \mathbf{o}_2\|_2^2. \quad (5.19)$$

We can further conduct the proposed compressive quantization using the projected data samples to obtain l -bit binary codes by the square rotation matrix $\mathbf{R} \in \mathbb{R}^{l \times l}$.

5.1.2.6 Relation to Existing Methods

K-means [116] is a widely used vector quantization method. It minimizes the quantization errors by iteratively updating the assignments of the data samples and the centroids of the clusters. Our compressive quantization also seeks to minimize the quantization errors, but the codewords generated from our method are binary codes.

Iterative Quantization [117] is a popular hashing method. It learns the rotation matrix by minimizing the distortions between the data samples with their rotated binary codes. Like other hashing methods, it maps a real-value vector $\mathbf{x}$ to a binary code $\mathbf{b}$. The proposed compressive quantization also learns a rotation matrix, but we map a set of real-value vectors $\mathcal{X} = \{\mathbf{x}_1, \dots, \mathbf{x}_M\}$ to another set of binary codes $\mathcal{B} = \{\mathbf{b}_1, \dots, \mathbf{b}_k\}$, where $|\mathcal{B}| \ll |\mathcal{X}|$.

K-means Hashing [120] is another hashing method, which minimizes a weighted summation of the quantization errors caused by k-means and the affinity errors caused by hashing. Its formulation is very similar to Eq. (5.6), which is different from ours formulated as Eq. (5.7). Meanwhile, K-means Hashing maps one real-

value vector to a binary code. In contrast, our compressive quantization maps M real-value vectors to k ($\ll M$) binary codes, which can achieve significantly higher compression ratio.

5.1.3 Experiments

5.1.3.1 Settings, Dataset and Evaluation Metric

We adopt Edge Boxes [34] to generate the object proposals for each frame of the videos. For each frame, we extract 300 object proposals. For each object proposal, we further extract its feature by max-pooling the last convolutional layer of VGG-16 CNN model [103] pre-trained on Imagenet dataset. The max-pooled 512-dimensional features are further post-processed by principle component analysis (PCA) and whitening in order to suppress the burstiness [115] and the dimension of the feature is fixed as 256. Observing the object proposals from the same frame heavily overlap with each other, we use k-means to group 300 object proposals into 30 clusters. We select the centroids of the cluster as the compact object proposals. Given a video of n frames, we will obtain $30n$ object proposals.

We conduct the experiments on Groundhog Day [42], CNN2h [44] and Egocentric [45] datasets. The Groundhog Day contains 5640 keyframes. We equally divide them into 56 short videos. The CNN2h dataset contains 72,000 frames and we equally divide them into 720 short videos. The Egocentric dataset consists of 19 long videos. We uniformly sample the keyframes per 20 frames, obtain 51804 keyframes and further equally divide all the keyframes into 517 videos. The Groundhog Day dataset provides six query objects. On CNN2h and Egocentric datasets, we use eight query objects and skip those scene query images. Figure 5.13 visualizes the query objects of three datasets. The effectiveness of the proposed method is evaluated by mean average precision (mAP).

(a) Groundhog Day

(b) CNN2h

(c) Egocentric

Figure 5.3: Examples of query objects.

		Groundhog Day			CNN2h			Egocentric		
		Memory	Time	mAP	Memory	Time	mAP	Memory	Time	mAP
Baseline1		165.2 MB	1.398 s	0.724	2.060 GB	10.48 s	0.856	1.518 GB	8.826 s	0.974
Baseline2	k = 10	0.547 MB	0.004 s	0.530	7.031 MB	0.035 s	0.625	5.049 MB	0.026 s	0.617
	k = 20	1.094 MB	0.006 s	0.545	14.06 MB	0.069 s	0.674	10.10 MB	0.049 s	0.717
	k = 40	2.188 MB	0.012 s	0.682	28.12 MB	0.147 s	0.817	20.20 MB	0.084 s	0.744
	k = 100	5.469 MB	0.031 s	0.709	70.31 MB	0.314 s	0.845	50.49 MB	0.226 s	0.905

Table 5.1: The memory, time and mAP comparisons of Baseline1 and Baseline2 on three datasets.

5.1.3.2 Baseline Evaluations

Before evaluating the effectiveness of the proposed compressive quantization, we first evaluate three baselines. We denote by **Baseline1** the exhaustive matching scheme defined as Eq. (5.1). It is considerably memory demanding and computationally costly since it requires to compare the query with all the object proposals of the videos.

We denote by **Baseline2** the algorithm based on k-means defined as Eq. (5.3). We vary the number of codewords k per video among 10, 20, 40 and 100. As shown in Table 5.1, the precision obtained by Baseline2 improves as the number

Figure 5.4: mAP comparison between Baseline2 and Baseline3.

of codewords k per video increases. Meanwhile, Baseline2 achieves comparable mAP with Baseline1 with significantly less computation and memory cost. For instance, on the CNN2h dataset, the mAP achieved by Baseline1 is 0.856 using 10.48 seconds and 2.060GB memory, whereas Baseline2 achieves 0.817 mAP when $k = 40$ using only 0.147 seconds and 28.12 MB memory. The promising results achieved by Baseline2 is expected since there exists heavy redundancy among the object proposals from the same video.

We denote by **Baseline3** the algorithm based on k-means followed by iterative quantization defined as Eq. (5.5). Note that Baseline3 conducts the quantization and hashing sequentially. We vary the number of codewords k per video among 10, 20, 40 and 100 and vary the code length l of the binary codewords among 128, 256 and 512. As we can see from Figure 5.4, the mAP achieved by Baseline3 is considerably lower than that of Baseline2. This is expected since the ITQ brings additional distortion errors, which can significantly deteriorate the search precision if the accumulated distortion errors from k-means and ITQ are large. Particularly, on the CNN2h dataset, when $k = 100$, Baseline2 achieves 0.845 mAP but Baseline3 ($l = 512$) only achieves 0.687 mAP.

Figure 5.5: mAP comparison of the proposed compressive quantization (ours) with Baseline3 on the Groundhog Day dataset.

Figure 5.6: mAP comparison of the proposed compressive quantization (ours) with Baseline3 on the CNN2h dataset.

Figure 5.7: mAP comparison of the proposed compressive quantization (ours) with Baseline3 on Egocentric dataset.

5.1.3.3 Evaluations of Compressive Quantization

In this section, we evaluate the performance of the proposed compressive quantization. To demonstrate its effectiveness, we directly compare it with Baseline2 and Baseline3, respectively.

Figure 5.5, 5.6, 5.7 compare the performance of the proposed compressive quantization with Baseline3 on Groundhog Day, CNN2h and Egocentric datasets. In our implementation, we vary the code length (l) among 128, 256 and 512 and vary the number of codewords (k) per video among 10, 20, 40 and 100. As we can see from Figure 5.5, 5.6, 5.7 that, our compressive quantization significantly outperforms Baseline3 when the memory and computation cost are the same. Particularly, on the Groundhog Day dataset, when $k = 40$ and $l = 256$, our compressive quantization achieves 0.654 mAP whereas Baseline3 only obtains 0.523 mAP. On the CNN2h dataset, when $k = 40$ and $l = 256$, our compressive quantization achieves 0.751 mAP whereas Baseline3 only obtains 0.578 mAP.

Figure 5.8 compares the mAP of the proposed compressive quantization with Baseline2 on Groundhog Day, CNN2h and Egocentric datasets. In our implementation, we fix the code length (l) as 512 and vary the number of codeword (k) per video among 10, 20, 40 and 100. We can observe from Figure 5.8 that, the precision achieved by the proposed compressive quantization is comparable with that from Baseline2 implemented by k-means. For example, on the CNN2h dataset, when $k = 100$, our compressive quantization has a mAP of 0.847, which is as good as Baseline2 which achieves 0.846 mAP. Meanwhile, it is also comparable with Baseline1 (0.856 mAP). The slightly worse precision of ours is expected because of the inevitable distortion errors caused by the quantization and the binarization steps.

Figure 5.9 compares the memory cost of the proposed compressive quantization with Baseline2 on three datasets. As shown in Figure 5.9, the proposed compres-

Figure 5.8: mAP comparison between the proposed compressive quantization (ours) and Baseline2.

Figure 5.9: Memory comparison between the proposed compressive quantization (ours) and Baseline2.

Figure 5.10: Time comparison between the proposed compressive quantization (ours) and Baseline2.

sive quantization requires much less memory than Baseline2. In fact, Baseline2 requires to store 256-dimension real-value codewords whereas the codewords from the proposed compressive quantization are only 512-bit. The memory cost of our compressive quantization is only 1/16 of that of Baseline2. Particularly, on the CNN2h dataset, when $k = 100$, the memory cost from Baseline2 is 70.32 MB, whereas our compressive quantization only takes 4.39 MB. Compared with Baseline1 using 2.060 GB, the proposed compressive quantization brings 480 $\times$ memory reduction.

Figure 5.10 compares the time cost of the proposed compressive quantization with Baseline2 on three datasets. As shown in Figure 5.10, the search based on our compressive quantization is significantly faster than that of Baseline2. This is attributed to the efficiency of Hamming distance computation. Particularly, on the CNN2h dataset, when $k = 100$, Baseline2 requires 314 milliseconds to conduct the search, whereas our compressive quantization only takes 8 milliseconds, which is around 39 $\times$ faster. Meanwhile, compared with Baseline1 which takes 10.48s, the acceleration of the proposed compressive quantization is around 1310 $\times$. Note that the reported time cost does not consider the encoding time of the query, since it can be ignored compared to the search time when dealing with huge datasets.

(a) *black clock* (b) *microphone*Figure 5.11: Top-3 search results of *black clock* and *microphone*.

	Groundhog Day			CNN2h		
	128	256	512	128	256	512
LSH [21]	0.393	0.526	0.533	0.313	0.376	0.473
SH [23]	0.512	0.632	0.615	0.242	0.731	0.771
BP [121]	0.576	0.586	0.617	0.317	0.691	0.645
CBE [122]	0.491	0.590	0.665	0.289	0.607	0.791
SPBE [102]	0.578	0.536	0.603	0.295	0.599	0.696
Ours	0.635	0.656	0.674	0.347	0.751	0.847

Table 5.2: Comparison of our method with k-means followed by other hashing methods.

In Figure 5.11, we visualize top-3 retrieved videos from the proposed compressive quantization method using queries *black clock* and *microphone* on the Groundhog Day dataset. For each video, we shows 6 keyframes uniformly sampled from it. We use the green bounding boxes to show the location of the query object. As shown in Figure 5.11, although the object only occupies a very small area in relevant frames, we are still able to retrieve the correct videos.

5.1.3.4 Comparison with Other Hashing Methods

We also compare our method with k-means followed by other hashing methods, *e.g.*, Locality-Sensitivity Hashing (LSH) [21], Spectral Hashing (SH) [23], Bilinear Projections (BP) [121], Circulant Binary Embedding (CBE) [122], and Sparse Projection Binary Encoding (SPBE) [102]. All of the above methods are conducted on the codewords obtained from k-means. We fix the number of codewords k per video as 100 and vary the code length l among 128, 256 and 512. As shown in Table 5.2, the mAP achieved by ours outperforms all above-mentioned hashing methods. When $l = 512$, ours achieves 0.847 mAP, whereas the second best mAP is only 0.791 achieved by CBE. The excellent performance of our compressive quantization is attributed to the joint optimization mechanism, since all other hashing methods optimize the hashing and quantization independently.

5.1.3.5 Summary

To remedy the huge memory and computation cost caused by leveraging object proposals in video-level object instance search task, we propose the compressive quantization. It compresses thousands of object proposals from each video into tens of binary codes but maintains good search performance. Thanks to our compressive quantization, the number of distance computations are significantly reduced and every distance calculation can be extremely fast. Experimental results on three benchmark datasets demonstrate that our approach achieves comparable performance with exhaustive search using only a fraction of memory and significantly less time cost.

5.2 Reconstruction Model

Although the number of object proposals per video is huge, the object proposals in a video tend to have high redundancy because of their large spatio-temporal overlaps [123]. Thus one plausible solution to speed up the search and reduce the memory cost of storing all the object proposals is to first select the representative ones for each video [70]. Then we only need to match the query with the selected representatives instead of all. According to [70], 10 percent selected representatives can achieve excellent performance in shot-level object instance search. However, to guarantee a reasonable recall, the selection ratio cannot be too small. Therefore, the reduction in memory and computational cost by representative selection is limited.

Different from representative selection, we propose the Reconstruction-based Object SEarch (ROSE) method to further improve the efficiency and accuracy of video-level object instance search, without exhaustively comparing the query with each object proposal. Specifically, for each video, instead of storing all of its object

proposals that may capture the query object, we train a compact model for each video to reconstruct all of its object proposals. Then in the search phase, the reconstruction model of each video can answer whether it has ever seen the query object, by trying to reconstruct the query. As the query is not directly compared with individual object proposals, our proposed solution brings a significant reduction in both computational and memory cost, while achieving comparable search accuracy to exhaustive search, due to the following properties:

- Instead of locating all instances of the query accurately in the videos, video-level object search only cares about whether the query object is contained in a video or not. As will be shown in our experiments, such an identification problem can be addressed by a simple reconstruction model.
- If a video contains an instance of the query object, more often than not, it will generate many spatio-temporally overlapping object proposals that capture the instance. Therefore, instead of “finding a needle in a haystack”, we are actually checking whether there are “a bunch of needles” in a haystack. Although the reconstruction model may not be able to capture a single needle, it is likely to capture “a bunch of needles”. Therefore, when the query object comes, the reconstruction model will not miss it.
- Training a model to reconstruct all object proposals can be time consuming, but it is performed offline for only once thus does not affect runtime efficiency. In the search phase, we only need to reconstruct the query to tell whether it is contained in the video, and the same reconstruction model works for different queries.

As illustrated in Figure 5.12, our Reconstruction-based Object SEarch (ROSE) method converts the set of object proposals into the parameters of the reconstruction model. The learned reconstruction model is a higher-level abstraction of the

(a) Training Phase

(b) Search Phase

Figure 5.12: Visualization of the reconstruction model.

set of features of object proposals, which is not only compact but also shows better generalization ability.

5.2.1 Reconstruction-based Object Search

We denote by $r_{\theta_i}(\cdot) : \mathbb{R}^d \rightarrow \mathbb{R}^d$ the reconstruction model learned by the set of object proposals $\mathcal{S}_i = \{\mathbf{x}_i^1, \dots, \mathbf{x}_i^m\}$ from the video V_i , where θ_i is the parameters of the reconstruction model which are learned by reconstructing all object proposals in the training phase:

$$\theta_i = \underset{\theta}{\operatorname{argmin}} \sum_{\mathbf{x} \in \mathcal{S}_i} \|\mathbf{x} - r_{\theta}(\mathbf{x})\|_2^2. \quad (5.20)$$

In the search phase, for each video V_i , we calculate the query's reconstruction error $\|r_{\theta_i}(\mathbf{q}) - \mathbf{q}\|_2$ using the reconstruction model learned from the video. We use the reconstruction error as a similarity measurement to determine the relevance between the query $\mathbf{q}$ and the whole video V_i :

$$\operatorname{dist}(\mathbf{q}, V_i) = \|\mathbf{q} - r_{\theta_i}(\mathbf{q})\|_2 \quad (5.21)$$

The smaller the reconstruction error $\|\mathbf{q} - r_{\theta_i}(\mathbf{q})\|_2$ is, the more relevant the query is to the video V_i .

If the query object is similar to some object proposals from the video (training data), its reconstruction error should be small. After training the reconstruction model for the video V_i , we no longer rely on the object proposals $\mathcal{S}_i = \{\mathbf{x}_i^1, \dots, \mathbf{x}_i^m\}$ and only need to store the parameters θ_i , which is more compact than $\mathcal{S}_i$. Meanwhile, rather than comparing the query $\mathbf{q}$ with all the object proposals in the set $\mathcal{S}_i$, the reconstruction model only need to compute $\|\mathbf{q} - r_{\theta_i}(\mathbf{q})\|_2$ to obtain the relevance between $\mathbf{q}$ and $\mathcal{S}_i$, which can be calculated very efficiently.

The Reconstruction-based Object SEarch (ROSE) method characterizes the set

of object proposals $\mathcal{S}_i$ extracted from each video V_i by the reconstruction model r_{θ_i} trained from V_i . The reconstruction error of the query $\|\mathbf{q} - r_{\theta_i}(\mathbf{q})\|_2$ serves as a relevance measurement to rank the videos. It should be avoided to build the reconstruction model which can perfectly reconstruct any input vector since it will make the reconstruction error meaningless. This can be achieved by adding constraints or regularization terms in training the reconstruction model. We attempt multiple methods to implement the reconstruction model $r_{\theta_i}(\cdot)$. It includes subspace projection, auto-encoder and sparse dictionary learning.

5.2.1.1 Subspace Projection

We define the reconstruction model based on subspace projection as:

$$r_{\theta_i}(\mathbf{x}) = \boldsymbol{\theta}_i^\top \boldsymbol{\theta}_i \mathbf{x}, \text{ s.t. } \boldsymbol{\theta}_i \boldsymbol{\theta}_i^\top = \mathbf{I}, \quad (5.22)$$

where $\boldsymbol{\theta}_i^\top \boldsymbol{\theta}_i$ is the projection matrix and $\boldsymbol{\theta}_i \in \mathbb{R}^{l \times d}$ consists of the bases of the projected subspace. To avoid $r_{\theta_i}(\mathbf{x}) = 0$ for any $\mathbf{x} \in \mathbb{R}^d$, we must set $l < d$. Otherwise, if $l = d$, $r_{\theta_i}(\mathbf{x}) = \mathbf{x}$.

In the training phase, we seek to learn $\boldsymbol{\theta}_i$ by optimizing the following problem:

$$\begin{aligned} \boldsymbol{\theta}_i &= \underset{\boldsymbol{\theta} \in \{\mathbf{A} \in \mathbb{R}^{l \times d} | \mathbf{A}\mathbf{A}^\top = \mathbf{I}\}}{\operatorname{argmin}} \sum_{\mathbf{x} \in \mathcal{S}_i} \|\mathbf{x} - \boldsymbol{\theta}^\top \boldsymbol{\theta} \mathbf{x}\|_2^2 \\ &= \underset{\boldsymbol{\theta} \in \{\mathbf{A} \in \mathbb{R}^{l \times d} | \mathbf{A}\mathbf{A}^\top = \mathbf{I}\}}{\operatorname{argmin}} \|\mathbf{X}_i - \boldsymbol{\theta}^\top \boldsymbol{\theta} \mathbf{X}_i\|_F^2, \end{aligned} \quad (5.23)$$

where $\mathbf{X}_i = [\mathbf{x}_1, \dots, \mathbf{x}_m] \in \mathbb{R}^{d \times m}$ is the matrix consisting of features of all the object proposals from the video V_i . The optimal $\boldsymbol{\theta}_i$ is obtained by the singular value decomposition. The singular value decomposition of $\mathbf{X}_i$ is defined as $\mathbf{X}_i = \mathbf{U}_i \boldsymbol{\Sigma}_i \mathbf{V}_i^\top$. It can be proven that the optimal $\boldsymbol{\theta}_i \in \mathbb{R}^{l \times d}$ can be simply obtained by

choosing the first l columns of the matrix $\mathbf{U}_i$, *i.e.*, the left singular vectors of the l largest eigen-values.

In the testing phase, the reconstruction error of $\mathbf{q}$ using the reconstruction model learned from each set $\mathcal{S}_i$ can be obtained by $\|\mathbf{q} - \boldsymbol{\theta}_i^\top \boldsymbol{\theta}_i \mathbf{q}\|_2$. Intuitively, it measures the shortest euclidean distance between the query object $\mathbf{q}$ and the data points which lies in the subspace spanned by the bases $\boldsymbol{\theta}_i$.

Compared with exhaustively comparing the query $\mathbf{q}$ with all the object proposals $\mathcal{S}_i = \{\mathbf{x}_i^1, \dots, \mathbf{x}_i^m\}$, the reconstruction model implemented by subspace projection is much more efficient, since $l \ll m$. Nevertheless, how to choose l is not a trivial problem. On one hand, if l is too small, it will fail to capture the information embedded in the batch of data. On the other hand, if l is too large, considering an extreme case when $l = d$, *i.e.*, $r_{\boldsymbol{\theta}}(\mathbf{x}) = \mathbf{x}$, it can perfectly reconstruct any d -dimensional vector since $\boldsymbol{\theta}_i^\top \boldsymbol{\theta}_i = \mathbf{I}$. The above situation limits the usefulness of the reconstruction model implemented by subspace projection.

5.2.1.2 Auto-encoder

We define the reconstruction model implemented by auto-encoder as:

$$r_{\boldsymbol{\theta}_i}(\mathbf{x}) = f_2(\mathbf{W}_i^2 f_1(\mathbf{W}_i^1 \mathbf{x} + \mathbf{b}_i^1) + \mathbf{b}_i^2), \quad (5.24)$$

where $\mathbf{b}_i^1 \in \mathbb{R}^l$ and $\mathbf{b}_i^2 \in \mathbb{R}^d$ are the bias vectors, $\mathbf{W}_i^1 \in \mathbb{R}^{l \times d}$ and $\mathbf{W}_i^2 \in \mathbb{R}^{l \times d}$ are the weight matrices, l is the number of hidden units, $f_1(\cdot)$ is the encoding activation function and $f_2(\cdot)$ is the decoding activation function.

In the training phase, $\boldsymbol{\theta}_i = \{\mathbf{W}_i^1, \mathbf{W}_i^2, \mathbf{b}_i^1, \mathbf{b}_i^2\}$ can be obtained by optimizing the following problem:

$$\boldsymbol{\theta}_i = \underset{\boldsymbol{\theta}}{\operatorname{argmin}} \sum_{\mathbf{x} \in \mathcal{S}_i} \|\mathbf{x} - f_2(\mathbf{W}^2 f_1(\mathbf{W}^1 \mathbf{x} + \mathbf{b}^1) + \mathbf{b}^2)\|_2^2. \quad (5.25)$$

In the test phase, the reconstruction error of $\mathbf{q}$ for each set $\mathcal{S}_i$ can be calculated by $\|\mathbf{q} - f_2(\mathbf{W}_i^2 f_1(\mathbf{W}_i^1 \mathbf{q} + \mathbf{b}_i^1) + \mathbf{b}_i^2)\|_2$.

Based on the reconstruction model implemented by the auto-encoder, for each video V_i , we only need to store $\theta_i = \{\mathbf{W}_i^1, \mathbf{W}_i^2, \mathbf{b}_i^1, \mathbf{b}_i^2\}$, which require $\mathcal{O}(dl)$ memory complexity. Meanwhile, the computational cost for calculating the reconstruction error for each video is $\mathcal{O}(dl)$, which is mainly cost on the matrix-vector product $\mathbf{W}_i^2 f_1(\cdot)$ and $\mathbf{W}_i^1 \mathbf{q}$.

In fact, auto-encoder is similar to the subspace projection implemented by singular value decomposition except for the nonlinear activation function $f_1(\cdot)$ and $f_2(\cdot)$ and the bias vector $\mathbf{b}_i^1$ and $\mathbf{b}_i^2$. It can be proven that if the input vector $\mathbf{x}$ is zero-mean and both the encoder activation function and decoder activation function are linear functions, optimizing auto-encoder will be equivalent to singular value decomposition. Note that, when the number of hidden units l is large, it tends to be capable of reconstructing any input vector like the situation we encounter in the reconstruction model implemented by the subspace projection. To tackle this problem, a natural solution is adding sparsity constraint. We implement two types of sparsity constraint auto-encoder. They are sparse auto-encoder [124] and k-sparse auto-encoder [125].

Sparse Auto-encoder In order to constrain the sparsity of the activation of the hidden units, sparse auto-encoder adds $KL(\rho \parallel \hat{\rho}_j)$, the Kullback-Leibler(KL) divergence between average activation of hidden units and sparsity parameter ρ , as a penalty term in the object function in Eq. (5.25), where $\hat{\rho}_j$ is the average activation of j -th hidden unit.

K-sparse Auto-encoder [125] proposed the k-sparse autoencoder with linear activation function. In the feedforward phase, after computing the hidden code $\mathbf{z} = \mathbf{W}_1 \mathbf{x} + \mathbf{b}$, rather than constructing the output from all of the hidden units, it identifies the k largest hidden units in $\mathbf{z}$ and set the others to zero.

5.2.1.3 Sparse Dictionary Learning

We define the reconstruction model implemented by sparse dictionary learning as :

$$r_{\theta_i}(\mathbf{x}) = \theta_i \mathbf{h}_{\theta_i}(\mathbf{x}), \quad (5.26)$$

where $\theta_i \in \mathbb{R}^{d \times l}$ consists of the atoms of the dictionary learned from $\mathcal{S}_i$. $\mathbf{h}_{\theta_i}(\mathbf{x})$ is the sparse code of the vector $\mathbf{x}$ based on θ_i satisfying $\|\mathbf{h}_{\theta_i}(\mathbf{x})\|_0 = z$, where z is the number of non-zero elements of the code.

In the training phase, θ_i is learned by optimizing the following problem:

$$\begin{aligned} \theta_i = \underset{\theta}{\operatorname{argmin}} \sum_{\mathbf{x} \in \mathcal{S}_i} \|\mathbf{x} - \theta_i \mathbf{h}_{\theta_i}(\mathbf{x})\|_2^2 \\ \text{s.t. } \|\mathbf{h}_{\theta_i}(\mathbf{x})\|_0 = z, \forall \mathbf{x} \in \mathcal{S}_i. \end{aligned} \quad (5.27)$$

The above l_0 -norm constrained optimization is normally converted into l_1 -penalty problem given by

$$\begin{aligned} \theta_i = \underset{\theta}{\operatorname{argmin}} \sum_{\mathbf{x} \in \mathcal{S}_i} \|\mathbf{x} - \theta_i \mathbf{h}_{\theta_i}(\mathbf{x})\|_2^2 + \lambda \|\mathbf{h}_{\theta_i}(\mathbf{x})\|_1 \\ = \underset{\theta}{\operatorname{argmin}} \|\mathbf{X}_i - \theta_i \mathbf{H}_i\|_F^2 + \lambda \|\mathbf{H}_i\|_{1,1}, \end{aligned} \quad (5.28)$$

where l_1 -penalty on $\mathbf{h}_{\theta_i}(\mathbf{x})$ (sum of the magnitude of its elements) encourages a sparse solution of $\mathbf{h}_{\theta_i}(\mathbf{x})$, $\mathbf{X}_i = [\mathbf{x}_1, \dots, \mathbf{x}_m] \in \mathbb{R}^{d \times m}$ is the matrix of all the data points from the set $\mathcal{S}_i$, $\mathbf{H}_i = [\mathbf{h}_{\theta_i}(\mathbf{x}_i^1), \dots, \mathbf{h}_{\theta_i}(\mathbf{x}_i^m)]$ is the matrix consisting of the sparse codes, $\|\mathbf{H}_i\|_{1,1} = \sum_{j=1}^m \|\mathbf{h}_{\theta_i}(\mathbf{x}_i^j)\|_1$ and λ is the parameter to control the sparsity of $\mathbf{h}_{\theta_i}(\cdot)$. We use online dictionary learning [126] to solve the above optimization problem since it is efficient to handle large-scale high-dimensional data.

In the search phase, given a query point $\mathbf{q}$, for each set $\mathcal{S}_i$, we can efficiently obtain its sparse code $\mathbf{h}_{\theta_i}(\mathbf{q})$ using orthogonal matching pursuit (OMP):

$$\mathbf{h}_{\theta_i}(\mathbf{q}) = \text{OMP}(\mathbf{q}, \theta_i, z), \quad (5.29)$$

The reconstruction error of $\mathbf{q}$ from the reconstruction model $r_{\theta_i}(\mathbf{x})$ is further obtained by $\|\mathbf{q} - \theta \mathbf{h}_{\theta_i}(\mathbf{q})\|_2$.

The reconstruction model implemented by sparse coding only requires to store the dictionary atoms $\theta_i \in \mathbb{R}^{d \times l}$ which takes $O(ld)$ memory cost. The most time-consuming part in the search phase is to obtain the sparse codes $\mathbf{h}_{\theta_i}(\mathbf{q})$ through orthogonal matching pursuit. It takes $O(zld)$ computational complexity. In our experiments, we find the number of non-zero element of the code $z = 2$ achieves excellent performance in search precision as well as efficiency.

5.2.2 Experiments

5.2.2.1 Settings

In this paper, we adopt Edge Boxes [34] to generate 300 object proposals for each frame of the videos. For each object proposal, we further extract its feature by max-pooling the last convolutional layer of VGG-16 CNN model [103] pre-trained on Imagenet dataset. The max-pooled 512-dimensional features are further post-processed by principal component analysis (PCA) and whitening in order to suppress the burstiness but the dimension of the feature is kept as 512.

Observing that the object proposals from the same frame tend to overlap with each other and there exist strong redundancy among them. To boost the efficiency of the training process, we use k-means clustering to group 300 object proposals from every frame into 30 clusters and select the centroids of the clusters as compact object proposals. In this scheme, each frame will be represented by 30 compact

object proposals and the whole video will be represented by a pool of $30m$ compact object proposals, where m is the number of frames.

5.2.2.2 Evaluation Metric and Dataset

The effectiveness of the proposed method is evaluated by mean average precision (mAP). We conduct the systematic experiments on CNN-2h [44], Egocentric1 [45] and Egocentric2 [45] datasets. The CNN-2h dataset contains a long video of 72,000 frames and we equally divide them into 100 videos. Each video consists of 720 frames and 21600 object proposals. The Egocentric1 dataset consists of 19 long videos. We uniformly sample the keyframes per 20 frames, obtain 51804 keyframes and further equally divide all the keyframes into 101 videos. Each video consists of around 500 keyframes and around 15000 object proposals. The Egocentric2 dataset consists of 21 long videos. We uniformly sample the keyframes per 20 frames, obtain 57802 keyframes and equally divide all the keyframes into 114 videos. Each video consists of around 500 keyframes and around 15000 object proposals.

For the evaluation of object instance retrieval, we used eight query objects for each dataset and skipped those scene query images. Figure 5.13 visualizes the query objects. In fact, the number of frames where query object appears is much fewer than the total number of frames in the video. We define the frame ratio $\eta(q, V_i)$ as the number of frames in V_i which contain the query object q divided by the number of total frames in the video V_i . Furthermore, we define $\hat{\eta}(q)$ as the average frame ratio of q which is computed by:

$$\hat{\eta}(q) = \sum_{V \in \mathcal{V}_q} \eta(q, V) / |\mathcal{V}_q|, \quad (5.30)$$

where $\mathcal{V}_q$ denotes the set of videos which are relevant to q . Table 5.3 shows the average ratio of all the queries on three datasets. It can be seen that the average ratios of queries are quite small. For example, on Egocentric1 dataset, the average

(a) CNN2h

(b) Egocentric1

(c) Egocentric2

Figure 5.13: Query Objects Visualization.

	1	2	3	4	5	6	7	8	Avg
CNN2h	5.0%	10.2%	3.9%	2.0%	3.5%	14.5%	5.6%	9.1%	6.7%
Egocentric1	2.0%	1.4%	1.6%	2.2%	1.2%	3.6%	0.8%	1.1%	1.7%
Egocentric2	1.6%	2.0%	1.8%	0.9%	1.8%	3.6%	4.1%	5.4%	2.7%

Table 5.3: Average frame ratios of 8 queries on three datasets.

frame ratio of 5-th query is only 1.2%. Since each video of Egocentric1 dataset contains around 500 keyframes, it means that, on average, the query object only appears in only 6 keyframes in the relevant videos.

5.2.2.3 Effectiveness of ROSE Method

We evaluate the performance of the proposed Reconstruct-based Object SEarch (ROSE) method implemented by subspace projection, sparse autoencoder, k-sparse autoencoder and sparse dictionary learning, in the video-level object instance retrieval task in this section.

Before we evaluate the search precision of ROSE, we design an experiment to demonstrate the effectiveness of reconstruction model in capturing the relevance

Figure 5.14: The reconstruction error decreases as the ratio of the query increases.

of the query with the video consisting of a set of object proposals. Particularly, we choose the first query from the CNN2h dataset as our test query. We choose the first video from the CNN2h dataset consisting of 21000 (700 frames * 30 proposals / frame) object proposals, as the training data to learn the reconstruction model. We manually mix the original training data with multiple duplicates of the feature of the query object to form the new training data. We adjust the ratio of the query in the new training data from 0 to 0.02. For example, we add 420 duplicates of the query object to original 21000 object proposals to achieve 0.02 ratio. We can see from Figure 5.14 that, the reconstruction error of the query object decreases as the ratio of the query object increases, which verifies the effectiveness of the reconstruction error in characterizing the relevance of the query object with the video.

Figure 5.15(a) shows the mAP from the reconstruction model implemented by subspace projection. We vary the number of singular vectors l from 10 to 500. It is interesting that the mAP increases and then decreases as number of selected singular vectors increase. Particularly, when $l = 10$ and $l = 500$, the mAP is relatively low. This is because of the fact that too few singular vectors can not capture enough information contained in the set of the object proposals.

Figure 5.15: The performance comparison of the reconstruction model implemented by subspace projection and sparse autoencoder.

Meanwhile too many singular vectors will make the reconstruction model easily reconstruct any input vectors, which makes it difficult to discriminate different videos.

Figure 5.15(b) shows the mAP of the proposed reconstruction model implemented by sparse autoencoder. The encoder activation function $f_1(\cdot)$ is implemented by rectifier defined as $f_1(x) = \max(0, x)$.

The decoder activation function is implemented by the linear function $f_2(x) = x$. We set the expected average activations $\hat{\rho}$ of all the hidden neural nodes as 0.05. It can be observed that from Figure 5.15(b) that, the reconstruction model implemented by sparse autoencoder can achieve more stable performance compared with the reconstruction model implemented by subspace projection. Particularly, it achieves 0.746 mAP on CNN-2h dataset when the number of hidden units is 500, whereas the subspace projection can only achieves 0.443 mAP when the number of singular vector is 500. The more stable performance can be attributed to the sparse penalty term.

Figure 5.16 shows the performance of the reconstruction model implemented

Figure 5.16: The performance of the reconstruction model implemented by k -sparse autoencoder.

Figure 5.17: The performance of the reconstruction model implemented by sparse coding.

by k -sparse autoencoder. We vary the number of nonzero activations of the hidden layer k among 10, 20 and 30. Compared with sparse autoencoder, the performance of k -sparse autoencoder is much better and more stable. Especially on Egocentric2 dataset, it can achieve $\text{mAP} = 1.00$ when $k = 10$ and the number of hidden units is 200 whereas the sparse autoencoder only achieved around $\text{mAP} = 0.75$ when the number of hidden units is 200.

Figure 5.17 shows the mAP from the reconstruction model implemented by sparse dictionary learning as the number of atoms and the number of non-zero element in the codes z changes. It can be observed that the value of z does not influence the performance significantly when it varies between 2 and 4. Therefore,

Method	CNN-2h		Egocentric1		Egocentric2	
	Memory	mAP	Memory	mAP	Memory	mAP
ROSE-SDL (ours)	400KB	1.00	400KB	0.94	400KB	1.00
SMRS	400KB	0.73	400KB	0.83	400KB	0.62
K-mediods	400KB	0.63	400KB	0.69	400KB	0.89
Compressive Quantization	400KB	0.95	400KB	0.91	400KB	0.97
Exhaustive Search	42MB	1.00	30MB	1.00	30MB	1.00

Table 5.4: Comparison of our ROSE method with the representative selection method and exhaustive search method.

we set the default number of z as 2 on all three datasets. Compared with the reconstruction model implemented by sparse autoencoder or k-sparse autoencoder, the reconstruction model implemented by sparse dictionary learning is more accurate and stable. It can achieve $\text{mAP} = 1.00$ on both CNN-2h and Egocentric2 datasets when the number of atoms per video is only 150. We choose the sparse dictionary learning as the default implementation of the proposed reconstruction model.

5.2.2.4 Comparison with Baselines

In order to further demonstrate the effectiveness of the proposed ROSE method, we compare it with exhaustive search scheme and representative selection scheme [2]. Particularly, the exhaustive search scheme directly compares the query with all the object proposals from each video to determine the relevance of the video with the query. The representative selection scheme selects a set of representative objects $\mathcal{O}_i$ from the set of object proposals $\mathcal{S}_i$ for each video V_i . The relevance of the video V_i with the query $\mathbf{q}$ is determined by the similarity score of the best-matched representatives as

$$R(\mathbf{q}, V_i) = \max_{\mathbf{o} \in \mathcal{O}_i} \text{sim}(\mathbf{q}, \mathbf{o}) \quad (5.31)$$

We compare with two types of representative selection methods. They are k-mediods [1] and SMRS [2]. The reconstruction model we use for comparison is

implemented by sparse dictionary learning and the number of non-zero elements in the codes z is set as 2. We conduct the comparison on the condition when the number of atoms/representatives l is set to be 200, which costs 400KB memory for each video.

From Table 5.4, we can see that our ROSE implemented by sparse dictionary learning (ROSE-SDL) achieves comparable search precision as exhaustive search but takes much less memory. For example, on the CNN2h dataset, the exhaustive search takes 42MB to achieve $\text{mAP} = 1.00$. In contrast, to achieve $\text{mAP} = 1.00$, our ROSE-SDL only takes 400KB memory, which is only around 1% as that for the exhaustive search method. Meanwhile, using comparable memory cost, our ROSE-SDL achieves significantly higher precision than the representative selection implemented by k-medoids and SMRS on all three datasets. For example, on CNN-2h dataset, the proposed reconstruction model can achieve $\text{mAP} = 1.00$, whereas the mAP of the representative selection scheme implemented by k-medoids and SMRS are both less than 0.75. Moreover, we also compare the proposed ROSE-SDL with the proposed compressive quantization in section 5.1. To make a fair comparison, we generate 800 512-bit binary codes for each video using compressive quantization, which takes 400KB. As shown in Table 5.4, ROSE-SDL consistently outperforms compressive quantization on three datasets.

We visualize the search results from our ROSE-DSL method on CNN2h, Egocentric1 and Egocentric2 datasets in Figure 5.18. For each dataset, we show one query and its top-3 retrieved videos. We can see that, in many cases the query object only appears in few frames of the video and occupies a very small area surrounded by dense clutter in the frame but our method can still successfully retrieve the videos where the query object appears.

(a) CNN2h

(b) Egocentric1

(c) Egocentric2

Figure 5.18: Visualization of top-3 search results of the proposed ROSE method on three datasets.

5.2.3 Summary

Our work provides a novel perspective to video-level object instance search and proposes the Reconstruction-based Object SEarch (ROSE) method to achieve search efficiency. In this method, a huge number of object proposals generated from the

frames of a video are compactly characterized by the parameters of a learned reconstruction model. Then the video-level object instance search is converted into computing the reconstruction error of the query object using the reconstruction model learned for each video. It is significantly faster than exhaustively comparing the query object with all the object proposals from the video. Meanwhile, storing the reconstruction model of the video greatly reduces the memory cost.

Chapter 6

Conclusions and Future Research

6.1 Conclusions

This thesis studies fast visual search in images and videos, which has various applications in social network applications. Particularly, we tackle three types of visual search tasks including image-to-image search, image-to-video search and video-to-video search. We not only propose several novel neural networks to obtain compact and discriminative representations of images and videos, but also propose novel matching strategy to speed up the search process. The main contribution of this thesis are a detailed review of related works in this field and a set of solutions to speed up the object instance search. This thesis revisits and extends several existing algorithms in computer vision including product quantization, residual quantization, sparse coding and feature reconstruction. Our work in this thesis is summarized as follows.

First, we make the product quantization differentiable by constructing an approximation function. We implement the approximation function as a layer of the proposed product quantization network (PQN), which supports an end-to-end training. By revisiting the residual quantization, we extend the proposed PQN

to residual product quantization network (RPQN). RPQN triggers the residual learning, boosting the retrieval accuracy. By utilizing the temporal consistency inherited in videos, we extend PQN to temporal product quantization network (TPQN) to speed up the video-to-video search. TPQN incorporates frame feature learning, feature aggregation and video feature product quantization in a single neural network, supporting an end-to-end training.

Second, observing the redundancy among the object proposals, we propose to utilize sparse coding to factorize their features into a product between a compact codebook matrix and a sparse code matrix. To speed up the object instance search in images, we propose fuzzy object matching (FOM), which exploits the spatial redundancy among the object proposals from the same image. To speed up the object instance search in video frames, we further propose hierarchical object prototype encoding (HOPE) which utilizes not only the spatial redundancy among the object proposals from the same frame but also the temporal redundancy among the object proposals from the neighboring frames. We develop a hierarchical sparse coding scheme and significantly boost the search efficiency.

Third, we formulate the task of the object instance search in video clips into a point-to-set matching problem. To calculate the point-to-set similarity efficiently, we propose two strategies including compressive quantization (CQ) and reconstruction-based method (RM). CQ converts a large set of object proposals' features into a small set of binary codes. It not only speeds up similarity measurement between two vectors benefited from binary codes but also avoids exhaustive comparisons with all object proposals. In parallel to CQ, RM converts a set of object proposals into the parameters of the reconstruction model, which is not only compact but also shows better generalization ability.

6.2 Future Research

In this section, three topics are proposed to be explored in future research.

6.2.1 Fast Search Based on Deep Learning

Despite that many works such as deep hashing and deep quantization have successfully integrated some traditional fast search techniques in neural network for achieving an end-to-end training, many fast search algorithms such as tree partition and inverted indexing are still based on off-the-shelf features. How to integrate them in a deep neural network are rarely exploited. Deep exploration in this topic will be one of directions in my future research.

6.2.2 Self-supervised Learning for Fast Visual Search

Recently, we have witnessed great progress of self-supervision in computer vision and machine learning. The self-supervised learning does not rely on manual labels, which is a more practical solution for handling a large-scale dataset. It has been widely exploited in object recognition, object detection and video recognition. Nevertheless, it is rarely exploiting in fast visual search. My future research will explore this problem in depth.

6.2.3 Distilling for Compact Representation Learning

Distilling has achieved great success in model compression. It empowers a small-scale model the capability of a large-scale model by knowledge distilling. It is natural to utilize the distilling idea to learn compact representation such as binary codes. In this case, the distilling can empower the binary codes the discriminating power of their uncompressed counterparts.

Author's Publications

1. Tan Yu, Zhou Ren, Yuncheng Li, Enxu Yan, Ning Xu, and Junsong Yuan, “Temporal Structure Mining for Weakly Supervised Action Detection”, in Proceedings of International Conference on Computer Vision (ICCV), 2019
2. Tan Yu, Junsong Yuan, Chen Fang, and Hailin Jin, “Product Quantization Network for Fast Image Retrieval”, in Proceedings of European Conference on Computer Vision (ECCV), 2018
3. Tan Yu, Jingjing Meng, and Junsong Yuan, “Multi-view Harmonized Bilinear Network for 3D Object Recognition”, in Proceedings of IEEE International Conference on Computer Vision and Pattern Recognition (CVPR), 2018
4. Tan Yu, Zhenzhen Wang, and Junsong Yuan, “Compressive Quantization for Fast Object Instance Search in Videos”, in Proceedings of International Conference on Computer Vision (ICCV), 2017
5. Tan Yu, Yuwei Wu, and Junsong Yuan, “HOPE: Hierarchical Object Prototype Encoding for Efficient Object Instance Search in Videos”, in Proceedings of IEEE International Conference on Computer Vision and Pattern Recognition (CVPR), 2017
6. Tan Yu, Jingling Meng, and Junsong Yuan, “Is My Object in This Video? Reconstruction-based Object Search in Video”, in Proceedings of International Joint Conferences on Artificial Intelligence (IJCAI), 2017

-
7. Tan Yu, Yuwei Wu, Das Bhattacharjee Sreyasee, and Junsong Yuan, “Efficient Object Instance Search Using Fuzzy Object Matching”, in Proceedings of AAAI Conference on Artificial Intelligence (AAAI), 2017

Bibliography

- [1] H.-S. Park and C.-H. Jun, “A simple and fast algorithm for k-medoids clustering,” *Expert Systems with Applications*, vol. 36, no. 2, pp. 3336–3341, 2009.
- [2] E. Elhamifar, G. Sapiro, and R. Vidal, “See all by looking at a few: Sparse modeling for finding representative objects,” in *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*. IEEE, 2012, pp. 1600–1607.
- [3] A. Iscen, M. Rabbat, and T. Furon, “Efficient large-scale similarity search using matrix factorization,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016.
- [4] H. Jegou, M. Douze, and C. Schmid, “Product quantization for nearest neighbor search,” *IEEE Transactions on PAMI*, vol. 33, no. 1, pp. 117–128, 2011.
- [5] T. Ge *et al.*, “Optimized product quantization for approximate nearest neighbor search,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2013, pp. 2946–2953.
- [6] H. Jain *et al.*, “Approximate search with quantized sparse representations,” in *European Conference on Computer Vision*. Springer, 2016, pp. 681–696.

- [7] J. Philbin *et al.*, “Object retrieval with large vocabularies and fast spatial matching,” in *CVPR*, 2007, pp. 1–8.
- [8] F. Perronnin *et al.*, “Large-scale image retrieval with compressed fisher vectors,” in *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2010, pp. 3384–3391.
- [9] H. Jégou *et al.*, “Aggregating local descriptors into a compact image representation,” in *CVPR*, 2010, pp. 3304–3311.
- [10] A. Babenko *et al.*, “Neural codes for image retrieval,” in *ECCV*. Springer, 2014, pp. 584–599.
- [11] A. Babenko and V. Lempitsky, “Aggregating local deep features for image retrieval,” in *Proceedings of the IEEE International Conference on Computer Vision*, 2015, pp. 1269–1277.
- [12] J. Ng, F. Yang, and L. Davis, “Exploiting local features from deep networks for image retrieval,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 2015, pp. 53–61.
- [13] A. Gordo *et al.*, “Deep image retrieval: Learning global representations for image search,” in *European Conference on Computer Vision*. Springer, 2016, pp. 241–257.
- [14] T. Yu *et al.*, “Efficient object instance search using fuzzy objects matching.” *AAAI*, 2017.
- [15] S. Bai *et al.*, “Regularized diffusion process on bidirectional context for object retrieval,” *TPAMI*, 2018.
- [16] S. Bai *et al.*, “Ensemble diffusion for retrieval.”

- [17] T. Yu, J. Meng, and J. Yuan, “Is my object in this video? reconstruction-based object search in videos,” in *Proceedings of the 26th International Joint Conference on Artificial Intelligence*. AAAI Press, 2017, pp. 4551–4557.
- [18] T. Yu, Y. Wu, and J. Yuan, “Hope: Hierarchical object prototype encoding for efficient object instance search in videos,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 2424–2433.
- [19] P. Indyk and R. Motwani, “Approximate nearest neighbors: towards removing the curse of dimensionality,” in *Proceedings of the thirtieth annual ACM symposium on Theory of computing*. ACM, 1998, pp. 604–613.
- [20] M. S. Charikar, “Similarity estimation techniques from rounding algorithms,” in *Proceedings of the thirty-fourth annual ACM symposium on Theory of computing*, 2002, pp. 380–388.
- [21] M. Datar *et al.*, “Locality-sensitive hashing scheme based on p-stable distributions,” in *Proceedings of the twentieth annual symposium on Computational geometry*. ACM, 2004, pp. 253–262.
- [22] R. Salakhutdinov and G. Hinton, “Semantic hashing,” *RBM*, vol. 500, no. 3, p. 500, 2007.
- [23] Y. Weiss, A. Torralba, and R. Fergus, “Spectral hashing,” in *NIPS*, 2009, pp. 1753–1760.
- [24] Y. Gong *et al.*, “Iterative quantization: A procrustean approach to learning binary codes for large-scale image retrieval,” *IEEE T-PAMI*, vol. 35, no. 12, pp. 2916–2929, 2013.

- [25] R. Xia *et al.*, “Supervised hashing for image retrieval via image representation learning,” in *AAAI*. AAAI Press, 2014, pp. 2156–2162.
- [26] H. Lai *et al.*, “Simultaneous feature learning and hash coding with deep neural networks,” *arXiv preprint arXiv:1504.03410*, 2015.
- [27] H. Liu *et al.*, “Deep supervised hashing for fast image retrieval,” in *CVPR*, 2016, pp. 2064–2072.
- [28] Q. Li *et al.*, “Deep supervised discrete hashing,” in *NIPS*, 2017, pp. 2479–2488.
- [29] M. Norouzi and D. J. Fleet, “Cartesian k-means,” in *CVPR*, 2013, pp. 3017–3024.
- [30] A. Babenko and V. Lempitsky, “Additive quantization for extreme vector compression,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2014, pp. 931–938.
- [31] T. Zhang, C. Du, and J. Wang, “Composite quantization for approximate nearest neighbor search.” in *Proceedings of the International Conference on Machine Learning*, 2014, pp. 838–846.
- [32] W. Hong, J. Meng, and J. Yuan, “Distributed composite quantization.” in *AAAI*, 2018.
- [33] X. Wang *et al.*, “Supervised quantization for similarity search,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. CVPR, 2016.
- [34] C. L. Zitnick and P. Dollár, “Edge boxes: Locating object proposals from edges,” in *European Conference on Computer Vision*. Springer, 2014, pp. 391–405.

- [35] A. Krizhevsky, “Learning multiple layers of features from tiny images,” 2009.
- [36] T.-S. Chua *et al.*, “Nus-wide: a real-world web image database from national university of singapore,” in *Proceedings of the ACM international conference on image and video retrieval*, 2009, p. 48.
- [37] K. Soomro, A. R. Zamir, and M. Shah, “Ucf101: A dataset of 101 human actions classes from videos in the wild,” *arXiv preprint arXiv:1212.0402*, 2012.
- [38] H. Kuehne *et al.*, “Hmdb: a large video database for human motion recognition,” in *Computer Vision (ICCV), 2011 IEEE International Conference on*. IEEE, 2011, pp. 2556–2563.
- [39] J. Philbin *et al.*, “Object retrieval with large vocabularies and fast spatial matching,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2007.
- [40] J. Philbin *et al.*, “Lost in quantization: Improving particular object retrieval in large scale image databases,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2008.
- [41] R. Arandjelović and A. Zisserman, “Smooth object retrieval using a bag of boundaries,” in *Proc. of the IEEE International Conference on Computer Vision*, 2011, pp. 375–382.
- [42] J. Sivic and A. Zisserman, “Efficient visual search of videos cast as text retrieval,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 31, no. 4, pp. 591–606, 2009.

- [43] J. Meng *et al.*, “Object instance search in videos via spatio-temporal trajectory discovery,” *IEEE Transactions on Multimedia*, vol. 18, no. 1, pp. 116–127, 2016.
- [44] A. Araujo *et al.*, “Efficient video search using image queries,” in *ICIP*, 2014, pp. 3082–3086.
- [45] V. Chandrasekhar *et al.*, “Efficient retrieval from large-scale egocentric visual data using a sparse graph representation,” in *CVPR Workshops*, 2014, pp. 527–534.
- [46] W. Hong, J. Yuan, and S. D. Bhattacharjee, “Fried binary embedding for high-dimensional visual features,” *CVPR*, vol. 11, p. 18, 2017.
- [47] W. Hong and J. Yuan, “Fried binary embedding: From high-dimensional visual features to high-dimensional binary codes,” *IEEE Transactions on Image Processing*, vol. 27, no. 10, 2018.
- [48] W. Hong, J. Meng, and J. Yuan, “Tensorized projection for high-dimensional binary embedding.” in *AAAI*, 2018.
- [49] T. Yu, Z. Wang, and J. Yuan, “Compressive quantization for fast object instance search in videos,” in *ICCV*, 2017, pp. 833–842.
- [50] F. Shen *et al.*, “Supervised discrete hashing,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 37–45.
- [51] W. Liu *et al.*, “Supervised hashing with kernels,” in *CVPR*, 2012, pp. 2074–2081.
- [52] W. Liu *et al.*, “Discrete graph hashing,” in *Advances in neural information processing systems*, 2014, pp. 3419–3427.

- [53] “Class-wise supervised hashing with label embedding and active bits,” 2016.
- [54] X. Shi *et al.*, “Kernel-based supervised discrete hashing for image retrieval,” in *European Conference on Computer Vision*. Springer, 2016, pp. 419–433.
- [55] M. Norouzi, D. J. Fleet, and R. R. Salakhutdinov, “Hamming distance metric learning,” in *Advances in neural information processing systems*, 2012, pp. 1061–1069.
- [56] Y. Cao *et al.*, “Deep quantization network for efficient image retrieval.” in *AAAI*, 2016.
- [57] B. Klein and L. Wolf, “In defense of product quantization,” *arXiv preprint arXiv:1711.08589*, 2017.
- [58] H. Jégou *et al.*, “Searching in one billion vectors: re-rank with source coding,” in *ICASSP*, 2011, pp. 861–864.
- [59] A. Babenko and V. Lempitsky, “The inverted multi-index,” in *CVPR*, 2012, pp. 3069–3076.
- [60] A. Babenko and V. Lempitsky, “Efficient indexing of billion-scale datasets of deep descriptors,” in *CVPR*, 2016, pp. 2055–2063.
- [61] J. L. Bentley, “Multidimensional binary search trees used for associative searching,” *Communications of the ACM*, vol. 18, no. 9, pp. 509–517, 1975.
- [62] C. Silpa-Anan and R. Hartley, “Optimised kd-trees for fast image descriptor matching,” in *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*. IEEE, 2008, pp. 1–8.
- [63] J. Wang *et al.*, “Trinary-projection trees for approximate nearest neighbor search,” *IEEE transactions on pattern analysis and machine intelligence*, vol. 36, no. 2, pp. 388–403, 2014.

- [64] A. Babenko and V. Lempitsky, “Product split trees,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 6214–6222.
- [65] A. S. Razavian *et al.*, “A baseline for visual instance retrieval with deep convolutional networks,” *arXiv preprint arXiv:1412.6574*, 2014.
- [66] K. Mopuri and R. Babu, “Object level deep feature pooling for compact image representation,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 2015, pp. 62–70.
- [67] G. Tolias, R. Sivic, and H. Jégou, “Particular object retrieval with integral max-pooling of cnn activations,” in *Proceedings of International Conference on Learning Representations*, 2016.
- [68] E. Mohedano *et al.*, “Bags of local convolutional features for scalable instance search,” in *Proceedings of the 2016 ACM on International Conference on Multimedia Retrieval*, ser. ICMR ’16. New York, NY, USA: ACM, 2016, pp. 327–331. [Online]. Available: <http://doi.acm.org/10.1145/2911996.2912061>
- [69] S. D. Bhattacharjee *et al.*, “Query-adaptive small object search using object proposals and shape-aware descriptors,” *IEEE Transactions on Multimedia*, vol. 18, no. 4, pp. 726–737, 2016.
- [70] J. Meng *et al.*, “From keyframes to key objects: Video summarization by representative object proposal selection,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 1039–1048.
- [71] J. Cao *et al.*, “Where to focus: Query adaptive matching for instance retrieval using convolutional feature maps,” *arXiv preprint arXiv:1606.06811*, 2016.

- [72] L. Cao *et al.*, “Submodular video hashing: a unified framework towards video pooling and indexing,” in *Proceedings of the 20th ACM international conference on Multimedia*. ACM, 2012, pp. 299–308.
- [73] G. Ye *et al.*, “Large-scale video hashing via structure learning,” in *Proceedings of the IEEE International Conference on Computer Vision*, 2013, pp. 2272–2279.
- [74] G. Wu *et al.*, “Unsupervised deep video hashing with balanced rotation.” IJCAI, 2017.
- [75] V. E. Liong *et al.*, “Deep video hashing,” *IEEE Transactions on Multimedia*, vol. 19, no. 6, pp. 1209–1219, 2017.
- [76] X. Liu *et al.*, “Deep hashing with category mask for fast video retrieval,” *CoRR*, vol. abs/1712.08315, 2017. [Online]. Available: <http://arxiv.org/abs/1712.08315>
- [77] A. Iscen *et al.*, “Efficient diffusion on region manifolds: Recovering small objects with compact cnn representation,” in *CVPR*, 2017.
- [78] R. Tao, A. W. Smeulders, and S.-F. Chang, “Attributes and categories for generic instance search from one example,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 177–186.
- [79] R. Tao *et al.*, “Locality in generic instance search from one example,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2014, pp. 2091–2098.
- [80] S. D. Bhattacharjee *et al.*, “Query adaptive instance search using object sketches,” in *Proc. of the ACM on Multimedia Conference*, 2016, pp. 1306–1315.

- [81] C.-Z. Zhu and S. Satoh, “Large vocabulary quantization for searching instances from videos,” in *Proc. of the ACM ICMR*, 2012, p. 52.
- [82] Y. Yang and S. Satoh, “Efficient instance search from large video database via sparse filters in subspaces,” in *ICIP*. IEEE, 2013, pp. 3972–3976.
- [83] H. Jain *et al.*, “Subic: A supervised, structured binary code for image search,” in *ICCV*, 2017, pp. 833–842.
- [84] X. Wang, Y. Shi, and K. M. Kitani, “Deep supervised hashing with triplet labels,” in *ACCV*. Springer, 2016, pp. 70–84.
- [85] Q.-Y. Jiang and W.-J. Li, “Asymmetric deep supervised hashing,” *AAAI*, 2018.
- [86] J. Martinez *et al.*, “Revisiting additive quantization,” in *European Conference on Computer Vision*. Springer, 2016, pp. 137–153.
- [87] R. Zhang *et al.*, “Bit-scalable deep hashing with regularized similarity learning for image retrieval and person re-identification,” *IEEE TIP*, vol. 24, no. 12, pp. 4766–4779, 2015.
- [88] F. Zhao *et al.*, “Deep semantic ranking based hashing for multi-label image retrieval,” in *CVPR*, 2015, pp. 1556–1564.
- [89] Z. Zhang, Y. Chen, and V. Saligrama, “Efficient training of very deep neural networks for supervised hashing,” in *CVPR*, 2016, pp. 1487–1495.
- [90] W.-J. Li, S. Wang, and W.-C. Kang, “Feature learning based deep supervised hashing with pairwise labels,” *arXiv preprint arXiv:1511.03855*, 2015.
- [91] H. Zhu *et al.*, “Deep hashing network for efficient similarity retrieval,” in *AAAI*, 2016.

- [92] F. Shen *et al.*, “Supervised discrete hashing,” *IEEE T-PAMI*, vol. 35, no. 12, pp. 2916–2929, 2013.
- [93] Y. Chen, T. Guan, and C. Wang, “Approximate nearest neighbor search by residual vector quantization,” *Sensors*, vol. 10, no. 12, pp. 11 259–11 273, 2010.
- [94] K. He *et al.*, “Deep residual learning for image recognition,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.
- [95] A. Gersho and R. M. Gray, *Vector quantization and signal compression*. Springer Science & Business Media, 2012, vol. 159.
- [96] Z. Chena *et al.*, “Deep hashing via discrepancy minimization,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 6838–6847.
- [97] C. L. M. D. F. René and V. A. R. G. D. Hager, “Temporal convolutional networks for action segmentation and detection,” in *ICCV*, 2017.
- [98] R. Hou, R. Sukthankar, and M. Shah, “Real-time temporal action localization in untrimmed videos by sub-action discovery,” in *BMVC*, 2017.
- [99] Y. Zhao *et al.*, “Temporal action detection with structured segment networks,” in *ICCV*, 2017.
- [100] Z.-H. Yuan *et al.*, “Temporal action localization by structured maximal sums.” in *CVPR*, 2017.
- [101] S. Ioffe and C. Szegedy, “Batch normalization: Accelerating deep network training by reducing internal covariate shift,” *arXiv preprint arXiv:1502.03167*, 2015.

- [102] Y. Xia *et al.*, “Sparse projections for high-dimensional binary codes,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 3332–3339.
- [103] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” *arXiv preprint arXiv:1409.1556*, 2014.
- [104] Y. Gao *et al.*, “Compact bilinear pooling,” *arXiv preprint arXiv:1511.06062*, 2015.
- [105] R. Arandjelovic and A. Zisserman, “All about vlad,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2013, pp. 1578–1585.
- [106] J. Wang *et al.*, “Locality-constrained linear coding for image classification,” in *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*. IEEE, 2010, pp. 3360–3367.
- [107] O. Chum *et al.*, “Total recall: Automatic query expansion with a generative feature model for object retrieval,” in *Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on*. IEEE, 2007, pp. 1–8.
- [108] G. Tolias, Y. Avrithis, and H. Jégou, “To aggregate or not to aggregate: Selective match kernels for image search,” in *Proceedings of the IEEE International Conference on Computer Vision*, 2013, pp. 1401–1408.
- [109] A. S. Razavian *et al.*, “Cnn features off-the-shelf: an astounding baseline for recognition,” in *Computer Vision and Pattern Recognition Workshops (CVPRW), 2014 IEEE Conference on*. IEEE, 2014, pp. 512–519.
- [110] A. Mikulík *et al.*, “Learning a fine vocabulary,” in *European Conference on Computer Vision*. Springer, 2010, pp. 1–14.

- [111] L. Zheng *et al.*, “Query-adaptive late fusion for image search and person re-identification,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 1741–1750.
- [112] L. Zheng, Y. Yang, and Q. Tian, “Sift meets cnn: A decade survey of instance retrieval,” *arXiv preprint arXiv:1608.01807*, 2016.
- [113] L. Liu, L. Wang, and X. Liu, “In defense of soft-assignment coding,” in *2011 International Conference on Computer Vision*. IEEE, 2011, pp. 2486–2493.
- [114] S. Williams *et al.*, “Optimization of sparse matrix–vector multiplication on emerging multicore platforms,” *Parallel Computing*, vol. 35, no. 3, pp. 178–194, 2009.
- [115] H. Jégou and O. Chum, “Negative evidences and co-occurrences in image retrieval: The benefit of pca and whitening,” in *European Conference on Computer Vision*. Springer, 2012, pp. 774–787.
- [116] T. Kanungo *et al.*, “An efficient k-means clustering algorithm: Analysis and implementation,” *IEEE Transactions on PAMI*, vol. 24, no. 7, pp. 881–892, 2002.
- [117] Y. Gong and S. Lazebnik, “Iterative quantization: A procrustean approach to learning binary codes,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, 2011, pp. 817–824.
- [118] W. Hong, J. Yuan, and S. D. Bhattacharjee, “Fried binary embedding for high-dimensional visual features,” in *CVPR*, 2017.
- [119] J. C. Gower, “Procrustes methods,” *Wiley Interdisciplinary Reviews: Computational Statistics*, vol. 2, no. 4, pp. 503–508, 2010.

- [120] K. He, F. Wen, and J. Sun, “K-means hashing: An affinity-preserving quantization method for learning binary compact codes,” in *CVPR*, 2013, pp. 2938–2945.
- [121] Y. Gong *et al.*, “Learning binary codes for high-dimensional data using bilinear projections,” in *CVPR*, 2013, pp. 484–491.
- [122] F. Yu *et al.*, “Circulant binary embedding,” in *ICML*, 2014, pp. 946–954.
- [123] Z. Tu *et al.*, “A combined post-filtering method to improve accuracy of variational optical flow estimation,” *Pattern Recognition*, vol. 47, no. 5, pp. 1926–1940, 2014.
- [124] A. Ng, “Sparse autoencoder,” *CS294A Lecture notes*, vol. 72, pp. 1–19, 2011.
- [125] A. Makhzani and B. J. Frey, “k-sparse autoencoders,” *CoRR*, vol. abs/1312.5663, 2013. [Online]. Available: <http://arxiv.org/abs/1312.5663>
- [126] J. Mairal *et al.*, “Online dictionary learning for sparse coding,” in *Proceedings of the 26th annual international conference on machine learning*. ACM, 2009, pp. 689–696.