MINING CANONICAL VIEWS FROM INTERNET IMAGE COLLECTIONS

by

LIN YANG

JOHN K. JOHNSTONE, COMMITTEE CO-CHAIR
CHENGCUI ZHANG, COMMITTEE CO-CHAIR
ALLAN C. DOBBINS
YIJUAN LU
ANTHONY SKJELLUM

A DISSERTATION

Submitted to the graduate faculty of The University of Alabama at Birmingham,
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy

BIRMINGHAM, ALABAMA

2011

MINING CANONICAL VIEWS FROM INTERNET IMAGE COLLECTIONS

LIN YANG

COMPUTER AND INFORMATION SCIENCES

ABSTRACT

Recent years have seen an explosion of internet image collections. The explosion was brought about by the popularity of photo sharing sites such as Flickr and Picasa Web Albums, where millions of internet users upload their personal photos online and share these photos with the public. The wealth of images provides an excellent resource for perceiving the world through the eyes of others. However, the gigantic volume of images also poses a challenge to the consumption of this unorganized visual information.

In this dissertation, we present research on canonical view mining. Given an image collection, we leverage a combination of computer vision and data mining techniques to infer and remove images of noisy and redundant views. The remaining images, which we term canonical views, exhibit both representativeness and diversity in image content, and form a succinct visual summary of the original image collection.

The main contribution of this dissertation is the development and evaluation of a fully automatic pipeline for canonical view mining. We also demonstrate two applications of canonical views in the context of image browsing and object recognition. Finally, we analyze the scalability of the pipeline for canonical view mining and propose an approximation algorithm that effectively removes the scalability bottleneck with low impact on the resulting canonical views.

Keywords: canonical views, internet images, the wisdom of crowds, place recognition, scalable image matching, kd-tree

TABLE OF CONTENTS

Page

ACKNOWLEDGMENTS

The support of many people has made the past five years a memorable experience. Firstly, I would like to thank my co-advisors, professors John Johnstone and Chengcui Zhang, for their guidance and inspiration. I have learned many things from them about computer vision, data mining, and how to do research in general. I am also very grateful to my dissertation committee, professors Allan Dobbins, Yijuan Lu and Anthony Skjellum, for their insights and careful examination of this work.

I thank all my colleagues and friends in the GRAIL lab: David O'Gwynn, Sagar Thapaliya, Amin Hassani, and the KDDM lab: Wei-Bang Chen, Song Gao, Richa Tiwari, Liping Zhou, Xin Chen, with whom I have enjoyed collaboration as well as many fun events. I also had the opportunity to work with a group of talented people during my internships at Google. I owe my special thanks to James Wu, who mentored my first industrial job.

I am deeply indebted to my parents for always encouraging me to pursue my dreams and giving me unconditional support to do so.

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

ANN	approximate nearest neighbor
BBF	best bin first
BLEU	bilingual evaluation understudy
DoG	difference of Gaussians
LSA	latent semantic analysis
PCA	principal component analysis
pLSA	probabilistic latent semantic analysis
RANSAC	random sample consensus
ROUGE	recall-oriented understudy for gisting evaluation
SfM	structure from motion
SIFT	scale-invariant feature transform
SSD	sum of squared distances
SURF	speeded up robust features
SVD	singular value decomposition
tf-idf	term frequency-inverse document frequency
VERT	video evaluation by relevant threshold

CHAPTER 1

INTRODUCTION

Recent years have seen an explosion of internet image collections. The explosion was brought about by the vast popularity of photo sharing sites such as Flickr [4], Picasa Web Albums [14], and Facebook [3]. Millions of internet users upload their personal photos online and share the photos with the public. The volume of internet image collections has grown to multiple billions, and it keeps growing at a staggering speed.

Internet image collections provide an excellent resource for perceiving the world through the eyes of others. Nowadays, if someone is planning a trip to Rome, not only can she gain a rich resource of textual information from online articles about Rome, but she can also gain a wealth of visual information from internet image collections. By initiating a search for the keyword `Rome` on Flickr, one gains nearly three million images in the search results. This gigantic collection of images is contributed by a huge body of internet users, captured from all possible angles, through all kinds of events. The wealth of visual information, *presented properly*, offers an extremely rich experience of Rome.

Current tools, however, are not optimal for presenting internet image collections. Photo sharing sites employ a text-based paradigm for serving images. Upon uploading an image, the owner can annotate the image with a list of keywords (*tags*)

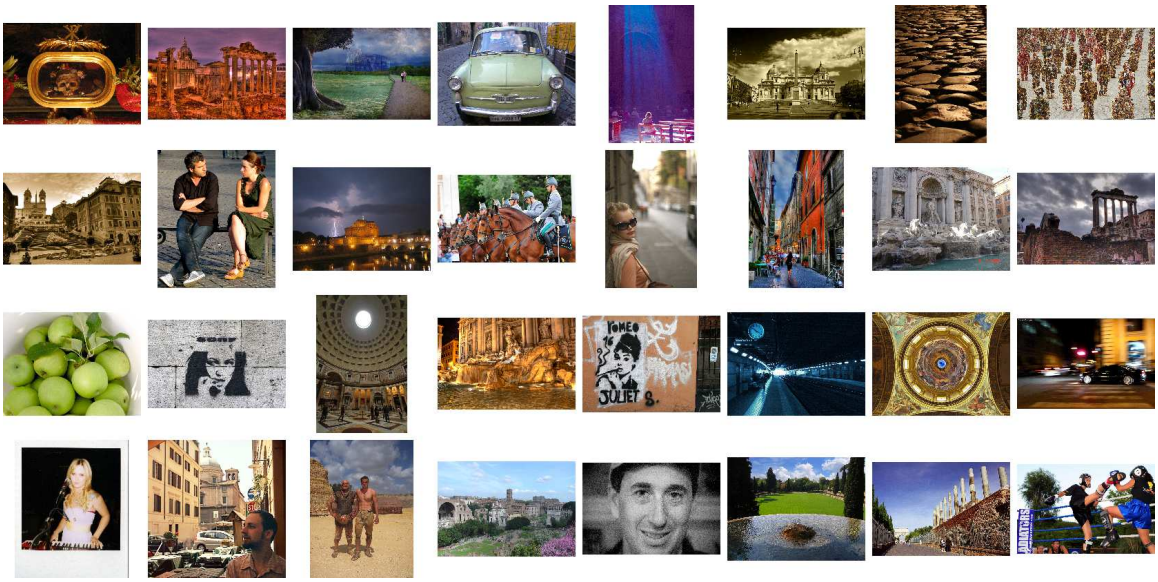


Figure 1.1: The first page of search results for the keyword `Rome` on Flickr. Even though all the retrieved images are annotated with the tag `Rome`, most of them have irrelevant content (or are at least not recognizable as Rome).

to describe the content of the image. From the search engine's perspective, the annotation of tags effectively converts the image to a text document. Henceforth a text search engine can handle the remaining tasks: on the back end of the search engine, images are indexed by tags, often in a inverted index structure to improve query-time efficiency [108]; on the front end of the search engine, a user initiates a query by entering one or more keywords. The search engine retrieves the images annotated with the same keywords and presents these images to the user as search results. The presentation often assumes the form of pages and pages of thumbnails linking to the original images.

While the text-based paradigm has demonstrated enormous success in web search engines such as Google [7], it fails to achieve comparable performance for

image search: even the top-ranked image search results contain a massive amount of noise (images of irrelevant content to the query keywords). The reasons are twofold: (1) Image tags are sparse. Unlike a text document (such as a webpage) which often consists of hundreds or thousands of words, an image is often annotated with only a few tags. A recent study by Sigurbjörnsson and van Zwol [90] reports that 64% of Flickr images are annotated with ≤ 3 tags. The sparsity of tag annotation results in incomplete interpretation of image content. (2) Image tags are inaccurate. There is no predefined dictionary or ontology for tag annotation. People are free to annotate an image with any tag that makes sense to them. The lack of specificity causes inaccuracies. For example, an image taken through the window of a plane may be annotated with `plane`, whereas the search engine user may expect the image to capture the physical appearance of a plane. A recent study by Kennedy *et al.* [63] reports that only about 50% of the tags annotated to a Flickr image actually describe the content of the image. Because of the sparsity and inaccuracy of tag annotation, noise abounds in image search results. Figure 1.1 shows the first page of search results for the keyword `Rome` on Flickr. Even though all the images are annotated with the tag `Rome`, most of them have irrelevant content (or are at least not recognizable as Rome). The images relevant to Rome are scattered across thousands of pages of thumbnails. If a user has little knowledge of Rome, it is likely that she will comb through several pages of thumbnails (hundreds of images) without seeing the big picture of Rome.

Noise is just one side of the problem in image search results. Since internet image collections have grown to a gigantic volume, for any non-obscure keyword, it is quite likely that a huge number of images are truly relevant to the keyword. Let us

assume that we have an ideal solution for measuring the relevance of images to the query keywords, and accordingly rank the images relevant to the keyword `Rome`. Since images of similar content should be assigned similar relevance scores, there would be blocks of redundant views in the ranking. Given the volume of images relevant to Rome, the first many pages of search results would be dominated by a few popular landmarks, each contributing a large number of images, while the other significant aspects of Rome would be greatly demoted. The redundancy issue is not obvious in Figure 1.1, because the text-based search results are dominated by random noise. In our experiments, we observe a high volume of redundancy when the images are ranked by a more sophisticated relevance metric (see Chapter 4). The sheer volume of internet image collections determines that presenting *all* relevant images for human browsing is not a viable solution.

Approach and Contributions

We present *canonical view mining*. Given an image collection, we leverage a combination of computer vision and data mining techniques to automatically infer and remove images of noisy and redundant views. The remaining images, or the *canonical views*, exhibit both representativeness and diversity in their photographed views, and form a succinct visual summary of the original image collection.

We make the key observation that the representativeness of a view can be inferred without knowing high-level semantics of the image (*i.e.*, what photographed objects are in this image), which is a persistent yet unsolvable problem in computer

vision [36]. Rather, we draw help from a large body of photographers to infer the representativeness of views: we make the assumption that each photographer captures views that she considers to be representative, and thereby infer representative views by analyzing the scene distribution in the entire image collection. In this way, the difficult problem of inferring high-level semantics from a single image is bypassed by detecting and aggregating similar views among a collection of images, for which automatic and robust tools are at hand. The success of this approach is a powerful demonstration of *the wisdom of crowds* [96]: the aggregation of opinions within a crowd results in information that is otherwise difficult to obtain.

Pipeline

The pipeline for canonical view mining consists of two main components (Figure 1.2):

1. Image encoding and matching. The input images are encoded and pairwise matched by scale-invariant feature transform (SIFT) [73]. The matched features between each pair of images are verified by a geometric constraint to ensure robustness [44]. The matching of SIFT features establishes the similarity metric among images. With the similarity metric defined, a similarity graph can be formed over the image collection, with vertices representing images and weighted edges indicating the similarity between images. Several large connected components of the similarity graph are visualized in the central part of Figure 1.2 for one of the experimental datasets (for clarity, small connected

components are trimmed; the visualization is generated by GraphViz [11]). In the visualization, the lengths of edges are inversely proportional to their weights. Therefore nearby vertices are likely to share redundant views. The similarity graph serves as the basis for subsequent representative view and canonical view ranking.

2. Representative view and canonical view ranking. First of all, a ranking of representative views is computed. Relying on the wisdom of crowds, we interpret a representative view as an image whose photographed scene is shared by many other images. In the similarity graph, such images are characterized by vertices that connect to many neighbors with high-weight edges. The representativeness of each image is quantified by the eigenvector centrality of the corresponding vertex in the similarity graph and solved using the power method [42]. The ranking of representative views, unfortunately, contains a massive amount of redundant views. A reranking scheme is applied to images to demote redundant views. The reranking scheme, which is based on adaptive non-maximal suppression [27], forces the top-ranked images to be not only representative, but also diverse. The output of the pipeline is a ranking of canonical views over the image collection.

Contributions

The main contribution of this dissertation is the development and evaluation of a fully automatic pipeline for canonical view mining. Secondly, we demonstrate

two applications of canonical views in the context of image browsing and object recognition. Finally, we analyze the scalability of the pipeline for canonical view mining and propose an approximation algorithm that effectively removes the scalability bottleneck with low impact on the resulting canonical views.

Automatic canonical view mining. We develop a fully automatic pipeline for canonical view mining. The pipeline is completely data-driven and requires no user input. Specifically, it does not require parameter tuning across different input datasets, which would be a painstaking process given the enormous amount of input datasets that can be obtained from internet image collections. Moreover, the pipeline does not require the number of canonical views to be known *a priori*. Instead, it computes a ranking of canonical views such that the top-ranked images are both representative and diverse, thereby approximating canonical views in a range of granularities. Once the ranking is computed offline, any number of canonical views for any subset (including the full set) of the image collection can be retrieved in real-time.

Evaluations of canonical views. Besides showing qualitative results of canonical views, we introduce three quantitative measurements to evaluate the canonical views by quantifying the amount of noise, redundancy, and summarization power for the top-ranked canonical views. Based on the quantitative measurements, we evaluate the pipeline for canonical view mining on a variety of datasets, and compare the proposed pipeline to several other methods, including the search engines of Flickr [4], Google Images [9] and the previous work of [53, 92].

Applications of canonical views. We demonstrate the applications of canonical views in the context of image browsing and object recognition. First of all, we

discuss the incorporation of the pipeline for canonical view mining with current image search engines, so that canonical views can be retrieved from image search results in *real-time* and presented to the user for enhanced image browsing. Secondly, we extend the applications of canonical views beyond image browsing, to non-parametric object recognition (object recognition based on nearest neighbor search instead of parametric modeling for object classes). By removing noise and redundancy from the database, we expect the set of canonical views to compress the database into a compact representation while still preserving most of the representative power for the purpose of object recognition. We validate this hypothesis on the place recognition problem, in which we estimate the geographic location of a query image by scene matching to a large database of images of known locations. By leveraging canonical views, we observe a significant improvement in the efficiency of query processing with minimal loss in the success rate of place recognition.

Scalable image matching / canonical view mining. Finally, we analyze the scalability of the pipeline for canonical view mining. We single out the stage of pairwise image matching as the scalability bottleneck and propose an approximation algorithm to remove the bottleneck. We evaluate the approximation algorithm by efficiency and accuracy for pairwise image matching. We demonstrate that the approximation algorithm speeds up pairwise image matching (and the entire pipeline for canonical view mining) by two orders of magnitude with low impact on the resulting canonical views.

Scope and Limitations

One fundamental assumption of canonical view mining is the validity of wisdom of crowds in the input image collection. That is, the input image collection should be contributed by a large body of independent sources as opposed to one centralized source. Internet image collections, being contributed by millions of internet users, make excellent input for canonical view mining. Obviously, not all image collections fulfill this assumption. For example, several industrial applications have initiated large-scale image acquisition with a unified procedure. One of the most prominent examples, the Google Street View [10], captures omnidirectional street level imagery at fixed intervals from cameras mounted on a vehicle. In a large city, Google Street View probably provides a denser coverage than internet image collections. However, the images in Google Street View reveal no wisdom of *crowds*. The scenes captured by these images form a roughly uniform distribution, which makes it impossible to infer which views are representative for the city.

Another assumption of canonical view mining is the existence of a robust similarity metric among the input images. Theoretically, any visual feature (such as color, texture) extracted from the images or even the metadata attached to the images (such as tags) can be used to establish the similarity metric. However, few of these features can *robustly* match the same photographed objects across different views. The development of robust features for image matching is an ongoing research topic in computer vision [102]. In this dissertation, we do not intend to improve the state of the art of image matching. Rather, we conduct a comparative study

on two widely used visual features on a controlled dataset and select the one that offers a better accuracy for image matching (see Chapter 3). The selected visual feature, SIFT, demonstrates extremely high precision and recall for matching objects of *rigid appearances*. On the other hand, SIFT cannot match object classes such as animals and plants, whose appearances undergo nonrigid transformations (deformations) from one image to another. Therefore the pipeline for canonical view mining is only tested on image collections of rigid objects. Fortunately, the class of rigid objects covers the most interesting targets for canonical view mining. In this dissertation, we demonstrate canonical view mining in three image categories: images of *places* (such as cities, national parks, landmarks), images of *products* (such as commercial advertisements), and images of *artworks* (such as paintings and sculptures).

Structure of Dissertation

The remainder of this dissertation is organized as follows. In Chapter 2, we review the study of canonical views in the human vision and computer vision literature. In Chapter 3, we lay down the groundwork for canonical view mining by establishing the similarity metric among images. In Chapter 4, we describe the algorithm for canonical view mining, and demonstrate the application of canonical views for large-scale image browsing. In Chapter 5, we systematically evaluate the quality of canonical views and compare the proposed pipeline for canonical view mining to other methods. In Chapter 6, we extend the applications of canonical views to non-parametric object recognition, and demonstrate it on a specific problem of place

recognition. In Chapter 7, we recognize the scalability bottleneck in the pipeline for canonical view mining, and propose an approximation algorithm to remove the bottleneck. In Chapter 8, we conclude this dissertation and present ideas for future work.

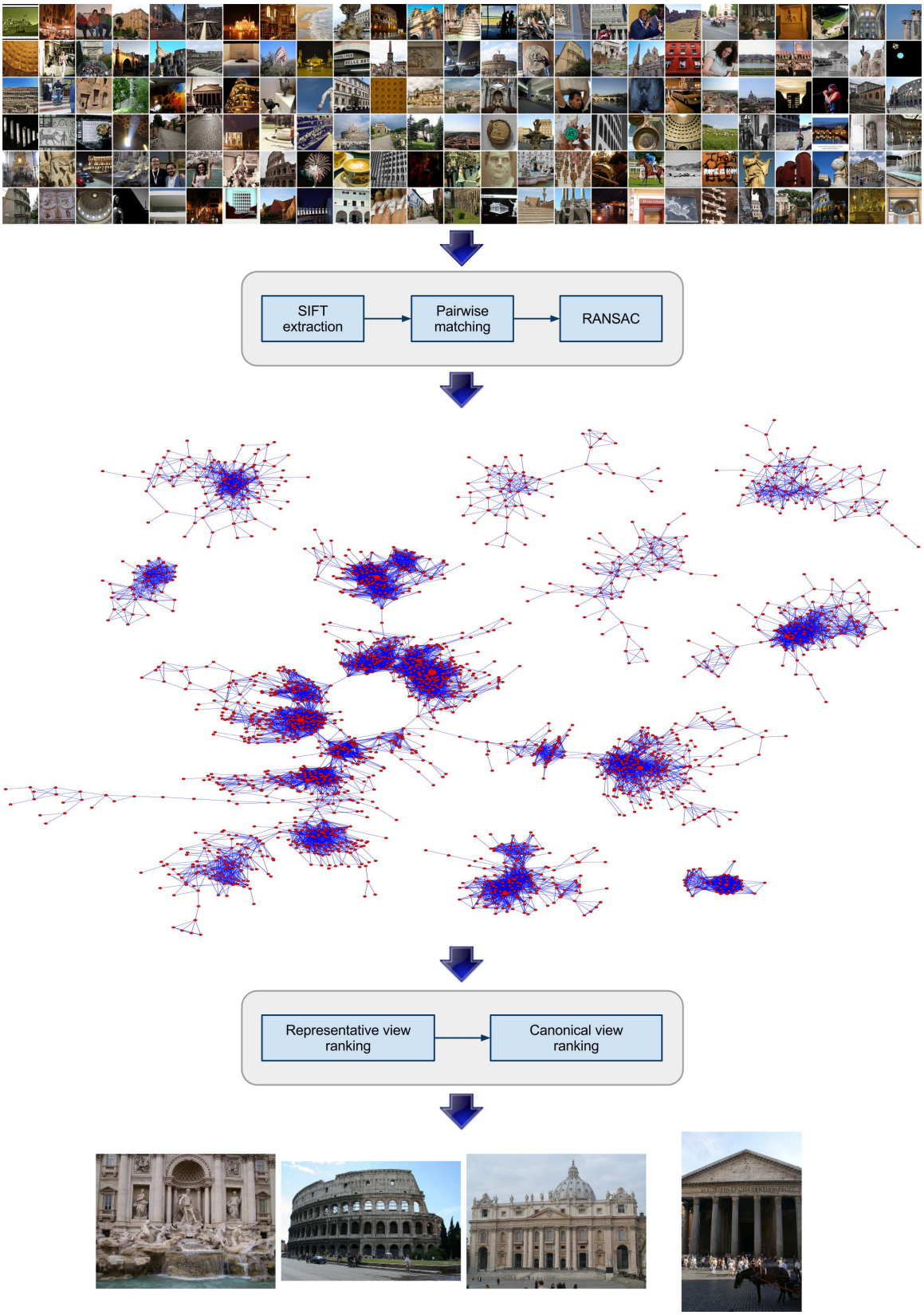


Figure 1.2: The pipeline of canonical view mining. See text for details.

CHAPTER 2

PREVIOUS WORK

In this chapter, we review the study of canonical views in the human vision and computer vision literature. The early work on canonical views originated in the human vision literature, where it focused on finding preferred viewpoints for familiar objects. We briefly review the early work and discuss its applicability to internet image collections. In the computer vision literature, the study of canonical views has recently gained popularity with the proliferation of internet image collections. By fundamental approach to the problem, we divide previous work into clustering-based methods and ranking-based methods. A clustering-based method groups images into visually proximal clusters and computes a set of canonical views by selecting images from different clusters. A ranking-based method computes a ranking of images, such that the top k images approximate a k-set of canonical views for the image collection. We review each category by discussing one representative work in detail, followed by a brief description of the rest. We conclude this chapter by acknowledging a recent thread of work that tackles large-scale image browsing in a different approach, by reconstructing the photographed scene from the images and enabling scene browsing in the 3D space.

Early Work

The study of canonical views predates internet image collections. The early work originated in the study of human vision. The term *canonical view* was first used by Palmer *et al.* in their seminal work on finding preferred viewpoints for familiar objects [81]. In their experiments, Palmer *et al.* showed each object in 12 viewpoints (front, back, side, top, and intermediate viewpoints at 45° angles to these) and asked human observers to perform a series of tasks:

- Assign a goodness rating to images captured from different viewpoints of the object.
- Identify the viewpoint that agrees with their mental image of the object.
- Actively select a viewpoint to photograph the object.
- Name the object as quickly as possible from different viewpoints.

The most preferred viewpoint by human observers is defined as the *canonical view* for the object. Palmer *et al.* found that the canonical views are consistent across all four tasks. Specifically, human observers prefer off-axis views in viewing, imagining, photographing, and recognizing objects. According to their explanation, off-axis views are consistently preferred because they can maximize the amount of visible surface while avoiding self-occlusion. The work of Palmer *et al.* was followed by a number of experiments in a similar setting with human observers performing a series of viewing tasks and rating preferred viewpoints in each task [43, 83, 84].

Unfortunately, the early work has very limited applicability to internet image collections. First of all, the canonical views in the early work are defined and selected by surveying human observers, which is not affordable on the scale of internet image collections. Secondly, the geometric property of canonical views (maximizing the amount of visible surface and avoiding self-occlusion) is not applicable to many photographed objects in internet image collections. For example, a canonical view for a famous landmark should be an image that captures its iconic appearance, which does not necessarily maximize the amount of visible surface or avoid self-occlusion. Thirdly, the early work selects a single canonical view, and the selection takes place in a small number of viewpoints of a single object. On the other hand, an internet image collection may contain tens of thousands of images of countless photographed objects, and in most cases, multiple canonical views are required to form a reasonable summary for the entire collection.

Clustering-based Methods

Most previous work in the computer vision literature reduces canonical view mining to a clustering problem. Given a collection of images, the fundamental idea of clustering-based methods is to group images into visually proximal clusters. Ideally, images of the same cluster all share the same view (therefore the view is frequently photographed), and images of different clusters share no view in common. Therefore one image (often the image corresponding to the centroid of a cluster) can be selected from each cluster to form a set of canonical views for the image collection.

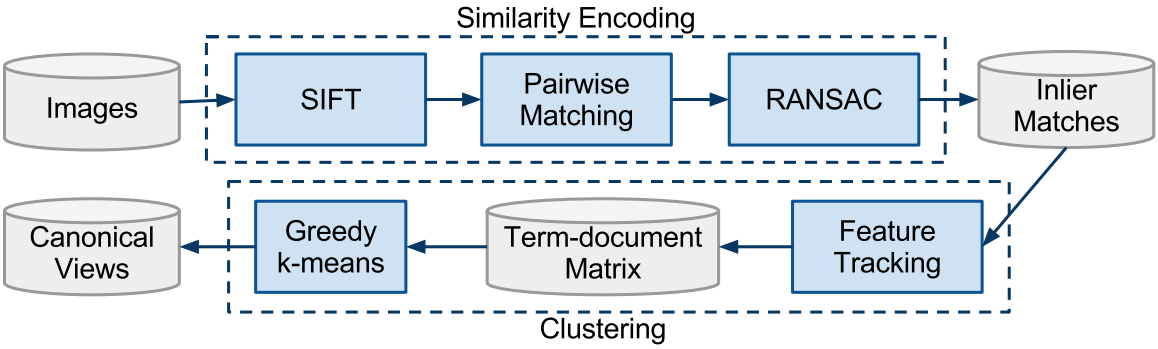
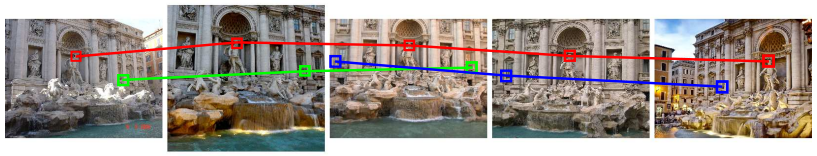


Figure 2.1: The pipeline of canonical view selection [92].

One representative work of clustering-based methods is the work of Simon *et al.* [92]. Simon *et al.* study canonical view selection for images of tourist attractions. The pipeline of their algorithm is illustrated by Figure 2.1.

Given an image collection, the algorithm starts by encoding and matching images in a pairwise manner: SIFT features are extracted from images and used for image matching [73]; the SIFT matches between a pair of images are verified by a geometric constraint using Random Sample Consensus (RANSAC) [44]. The result of this step is a set of verified SIFT matches among all images. A detailed description of SIFT feature extraction and matching can be found in Chapter 3.

The verified SIFT matches among all images are organized into *tracks*. A track consists of multiple SIFT features that correspond to the same object point. A term-document matrix [32] is formed where terms correspond to SIFT tracks and documents correspond to images. In the term-document matrix, a cell of non-zero value indicates that the corresponding term (SIFT track) appears in the corresponding document (image). By definition, a SIFT track can only appear in an image 0 or 1



	1	2	3	4	5
R	1	1	1	1	1
G	1	1	1	0	0
B	0	0	1	1	1

Figure 2.2: Organizing SIFT matches into tracks and a term-document matrix. On the left, the SIFT matches corresponding to three object points are organized into three tracks. On the right, a term-document matrix is formed, where the row headers R, G, B correspond to the red, green, blue tracks, and the column headers 1, 2, 3, 4, 5 correspond to the five images in the order of appearance. In the term-document matrix, a cell of value 1 indicates that the corresponding term (SIFT track) appears in the corresponding document (image). This figure is best viewed in color.

times. Therefore the term-document matrix consists of 0 and 1 values. This step is illustrated by Figure 2.2.

The purpose of SIFT matching and tracking is to establish a similarity metric among the images. Let the set of images be denoted by $\mathcal{P} = \{P_1, P_2, \cdots, P_n\}$, and their corresponding columns in the term-document matrix be denoted by $\mathcal{V} = \{V_1, V_2, \cdots, V_n\}$. Simon *et al.* define the similarity between two images as the dot product of their corresponding columns in the term-document matrix:

$$similarity(P_i, P_j) = \frac{V_i \cdot V_j}{\|V_i\|\|V_j\|}. \tag{2.1}$$

The similarity values range from 0 to 1, where 0 indicates two images sharing no SIFT tracks in common, and 1 indicates two images sharing the exact same set of SIFT tracks. With the similarity metric, images can be grouped into visually proximal clusters. Simon *et al.* adopts greedy k-means [29] for the clustering step. Greedy k-means is a variant of the classic spherical k-means algorithm [33]. It demonstrates superior performance compared to spherical k-means when the data points to be clus-

tered are in a high-dimensional space and the initial seeding is not close to optimum. Greedy k-means aims to maximize the following objective function:

$$Q(\mathcal{C}) = \sum_{P_i \in \mathcal{P}} similarity(P_i, C_{c(i)}) \; - \; \alpha|\mathcal{C}| \; - \; \beta \sum_{C_i \in \mathcal{C}} \sum_{C_j \in \mathcal{C}} similarity(C_i, C_j) \qquad (2.2)$$

where $\mathcal{C} = \{C_1, C_2, \cdots, C_k\}$ is the set of canonical views, and $C_{c(i)}$ denotes the closest canonical view to image P_i: $C_{c(i)} = \arg\max_{C \in \mathcal{C}} similarity(P_i, C)$. The first term in the objective function encourages each image P_i to be represented by at least one canonical view (notice that this term alone forms the objective function for spherical k-means). The second term imposes a punishment when too many canonical views are selected. The third term imposes a punishment when the canonical views are similar to each other. α and β are user-defined parameters to balance the influences of the three terms.

Greedy k-means attempts to maximize $Q(\mathcal{C})$ in an iterative procedure. The set of canonical views is initialized to be empty. During each iteration, the remaining images are scanned to find the one that maximizes the gain in $Q(\mathcal{C})$, which is added to the canonical set. The procedure repeats until no remaining image can bring a positive gain in $Q(\mathcal{C})$, at which point $\mathcal{C}$ contains the canonical views for the image collection. The algorithm for canonical view selection is provided in Algorithm 1.

The work of Simon *et al.* illustrates the basic workflow of clustering-based methods. First of all, a similarity metric is established among images. Secondly, a clustering algorithm is applied to group images into clusters based on the similarity metric. Finally, a set of canonical views is formed by selecting images from the

Algorithm 1 Select canonical views $\mathcal{C}$ from images $\mathcal{P}$ using greedy k-means.

$\mathcal{C} \Leftarrow \emptyset$
repeat
  $\Delta Q \Leftarrow 0$, $C \Leftarrow null$
  for $P \in \mathcal{P} \backslash \mathcal{C}$ **do**
    $\Delta Q' \Leftarrow Q(\mathcal{C} \cup \{P\}) - Q(\mathcal{C})$
    if $\Delta Q' > \Delta Q$ **then**
      $\Delta Q \Leftarrow \Delta Q'$, $C \Leftarrow P$
    end if
  end for
  if $\Delta Q > 0$ **then**
    $\mathcal{C} \Leftarrow \mathcal{C} \cup \{C\}$
  end if
until $\Delta Q = 0$

Table 2.1: A summary of clustering-based methods for canonical view selection. Each row summarizes one method, with its similarity metric, clustering algorithm, and the targeted image type.

	Similarity metric	Clustering algorithm	Image type
[92]	SIFT	greedy k-means	places
[30]	SIFT	spectral clustering	places
[34]	color+texture+SIFT	minimizing within-cluster variation	any
[52]	color moments	affinity propagation	any
[54]	SIFT	degree centrality	products
[56]	visual + textual	hierarchical clustering	any
[64]	color+texture	k-means	places
[103]	color+shape+texture	folding, maxmin, reciprocal election	any
[86]	text + Gist	joint k-means	concepts
[106]	visual words	greedy selection	any

clusters. Most previous work follows this workflow, which is described in the sequel and summarized in Table 2.1.

In [30], Crandall *et al.* investigate the organization of a gigantic collection of 35 million geo-tagged images. Images are clustered by mean-shift [40] on the metropolitan level and on the landmark level. Images of each metropolitan or landmark area are encoded by SIFT features. A similarity graph is formed with vertices represent-

ing images and weighted edges indicating the similarity between images measured by their SIFT matches. Spectral clustering [89] is applied to the similarity graph. Vertices (images) are partitioned by the second eigenvector of the Laplacian matrix of the graph (the Fiedler vector). The vertex (image) with the largest weighted degree from each cluster is selected as the canonical view for the metropolitan or landmark area.

In [34], Fan *et al.* propose a novel scheme to explore internet image collections. Latent semantic analysis (LSA) [32] is adopted to extract topics from all the image tags. A topic network is formed with vertices representing tags and weighted edges indicating the similarity between tags. For each topic, a set of images annotated with the corresponding tag are grouped. The grouped images are encoded by a combination of global (color and texture) and local features, and are clustered by minimizing the trace of a within-cluster scatter matrix. A small set of images that are close to cluster centroids are selected for exploring the corresponding topic.

In [52], Jia *et al.* adopt affinity propagation [38] to select a set of exemplars that can best represent an internet image collection. A similarity graph is formed with similarities measured by color moments of images. Affinity propagation finds clusters by iteratively passing messages between each pair of neighboring vertices until a set of exemplars and corresponding clusters emerge. Affinity propagation does not allow the number of clusters (exemplars) to be specified directly. The number of clusters is influenced by a *preference* parameter associated with each vertex in the similarity graph, as well as the message-passing procedure. Therefore, the algorithm needs to

be run multiple times with different preference values in order to produce the desired number of clusters.

In [54], Jing *et al.* propose an algorithm for selecting a single iconic view for an image category of a commercial product. SIFT features are used to encode images and measure the similarity between images. A similarity graph is formed over the image category. The vertex (image) whose accumulated similarity with other vertices is the highest is selected as the iconic view for the category. Their work can be regarded as a simplified clustering-based method, where all input images are treated as a single cluster and a single canonical view is selected for the cluster.

In [56], Jing *et al.* present Google Image Swirl [8], which builds upon the Google Images search engine [9] and organizes image search results into an exemplar tree for hierarchical browsing. Upon receiving a user query, Google Image Swirl retrieves the top 1000 search results, and builds a pairwise similarity matrix among the images. The similarity among images is measured by a combination of visual features such as color, texture, local features, face features, as well as textual keywords associated with the images. Hierarchical clustering is applied to the similarity matrix to recursively partition the images into a hierarchical tree. A user starts browsing the search results from the top-level clusters and traverses down the hierarchical tree along the clusters that best matches his/her needs.

In [64], Kennedy *et al.* leverage both metadata and visual features for canonical view selection. The proposed method starts by learning spatial-temporal patterns of image tags and discovering tags related to landmarks. Tags related to landmarks appear in peaks in the spatial domain but uniformly in the temporal domain. For

each discovered landmark, images with corresponding tags are retrieved and clustered using k-means based on global color and texture features. A set of statistics on both metadata and visual features are computed to give a representativeness score to each cluster and all images within the cluster. Canonical views are selected as top-ranked images from top-ranked clusters.

In [103], Leuken *et al.* propose to diversify image search results by clustering. Similarity among images is measured by six visual features that characterize the color, shape, and texture of images. The influences of different visual features are dynamically normalized by the variances in their own domains, so that the discriminative features (with small variances) are weighed more heavily than the others. Three clustering algorithms are proposed – Folding, Maxmin, and Reciprocal Election – all of which are lightweight techniques that cluster image search results and select exemplars from all clusters to form a diverse set of images.

In [86], Raguram and Lazebnik propose to compute iconic views for images of abstract concepts (*e.g.*, images tagged with `love`). They use Gist features [80] to encode the scene structure and probabilistic latent semantic analysis (pLSA) [47] on image tags to encode the textual topic for each image and perform joint clustering based on visual and textual features. Images in a resulting cluster are both perceptually and semantically similar. Iconic views are selected by choosing the image with highest visual quality [61] from each cluster.

In [106], Yang *et al.* propose a lightweight technique for canonical view selection from image search results. After encoding images using SIFT features, Yang *et al.* collect SIFT features from all images and cluster the SIFT features using k-means

into a small set of *visual words* [93]. Each visual word corresponds to a cluster of SIFT features that are close in the feature space. Therefore an image can be represented by a bag of visual words. A list of informative visual words are selected by a variant of the *tf-idf* (term frequency-inverse document frequency) weighting scheme [20]. Canonical views are selected in a greedy manner. During each iteration, the image that provides the best coverage in the list of informative visual words is added to the canonical set. The covered visual words are removed from the list so that no redundant views will be selected in future iterations.

The clustering-based methods have demonstrated promising results of canonical views in various experimental settings. However, this class of algorithms suffers a severe disadvantage: most clustering algorithms can only generate a fixed number of clusters (canonical views) in one run. Therefore, if a different number of canonical views is desired, the algorithm must be rerun with a different parametric setting (*e.g.*, by adjusting k for k-means, α and β for greedy k-means, the preference values for affinity propagation). The only exception to this is probably the hierarchical clustering algorithm, which is able to generate clusters on discrete granularities. Even so, it does not guarantee *all* numbers of clusters to be readily available.

Ranking-based Methods

Given a collection of images, a ranking-based method computes an ordering of the images, such that the top k images approximates a k-set of canonical views for the image collection. Despite the amount of previous work on canonical view mining,

most algorithms reduce it to a clustering problem. This dissertation is one of the first to propose a ranking-based method. Besides our work, there is only one paper to our knowledge that falls into this category.

In [50], Jaffe *et al.* propose an algorithm for ranking images of a tourist attraction based on image metadata. Upon output, the top k images approximate the k-set of canonical views for the tourist attraction. The algorithm proceeds in three steps:

1. Recursively cluster images into a hierarchical tree using geo-tags.
2. Compute an importance score for each sub-cluster in the hierarchy.
3. Recursively rank sub-clusters from the leaf level to the root, generating a flat ordering of all images.

The input to the first step is the set of geo-tags associated with images:

$$\mathcal{L} = \{(x_i, y_i) \in \mathbb{R}^2,\ 1 \leq i \leq n\} \tag{2.3}$$

Jaffe *et al.* adopt the Hungarian clustering algorithm [41] to cluster $\mathcal{L}$ into a hierarchical tree. As input to the clustering algorithm, a weighted graph is formed by $\mathcal{L}$ with vertices representing geo-tags and weighted edges indicating the geographic distance between geo-tags. The Hungarian method [65] serves as the building block for the clustering algorithm. Given a weighted graph, the Hungarian method finds a set of disjoint cycles that forms a cover of the graph while minimizing the total weight. Because the total weight of the cycles is minimized, each cycle is likely to

connect only close-by points. Therefore, the cycles form the initial clusters of $\mathcal{L}$. The Hungarian clustering algorithm proceeds in a recursive manner. During each iteration, the existing clusters are viewed as data points and subjected to the Hungarian method to find higher-level cycles (clusters on a coarser granularity). The algorithm terminates at the root where one single cluster exists, producing a hierarchical tree of images.

In the second step, a score is computed for each sub-cluster in the hierarchy. The scoring strategy aims to evaluate the importance of the image content conveyed by a sub-cluster. Jaffe *et al.* consider a variety of image metadata, including: geotag, timestamp, photographer, tags, quality, and relevance. The quality and relevance metadata is derived externally, while the rest is associated with the images. Various statistics are derived from the metadata, which loosely evaluate a sub-cluster from different perspectives: relevance, semantic closeness, photographer support, geographic density, and image quality. Each statistic leads to a score from one perspective, which is incorporated into the final scoring strategy. The mathematical equations for computing these statistics are omitted in this review.

With a hierarchical clustering of images and an importance score assigned to each sub-cluster, the final goal is to generate a flat ordering of all images. A recursive interleaving algorithm is proposed. The algorithm starts from the leaf level of the hierarchy. Assuming all the images are ordered within current-level clusters, it interleaves the images in current-level clusters to produce a merged ordering in the parent level. The recursion terminates at the root, producing an ordering of the entire image collection. The interleaving of child clusters should ideally satisfy two

Figure 2.3: A screenshot from TagMaps [15]. TagMaps visualizes popular tags in a map view. At any time, only the canonical views corresponding to the selected tag are displayed (`Roma` in this example). The set of canonical views is updated in real-time as the user selects different tags or zooms in/out or translates to different regions.

properties: (1) in any section of the merged ordering, the frequencies of images from different child clusters are proportional to the scores of corresponding clusters (depth); (2) every child cluster should have one image appearing early in the merged ordering (breadth). The following procedure is proposed: the merged ordering is divided into two sections – a *header* and a *trailer*. When interleaving multiple child clusters, the first image from each child cluster is selected to compose the header section of the merged ordering (satisfying breadth). The remaining images in the child clusters are selected with probabilities proportional to the scores of corresponding clusters (satisfying depths).

After the entire image collection is ranked off-line, any number of canonical views for any subset of the image collection can be retrieved in real-time. Because

of the real-time execution, the work of Jaffe *et al.* leads to TagMaps [15], the first industrial application to our knowledge for browsing canonical views for internet image collections. TagMaps visualizes popular tags in a map view. At any time, only the canonical views corresponding to the selected tag are displayed. The set of canonical views is updated in real-time as the user selects different tags or zooms in/out or translates to different regions. A screenshot from TagMaps is shown in Figure 2.3.

However, the algorithm of Jaffe *et al.* has two shortcomings. First of all, only image metadata is considered in inferring the importance of images or sub-clusters. No visual features are involved. Therefore the similarity metric between images is highly unreliable, and the resulting canonical views tend to include many noisy and redundant views. Secondly, the algorithm is designed specifically for images of tourist attractions, which is the only case in which it makes sense to cluster images using geo-tags. It would be inappropriate to apply the algorithm on other types of images whose semantics are not correlated to their geographic locations (such as images of commercial products or artworks).

Structure From Motion

We conclude this chapter by acknowledging a recent thread of work that tackles large-scale image browsing with a different approach, by reconstructing the photographed scene from the images and enabling scene browsing in the 3D space.

Figure 2.4: An illustration of the system of Photo Tourism. Offline, the system takes as input an image collection of a tourist attraction (left) and automatically reconstructs a sparse set of 3D points and the camera parameters (center). During online exploration, Photo Tourism renders both the enhanced scene points as well as the camera viewpoints in 3D, and allows the user to select different viewpoints and browse the corresponding images (right). This figure is cited from [95].

In Photo Tourism [95], Snavely *et al.* present a novel 3D interface for exploring tourist attractions. Given an image collection of a tourist attraction, a sparse set of 3D points and the camera parameters are automatically reconstructed using structure from motion (SfM) [44]. Each image is then registered in the 3D space on top of the scene structure. During online exploration, Photo Tourism renders both the enhanced scene points as well as the camera viewpoints in 3D, and allows the user to select different viewpoints and browse the corresponding images. An illustration of the system can be found in Figure 2.4. In their recent work [94], Snavely *et al.* enhance Photo Tourism by extracting paths in 3D along which images are densely captured. Therefore Photo Tourism can automatically compute an optimal 3D fly-through for exploring the tourist attraction.

In Photo Navigator [48], Hsieh *et al.* propose a similar system. Given an image collection of a tourist attraction, they reconstruct the camera parameters in a similar way to [94, 95]. However, instead of reconstructing the scene and navigating users in

the 3D space, they employ the camera parameters to compute an optimal ordering of images such that the discontinuity between consecutive pairs of images is minimized. The reordered images, displayed in a slide show, can simulate sightseeing along the route of travel.

Scalability is a bottleneck for SfM techniques, because the reconstruction process involves pairwise image matching and bundle adjustment [100]. In [18], Agarwal *et al.* analyze the parallelization of the SfM pipeline. They explore a variety of alternative algorithms at each stage of the pipeline, and design a series of parallel distributed algorithms for image matching and bundle adjustment. They demonstrate building the 3D scene points and camera parameters for Rome from 150,000 images in less than one day on a cluster of 500 CPUs.

By browsing images in the 3D space, users have a sense of the geometric context, which can potentially help users to quickly navigate to the views of interest. However, SfM techniques do not address the problems of noise and redundancy as does canonical view mining. Although there are ways to detect and remove totally irrelevant images during the reconstruction process, all the relevant images, representative or not, redundant or not, are presented to the user.

CHAPTER 3

PRELIMINARIES IN IMAGE MATCHING

In this chapter we lay down the groundwork for canonical view mining. As discussed in Chapter 2, all previous work on canonical view mining assumes the existence of a similarity metric among images. This dissertation is no exception.

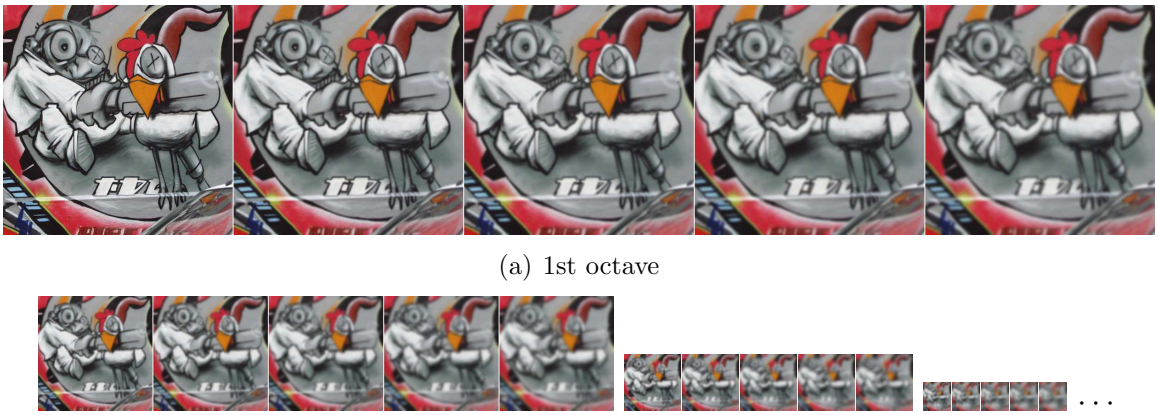
In this dissertation, we rely on local feature matching to measure the similarity between two images. As the name implies, local features encode an image at selected locations (or *feature points*) where the local image regions exhibit unique patterns. Such unique patterns allow feature points of the same objects to be distinctively encoded and robustly matched across different images. Since the matching between two images is point-based, objects can be matched even under severe occlusion and clutter, as long as the visible portions contain enough feature points. In recent years, numerous algorithms have been proposed to improve local features by making them invariant to various image transformations. The state-of-the-art local features have achieved invariance to image rotation, scaling, and even affine transformation (caused by a change in viewpoint) to some extent. Due to these reasons, local features have been widely used for image matching tasks. Their applications can be found in the areas of image retrieval [60], image mining [62], image mosaicking [26], and structure from motion (SfM) [95].

In general, there are three steps in any algorithm for local feature matching. First of all, feature points are detected at image locations where the local image regions exhibit unique patterns. Secondly, each feature point is encoded by a feature descriptor that aggregates statistics in the local image region. Finally, a measurement must be established for feature descriptors, based on which local features from different images can be matched.

For all experiments in this dissertation, the Scale-Invariant Feature Transform (SIFT) algorithm is used to detect and encode local features [73]. The matching of SIFT features establishes the similarity metric among images, based on which canonical views are mined. Notice that, a class of algorithms have been proposed (including SIFT) for local feature matching which offer different advantages such as high efficiency (*e.g.*, SURF [21]) and full affine-invariance (*e.g.*, Harris-Affine detector [74]). The performances of many state-of-the-art algorithms are systematically evaluated in [75]. Depending on the specific application or dataset, SIFT can be replaced by any other algorithm in this class for canonical view mining. In the sequel, we introduce the SIFT algorithm in greater detail. Interested readers are referred to [102] for an extensive survey on alternative algorithms.

Feature Extraction

The SIFT algorithm is proposed by Lowe [73]. Given an input image, SIFT detects feature points at local extrema (maxima and minima) in the *scale space* [71]. A scale space consists of a collection of images generated by progressively smoothing



(a) 1st octave

(b) 2nd octave (c) 3rd octave (d) 4th

Figure 3.1: An illustration of octaves in the scale space. Within each octave, the image is progressively smoothed by a Gaussian filter. At the end of the current octave, the scale space image is down-sampled by a factor of 2, and the construction of the next octave starts from the down-sampled image.

the input image by Gaussian filters. Therefore, a scale space image L is generated by:

$$L(x, y, \sigma) = G(x, y, \sigma) * I(x, y), \tag{3.1}$$

where $*$ is the convolution operator in the x, y directions between input image I and a Gaussian filter G:

$$G(x, y, \sigma) = \frac{1}{2\pi\sigma^2} e^{-(x^2+y^2)/2\sigma^2}. \tag{3.2}$$

Gaussian filtering allows high-frequency signals (fine details in the image) to be suppressed. Therefore feature points are not only extracted on the scale of the input image, but on coarser scales as well. Lowe proposes to speed up the construction of the scale space by segmenting the scale space into *octaves*. Within each octave, the image is progressively smoothed by a Gaussian filter. At the end of the current octave, the image is down-sampled by a factor of 2, and the construction of the next

octave starts from the down-sampled image. Therefore image convolutions become extremely efficient toward high levels in the scale space. An illustration of octaves in the scale space can be found in Figure 3.1. After the scale space is constructed, adjacent images in the same octave are subtracted to generate a series of Difference-of-Gaussian (DoG) images. Local extrema both in the image space and across DoG images are detected as feature points.

The principal novelty of SIFT lies in the construction of the scale space: instead of detecting feature points on the scale of the input image, it detects them through all levels in the scale space, and encodes each feature point at its "canonical" level (where it becomes a local extrema). Therefore, SIFT features are scale-invariant. For example, given two images of the same object captured at different distances or zoom levels, SIFT can still match the object by their feature points, because both sets of feature points are detected and encoded on the same canonical scales in the scale space, not on the scales of corresponding input images. Such an example is shown in Figure 3.2, where the same object captured at considerably different scales is matched across two images. The second row visualizes the local image regions based on which the matched SIFT features are encoded (see below for feature point encoding). The size of a local image region is proportional to the scale on which the feature point is detected. It can be observed that the local image regions of matched features have almost identical coverages relative to the real-world objects, even though their absolute sizes are drastically different in the images.

Once feature points are detected at integral pixel locations, their locations are refined to achieve sub-pixel accuracy by fitting a 3D quadratic and interpolating the

Figure 3.2: Matching objects on different scales using SIFT features. The first row visualizes some SIFT matches between two images of the Trevi Fountain captured at considerably different scales. Each blue line connects two matching SIFT features in the two images. The second row visualizes the local image regions based on which the matched SIFT features are encoded. Each local image region is enclosed by a green circle. It can be observed that the local image regions of matching features have almost identical coverages relative to the real-world objects, even though their absolute sizes are drastically different in the images. For clarity, only a subset of the SIFT features/matches are visualized. See Figure 3.6 for a full set of SIFT matches between the two images.

maximum. Due to the property of DoG filters, feature points are mostly located at the centers of blob-shaped regions, where a blob is characterized by a bright spot surrounded by a dark background (local maximum), or a dark spot surrounded by a

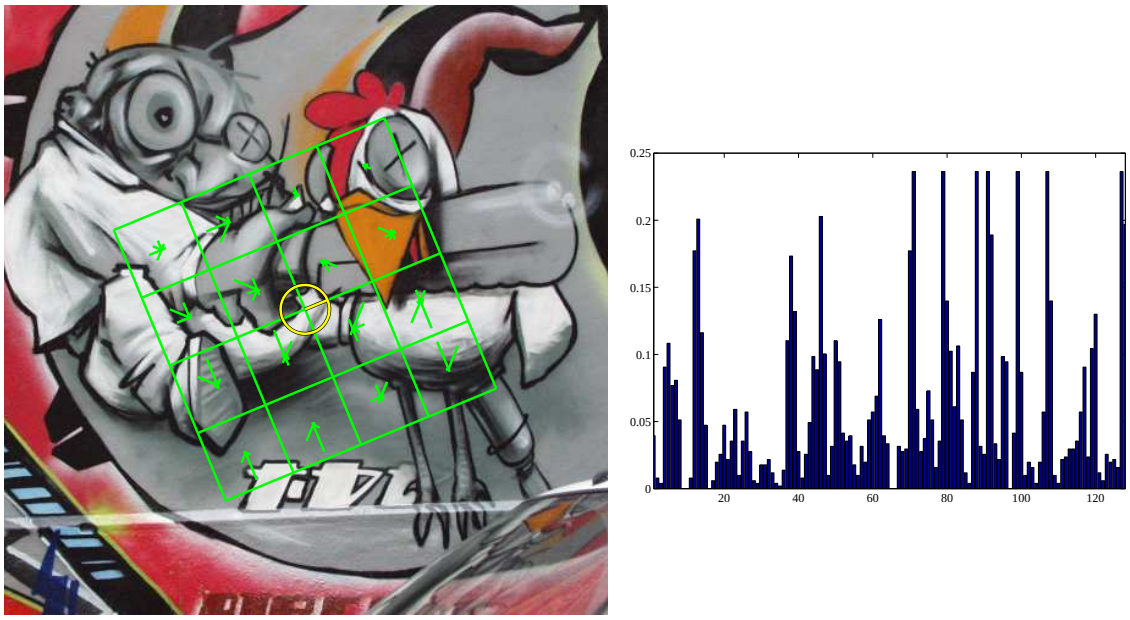


Figure 3.3: Feature descriptor computation for SIFT features. In order to encode the feature point in the left image (at the center of the yellow circle), SIFT first assigns the feature point a dominant orientation (illustrated by the line in the circle), and then computes the feature descriptor in the rotated local image patch (shown by the green square). The local image patch is divided into 4×4 blocks. Within each block, gradient magnitudes are aggregated in 8 directions (illustrated by the polar graphs in each green block). The aggregation of gradient magnitudes forms a histogram of $4 \times 4 \times 8 = 128$ bins. The histogram is normalized to unit L2-norm and serves as the feature descriptor for the feature point. The feature descriptor is shown in the right image. For clarity, the local image patch is enlarged to show details of the aggregation. The actual coverage of the feature point is smaller.

bright background (local minimum) in the image. Occasionally, DoG picks up feature points along the edges. Feature points along the edges are unstable for image matching, because their locations are sensitive to even small amounts of noise. Therefore SIFT computes for each feature point an edge response measured by the ratio between the principal local curvature and the one perpendicular, and feature points with strong edge responses are removed.

The remaining feature points are encoded by *feature descriptors*. A feature descriptor is a histogram formed by aggregating statistics in the local image region of the feature point. In order to achieve rotation-invariance, SIFT assigns each feature point a dominant orientation by aggregating local image gradient orientations. The local image patch is then rotated to align to the dominant orientation of the feature point. Feature descriptors are computed in the rotated image patches. At each feature point, SIFT divides the local image patch into 4×4 blocks. Within each block, SIFT aggregates gradient magnitudes in 8 directions. A Gaussian window is placed at the center of the local image patch to suppress the contribution of distant pixels. The aggregation of gradient magnitudes forms a histogram of $4 \times 4 \times 8 = 128$ bins. The histogram is normalized to unit L2-norm and serves as the feature descriptor for the feature point. The use of pixel gradients instead of raw pixel intensities allows SIFT feature descriptors to be invariant to brightness changes, since a constant addition to pixel intensities has no effect on pixel gradients. The normalization step further allows SIFT feature descriptors to be invariant to contrast changes, because contrast changes often correspond to multiplying each pixel intensity by a constant factor, and such changes are cancelled by the L2 normalization. A visualization of feature descriptor computation can be found in Figure 3.3.

In this dissertation, we obtained the implementation of SIFT from Lowe's website [12]. On a regular-sized image (*e.g.*, 640×480), SIFT can extract from several hundred to thousands of features. Each SIFT feature is encoded by two pieces of information: a location in the image coordinate, and a 128-dimensional feature descriptor. As we shall see in following sections, feature locations play an important

role in verifying the correctness of feature matches, while feature descriptors serve as the mechanism for matching feature points across different images.

Feature Matching

Encoded by feature descriptors, the candidate match for a feature point is identified as its nearest neighbor in the feature space measured by Euclidean distance. Let two images I_1 and I_2 be represented by their sets of SIFT feature descriptors: $I_1 = \{\boldsymbol{p}_1, \cdots, \boldsymbol{p}_m\}$, $I_2 = \{\boldsymbol{q}_1, \cdots, \boldsymbol{q}_n\}$, where $\boldsymbol{p}_i$ and $\boldsymbol{q}_j$ are 128-dimensional data points. Let NN_I be the function for nearest neighbor search for image I. That is, NN_I accepts a query descriptor from another image, and returns its nearest neighbor within all the descriptors of image I measured by Euclidean distance:

$$NN_I(\boldsymbol{q}) = \arg\min_{\boldsymbol{p} \in I} \|\boldsymbol{p} - \boldsymbol{q}\|. \tag{3.3}$$

Then the set of candidate matches $\mathcal{C}$ between images I_1 and I_2 is defined by:

$$\mathcal{C}(I_1, I_2) = \{(\boldsymbol{p}_i, \boldsymbol{q}_j) \mid \boldsymbol{p}_i \in I_1, \boldsymbol{q}_j \in I_2 \text{ s.t. } \boldsymbol{p}_i = NN_{I_1}(\boldsymbol{q}_j)\}. \tag{3.4}$$

Of course, merely being nearest neighbors cannot guarantee a true match: if the two images capture different objects, there is no true match between them at all. Yet by definition, every feature descriptor must have a nearest neighbor in the other image, no matter how large the absolute distance is between them. Lowe proposes to verify the robustness of a candidate match by a ratio test: for each feature descriptor, we

also find its *second nearest neighbor* in the other image, and the candidate match is deemed to be robust only if the ratio of the distances to the first and second nearest neighbors is below certain threshold θ. Let $NN_I^{(2)}$ be the function for second nearest neighbor search for image I:

$$NN_I^{(2)}(\boldsymbol{q}) = \underset{\boldsymbol{p} \in I - NN_I(\boldsymbol{q})}{\arg\min} \|\boldsymbol{p} - \boldsymbol{q}\|. \tag{3.5}$$

Then the set of robust matches $\mathcal{M}$ between images I_1 and I_2 can be defined by:

$$\mathcal{M}(I_1, I_2) = \{(\boldsymbol{p}_i, \boldsymbol{q}_j) \mid \boldsymbol{p}_i \in I_1, \boldsymbol{q}_j \in I_2 \text{ s.t. } \boldsymbol{p}_i = NN_{I_1}(\boldsymbol{q}_j), \frac{\|\boldsymbol{p}_i - \boldsymbol{q}_j\|}{\|NN_{I_1}^{(2)}(\boldsymbol{q}_j) - \boldsymbol{q}_j\|} < \theta\}, \tag{3.6}$$

where θ is set to 0.6 as suggested by Lowe. Notice that we are not constrained to use one image as the source for nearest neighbor search and the other for querying. By switching the roles of the two images in nearest neighbor search, we can obtain another set of feature matches $\mathcal{M}'$ between images I_1 and I_2:

$$\mathcal{M}'(I_1, I_2) = \{(\boldsymbol{p}_i, \boldsymbol{q}_j) \mid \boldsymbol{p}_i \in I_1, \boldsymbol{q}_j \in I_2 \text{ s.t. } \boldsymbol{q}_j = NN_{I_2}(\boldsymbol{p}_i), \frac{\|\boldsymbol{q}_j - \boldsymbol{p}_i\|}{\|NN_{I_2}^{(2)}(\boldsymbol{p}_i) - \boldsymbol{p}_i\|} < \theta\}. \tag{3.7}$$

One can compute both $\mathcal{M}(I_1, I_2)$ and $\mathcal{M}'(I_1, I_2)$ and use the union of the two for a larger set of feature matches. However, such a strategy is rarely adopted in practice. First of all, due to the high distinctiveness of SIFT feature descriptors, $\mathcal{M}$ and $\mathcal{M}'$ are unlikely to differ by much. Therefore the union of the two sets only leads to marginal improvement in the total number of matches. More importantly, feature matching

(nearest neighbor search in particular) is very time-consuming. Conducting nearest neighbor search in both directions between two images essentially doubles the time complexity for image matching, and may render the workload prohibitively expensive for large-scale applications. Actually, the issue of efficiency is a central topic for the rest of this section, and will be revisited in Chapter 7 in the context of large-scale applications.

Naively, one can implement the function for nearest neighbor search NN_{I_1} in Equation 3.3 by a linear scan through all the feature descriptors of image I_1, resulting in m operations of distance computation. In order to compute $\mathcal{M}(I_1, I_2)$ in Equation 3.6, nearest neighbor search needs to be initiated for each query feature descriptor of image I_2. Therefore the time complexity for computing $\mathcal{M}(I_1, I_2)$ is in the order of $\mathcal{O}(mn)$, where m and n are the number of feature descriptors of images I_1 and I_2 respectively. Given the volume and dimensionality of feature descriptors, the naive approach may take up to about ten seconds to match two regular-sized images.

In the literature of local feature matching, tree structures have been widely used to improve the efficiency for nearest neighbor search. The fundamental idea is to organize the feature descriptors from one image into a tree structure, thereby repetitive nearest neighbor queries from the other image can each finish in sub-linear time by querying the tree structure. Due to the large quantity of nearest neighbor queries, the time saved by each query adds up to a significant reduction in the overall processing time, compared to which the little overhead caused by the initial tree construction is ignorable.

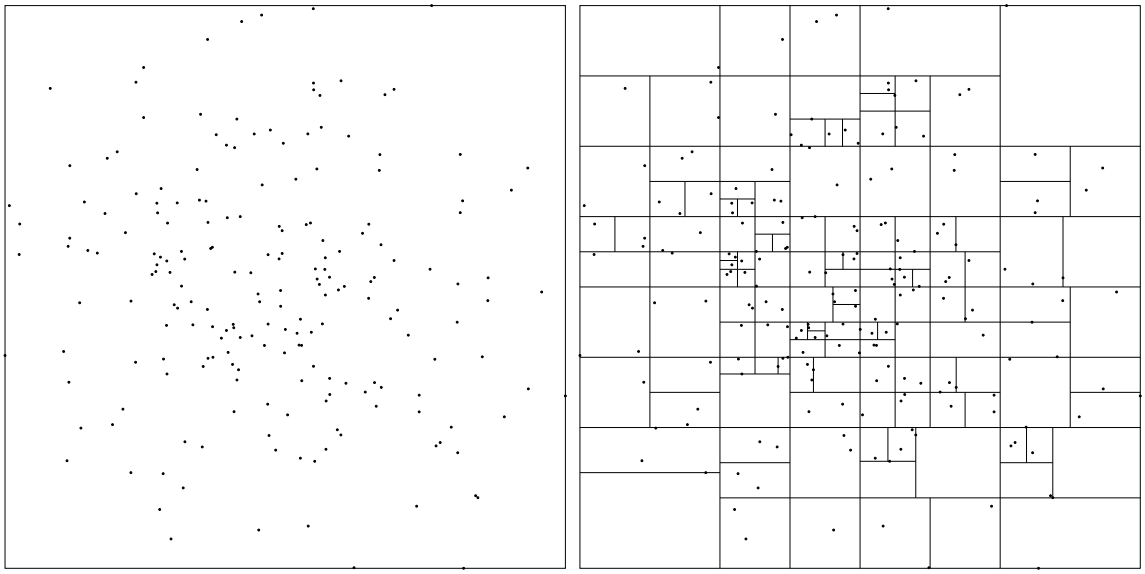


Figure 3.4: A demonstration of kd-tree in the 2-dimensional space. The left image shows 200 points randomly distributed in the 2-dimensional space. The right image shows the constructed kd-tree over the space. Each line corresponds to a split during the recursive construction of the kd-tree. The unsplit boxes correspond to the leaf buckets of the kd-tree. In this demonstration, we set $\phi = 3$. Therefore each leaf bucket contains ≤ 2 points. The nearest neighbor for any query point can be found in the leaf bucket which the query point falls or in adjacent leaf buckets.

In this dissertation, we rely on kd-tree for local feature matching. Kd-tree was invented as a general data structure for nearest neighbor search by Bentley [23] and adapted to local feature matching by Beis and Lowe [22]. The construction of a kd-tree is a recursive process. Given a set of data points in the k-dimensional space, a kd-tree recursively splits the space into two disjoint subspaces until the number of data points in each subspace falls below a predefined threshold ϕ (termination condition). At each recursion, let the set of data points in the current subspace be denoted by $P = \{\boldsymbol{p}_1, \cdots, \boldsymbol{p}_m\}$ where $\boldsymbol{p}_i$ is a k-dimensional vector. A split for the current subspace is specified by a partition axis $j \in \{1, \cdots, k\}$ and a cutoff value v

along this axis. Thereby P can be split into two subsets $P_{LE} = \{\boldsymbol{p}_i \in P \text{ s.t. } \boldsymbol{p}_{ij} \leq v\}$ and $P_{GT} = \{\boldsymbol{p}_i \in P \text{ s.t. } \boldsymbol{p}_{ij} > v\}$. Bentley proposes to select the coordinate axis along which the data points exhibit the largest variance, and sets the cutoff values at the median of the projections of all the data points. In this way, the data points can be split into two equal subsets and the variance within each subset can be minimized. After the split, the process repeats for each of the two subsets until the termination condition is met. The recursion yields a balanced binary tree, where the root of the tree corresponds to the entire k-dimensional space, and each leaf node of the tree corresponds to a subspace with fewer than ϕ data points. Data points are stored in their containing leaf nodes. Therefore a leaf node is also called a *leaf bucket*. At the root and each internal node of the tree, the partition axis and the cutoff value that form the split are recorded. Given m data points in the 128-dimensional space, the time complexity for kd-tree construction is $\mathcal{O}(m\log(m))$ [39]. The algorithm for kd-tree construction is provided in Algorithm 2. A demonstration of kd-tree in the 2-dimensional space can be found in Figure 3.4.

Once a kd-tree is constructed, the leaf buckets of the kd-tree form a complete partition of the k-dimensional space and the input data points P. Given any query data point $\boldsymbol{q}$, its nearest neighbor in P can be found in the leaf bucket which $\boldsymbol{q}$ falls in (let it be denoted by $B_{\boldsymbol{q}}$) or in leaf buckets adjacent to $B_{\boldsymbol{q}}$. This effectively rules out a majority of the data points in P. Since we have recorded the partition axis j and cutoff value v at each internal node during kd-tree construction, we can easily push $\boldsymbol{q}$ down the tree to its containing leaf bucket $B_{\boldsymbol{q}}$ by comparing $\boldsymbol{q}_j$ to v at each internal node. The data points in $B_{\boldsymbol{q}}$ serve as good candidates for the

Algorithm 2 Build kd-tree on data points P and return the root.

```
procedure BuildKdTree(P)
    if |P| < φ then
        v ⇐ new LeafBucket
        v.points ⇐ P
        return v
    else
        axis ⇐ Select a partition axis along which P exhibits the largest variance
        threshold ⇐ Compute the median projection of P on axis
        P_LE, P_GT ⇐ Split P by threshold
        v ⇐ new InternalNode
        v.axis ⇐ axis
        v.threshold ⇐ threshold
        v.leftChild ⇐ BuildKdTree(P_LE)
        v.rightChild ⇐ BuildKdTree(P_GT)
        return v
    end if
end procedure
```

nearest neighbor of $\boldsymbol{q}$. However, adjacent leaf buckets of $B_{\boldsymbol{q}}$ also need to be visited to ensure that the best candidate is the true nearest neighbor of $\boldsymbol{q}$. Friedman *et al.* suggest visiting adjacent leaf buckets by backtracking [39]. However, this strategy is suboptimal because the order in which adjacent leaf buckets are visited is solely dependent on the tree structure. The location of $\boldsymbol{q}$ is not taken into account. To this end, Arya and Mount propose *prioritized search*, in which adjacent leaf buckets are visited in ascending order of distance to $\boldsymbol{q}$ [19]. This is achieved with a little overhead of maintaining a priority queue that holds all the untaken branches during the tree traversal: when $\boldsymbol{q}$ is pushed down the tree, and a decision is made at an internal node to branch in one direction, the untaken branch, along with its distance to $\boldsymbol{q}$, is pushed into the priority queue. When the current traversal reaches the leaf bucket, the priority queue stores all the untaken branches along the way. The top entry in the priority queue corresponds to the branch that contains the next closest

leaf bucket, which provides the starting point for the next traversal. This process repeats until all adjacent leaf buckets to $B_{\boldsymbol{q}}$ are exhausted, or the distance between $\boldsymbol{q}$ and the next closest leaf bucket is already larger than that between $\boldsymbol{q}$ and the current best candidate. Upon termination, the current best candidate is guaranteed to be the true nearest neighbor of $\boldsymbol{q}$ within P.

The performance of kd-tree (with backtracking instead of prioritized search) is examined by Friedman *et al.* on low-dimensional data ($k \leq 6$) [39]. The expected time for nearest neighbor search is reported to be proportional to $\log(m)$, where m is the number of data points indexed by the kd-tree. This is a significant speedup over linear search. Unfortunately, the performance of kd-tree becomes very unstable when the dimensionality of data points grows. In a high-dimensional space, the leaf bucket that contains the query point, $B_{\boldsymbol{q}}$, has a large number of adjacent leaf buckets, and many of them must be visited to ensure true nearest neighbor. In order to adapt kd-tree to local feature matching (nearest neighbor search in the 128-dimensional space), Beis and Lowe propose the Best Bin First (BBF) algorithm, in which the prioritized search of the kd-tree is forced to terminate after a fixed number of leaf buckets are visited, and the current best candidate is returned as the *approximate nearest neighbor* for the query point. In their experiments, BBF achieves superior performance for matching SIFT features. Compared to linear search, BBF achieves a two orders of magnitude speedup while preserving more than 95% of true nearest neighbors.

In this dissertation, we obtained the implementation of BBF from the ANN library by Mount and Arya [16]. We limited the number of visited leaf buckets to 200

per query point. The average matching time per image pair is reduced from about 10 seconds using linear search to about 0.1 second using BBF.

Geometric Verification

The previous section has generated a set of feature matches between two images, where each feature match has passed the nearest neighbor and the ratio tests. However, mismatches still exist, where the matched feature points come from different real-world objects. Such mismatches are caused by the lack of discriminative power in corresponding SIFT feature descriptors. In this section, we introduce a geometric constraint which is enforced on the *locations* of matched feature points. Under the geometric constraint, many of the mismatches can be detected and removed as the locations of the corresponding feature points are geometrically inconsistent.

In this section, we ignore the feature descriptor that is associated with a feature point, and rely only on its location information. Let two images I_1 and I_2 be represented by their sets of SIFT feature locations: $I_1 = \{\boldsymbol{x}_1, \cdots, \boldsymbol{x}_m\}$ and $I_2 = \{\boldsymbol{x}'_1, \cdots, \boldsymbol{x}'_n\}$, where $\boldsymbol{x}_i = (x_i, y_i)$, $\boldsymbol{x}'_j = (x'_j, y'_j)$ encode feature locations in the image coordinate. Let their SIFT matches generated by the previous section be represented by $\{(\boldsymbol{x}, \boldsymbol{x}') \text{ s.t. } (d(\boldsymbol{x}), d(\boldsymbol{x}')) \in \mathcal{M}(I_1, I_2)\}$, where the function d returns the feature descriptor for a feature point, and $\mathcal{M}$ is the set of descriptor matches defined by Equation 3.6.

The geometric constraint between the two images is provided by the *epipolar geometry* [44]. The epipolar geometry describes the intrinsic projective geometry

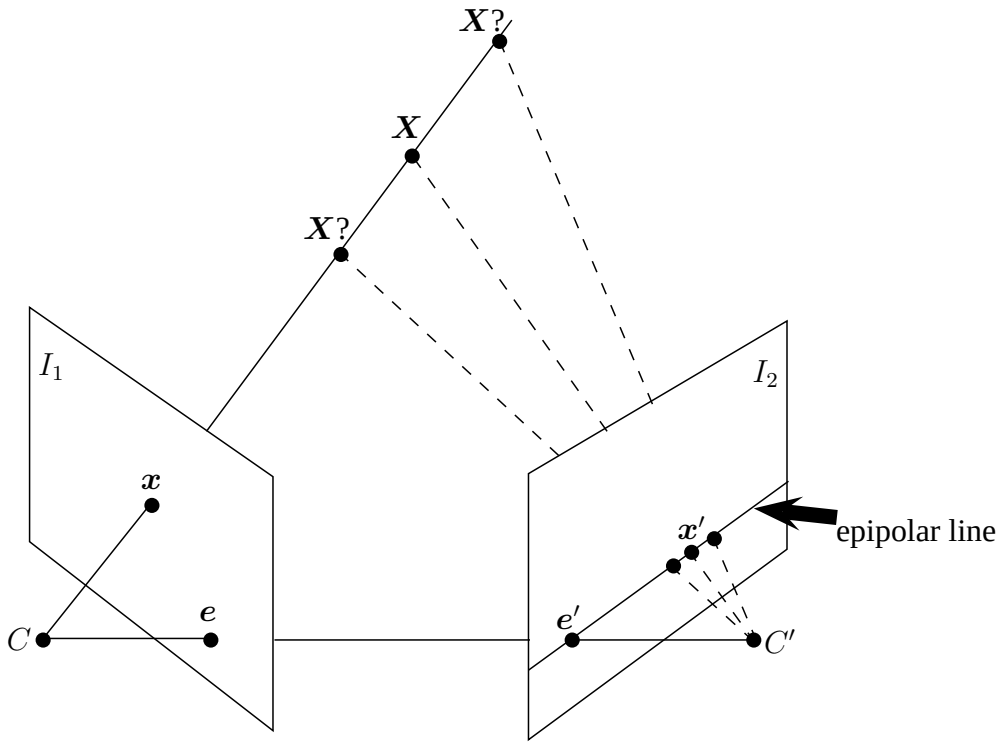


Figure 3.5: An illustration of the epipolar geometry. The two cameras behind images I_1 and I_2 are represented by their centers C and C'. Given an image point $\boldsymbol{x}$ in I_1, the corresponding real-world object point $\boldsymbol{X}$ must lie on line $C\boldsymbol{x}$. No matter where $\boldsymbol{X}$ is located along $C\boldsymbol{x}$, its projection in image I_2, $\boldsymbol{x}'$ must lie on the epipolar line, which is the projection of line $C\boldsymbol{x}$ by camera C'.

between two images of the same static object. It is derived directly from the internal parameters and the relative pose of the two cameras that capture the images. The epipolar geometry is illustrated by Figure 3.5. Under the constraint of the epipolar geometry, given an image point $\boldsymbol{x}$ in I_1, its matching point in I_2, if it exists, must lie on the *epipolar line*, which is the projection on I_2 of the 3D line formed by the first camera center and $\boldsymbol{x}$. If the claimed matching point, $\boldsymbol{x}'$, is off the epipolar line by a large amount, then the point match $(\boldsymbol{x}, \boldsymbol{x}')$ must be false, since they cannot be triangulated to the same object point in the 3D space.

In the following discussion, we assume image points $\boldsymbol{x}$ and $\boldsymbol{x}'$ are represented in homogeneous coordinates. That is:

$$\boldsymbol{x} = \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}, \qquad \boldsymbol{x}' = \begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix}. \tag{3.8}$$

The homogeneous coordinates allow 2D points and lines to have the same representation. A 2D line $\boldsymbol{l}$ defined by $ax + by + c = 0$ is represented by:

$$\boldsymbol{l} = \begin{pmatrix} a \\ b \\ c \end{pmatrix}. \tag{3.9}$$

Under homogeneous representation, a point $\boldsymbol{x} = (x, y, 1)^T$ lies on line $\boldsymbol{l} = (a, b, c)^T$ if and only if their dot product is 0: $\boldsymbol{x}^T\boldsymbol{l} = 0$. If we expand the terms in the dot product, we have exactly: $ax + by + c = 0$.

The epipolar geometry is encoded by the *fundamental matrix*, F, which is a 3×3 matrix of rank 2. F encodes the epipolar geometry in the sense that, given any point $\boldsymbol{x}$ in one image, multiplying $\boldsymbol{x}$ by F yields the epipolar line in the other image. Since its true match $\boldsymbol{x}'$ should lie on the epipolar line, a dot product between $\boldsymbol{x}'$ and the epipolar line should return 0. Therefore we have the following equation:

$$\boldsymbol{x}'^T F \boldsymbol{x} = 0. \tag{3.10}$$

It turns out that, even though the epipolar geometry is derived from camera parameters, the fundamental matrix F can be computed solely based on point matches. Therefore, given a set of point matches returned by SIFT, we can use a subset of the point matches to compute F, and then use the computed F to detect false matches in the rest of the point matches. Of course, we may have included false matches during the computation of F, in which case the computed F is not valid itself. Therefore, we need to deal with an optimization problem: given a set of point matches with a small amount of false matches in it, the goal is to find the fundamental matrix F that fits the largest subset of the point matches, and the remaining point matches are deemed to be false. In the sequel, we first describe the algorithm that solves for F based on a subset of 8 point matches, then we introduce a robust estimator to iteratively finds the fundamental matrix that fits the largest subset of point matches.

We follow the 8-point algorithm to solve for the fundamental matrix [44]. If we represent F by:

$$F = \begin{pmatrix} f_{11} & f_{12} & f_{13} \\ f_{21} & f_{22} & f_{23} \\ f_{31} & f_{32} & f_{33} \end{pmatrix}, \tag{3.11}$$

and expand Equation 3.10, we get the following dot product:

$$(x'x \;\; x'y \;\; x' \;\; y'x \;\; y'y \;\; y' \;\; x \;\; y \;\; 1)\boldsymbol{f} = 0, \tag{3.12}$$

where $\boldsymbol{f} = (f_{11}\ f_{12}\ f_{13}\ f_{21}\ f_{22}\ f_{23}\ f_{31}\ f_{32}\ f_{33})^T$ is the vectorized fundamental matrix. If we have n point matches, each of them can lead to a dot product in the same form. Therefore we can accumulate the first term of each dot product into a matrix A:

$$A = \begin{pmatrix} x'_1x_1 & x'_1y_1 & x'_1 & y'_1x_1 & y'_1y_1 & y'_1 & x_1 & y_1 & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x'_nx_n & x'_ny_n & x'_n & y'_nx_n & y'_ny_n & y'_n & x_n & y_n & 1 \end{pmatrix}, \tag{3.13}$$

and we reach the equation:

$$A\boldsymbol{f} = 0. \tag{3.14}$$

From this equation we can solve for $\boldsymbol{f}$ by computing a singular value decomposition (SVD) of A and retaining its right null-space. Notice that, in order for the solution for $\boldsymbol{f}$ to be unique, A must be at least rank 8. Therefore, this method requires a minimum of 8 point matches to compute the fundamental matrix F (theoretically, 7 point matches are the minimum requirement for computing F [44], but that method requires an extra constraint on F, and is not discussed here). Having more than 8 point matches is fine, in which case an exact solution for F may not exist, but a least-squares solution can be obtained using the same method based on SVD. Hartley and Zisserman propose several improvements over the basic 8-point algorithm, which include a preprocessing step to normalize the coordinates of image points, and an enforcement of the rank-2 property of the computed fundamental matrix. Interested readers are referred to [44] for a detailed description of the improvements.

As mentioned before, the input set of point matches contains false matches. If one or more false matches are included in the computation of F, then the computed F is not valid itself. Therefore, instead of computing one least-squares solution for F, we rely on a robust estimator to iteratively find the fundamental matrix that fits the largest subset of point matches. The robust estimator, called RANdom SAmple Consensus (RANSAC), is developed by Fischler and Bolles [35]. The idea is simple: during each RANSAC iteration, we randomly sample a minimum subset of 8 point matches, based on which we compute a candidate solution F' using the 8-point algorithm. Then we find the set of *inliers* for F', where an inlier is defined as a point match that roughly satisfies the epipolar constraint encoded by F'. In particular, given a point match $(\boldsymbol{x}, \boldsymbol{x}')$, we compute the distance d between $\boldsymbol{x}'$ and the epipolar line $F'\boldsymbol{x}$, and $(\boldsymbol{x}, \boldsymbol{x}')$ is considered to be an inlier if d is below a threshold (4 pixels in this dissertation). This process is repeated N times, by the end of which we have N candidate solutions $\{F'\}$ and N sets of inliers. Since the proportion of false matches is not high in SIFT matches, it is very likely that at least one of the candidate solutions is built on all true point matches. Out of all candidate solutions $\{F'\}$, the one that gains the largest set of inliers is retained. The corresponding set of inliers are deemed to be true point matches, and the rest (outliers) are removed. As a final robustness test, if the number of inliers is less than 16, the reconstructed fundamental matrix is considered to be unreliable and all the point matches between the two images are considered to be outliers and removed.

The number of RANSAC iterations, N, must be large enough to ensure a high probability that at least one candidate solution is built on all true point matches.

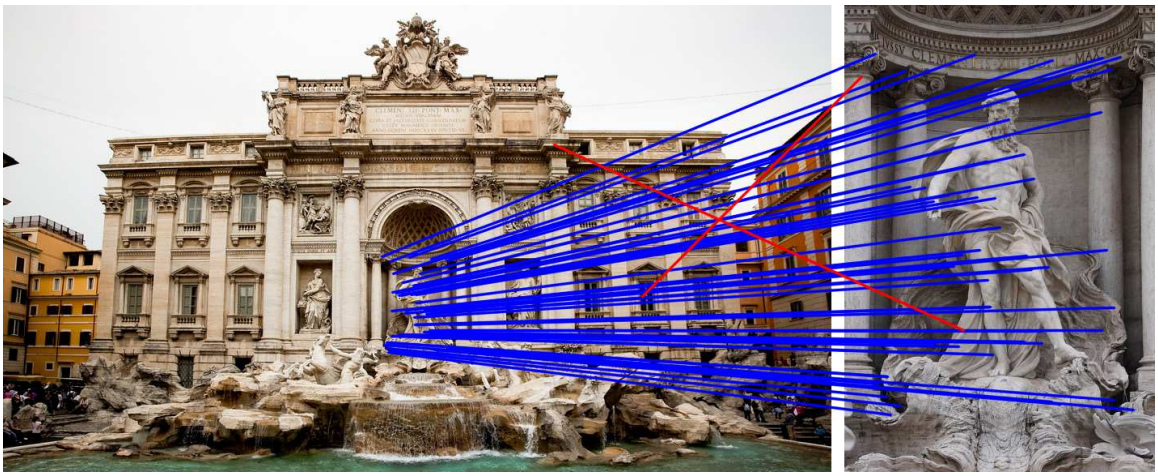


Figure 3.6: Outlier detection using RANSAC on the fundamental matrix. The false matches (red lines) are automatically detected and removed. This figure is best viewed in color.

In practice, the value of N can be derived systematically based on the precision of SIFT matches [98]. Let P_t be the probability that a SIFT match is true, and P_s be the desired probability of success after RANSAC (a success is achieved if at least one iteration samples all true point matches). During one iteration, the probability of all randomly sampled point matches being true is P_t^8. Therefore the probability of failure after RANSAC can be defined by:

$$1 - P_s = (1 - P_t^8)^N, \tag{3.15}$$

from which we can derive the minimum value of N to be:

$$N = \lceil \frac{\log(1 - P_s)}{\log(1 - P_t^8)} \rceil. \tag{3.16}$$

We tested the precision of SIFT matches on the Oxford VGG dataset [1], where ground-truth image transformation is provided among all pairs of images to verify the correctness of their SIFT matches. On average, SIFT matches achieve about 60% precision rate ($P_t = 0.6$). We would like to achieve a 99% success rate after RANSAC ($P_s = 0.99$). According to Equation 3.16, we run $N = 272$ RANSAC iterations for each pair of matching images to remove outliers. A demonstration of outlier detection between two images is shown by Figure 3.6. The remaining inlier SIFT matches, as we shall see in Chapter 4, are used to define the similarity metric for canonical view mining.

Local versus Global Features

To put local features into context, we conclude this chapter by a brief review on the alternative strategy – global features – for image matching. We conduct a comparative study of SIFT and a widely used global feature for image matching on a controlled dataset. We demonstrate the superior performance of SIFT in both precision and recall of image matches, thereby justifying our choice of SIFT feature matching for establishing the similarity metric among images.

In contrast to local features, which encode an image at selected locations, a global feature encodes the content of an image as a whole. Given an image, the simplest form of a global feature is the image itself: in [99], Torralba *et al.* leverage a dataset of 80 million tiny (32×32) color images for object and scene recognition. Each image is simply encoded by vectorizing its pixel matrix into a vector of $32 \times 32 \times 3 =$

3072 dimensions, and a basic similarity metric between two images is defined by the sum of squared distances (SSD) between the two vectors. This simple form of global feature can only encode tiny images. A large image must be downscaled before vectorization to avoid a feature vector in extremely high dimensions. A different strategy is to vectorize the image at its original resolution, and then apply dimension reduction as post-processing to the high-dimensional feature vector. This strategy is adopted by Turk and Pentland [101] for face recognition, and extended by Murase and Nayar [77] for general object recognition. In both papers, images are vectorized at their original resolutions and projected to a low-dimensional eigenspace, which is trained offline on a large dataset of face/object images using principal component analysis (PCA) [57].

Another class of global features involves quantizing pixel-level features and aggregating a histogram over all the pixels in the image. The widely used color histograms and variations are typical examples in this class [97]. Recently, Oliva and Torralba propose the *Gist* representation of an image [80]. A Gist feature encodes the structure of a photographed scene by computing the oriented edge energy at each pixel and aggregating the energies at multiple scales into coarse spatial bins. The representation of Gist is very similar to that of a SIFT descriptor. Therefore this approach can be regarded as using a single SIFT feature to encode the entire image. In practice, a Gist feature is often augmented with color information (a *Gist-color* feature) to compensate for the fact that Gist operates on the illuminance channel alone and ignores the color information in the image. In the literature, Gist/Gist-color features have been adopted and evaluated in a number of tasks. In [80], Oliva

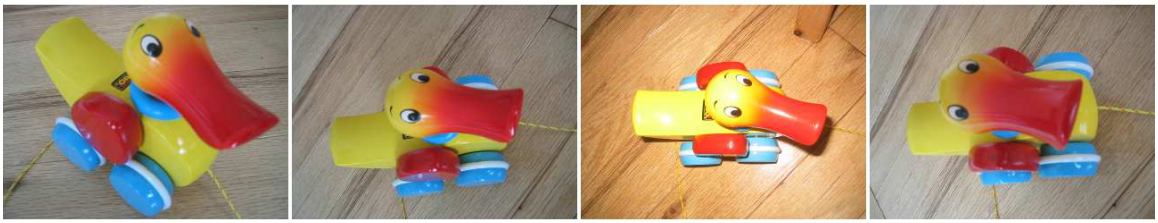


Figure 3.7: A sample object and four views in the UK Recognition Benchmark.

and Torralba use the Gist feature to classify images into semantically similar scene categories (typical scene categories include buildings, street, coast, open country). In [45], Hays and Efros use the Gist-color feature to retrieve semantically similar scenes to a query image from a database of over two million images. In [37], Frahm *et al.* use the Gist-color feature for approximate image matching in a scalable structure from motion pipeline: they use the Gist-color feature as a cheap means to segment an image collection into clusters of similar scenes; the more accurate, but also more expensive, matching using SIFT features is constrained within each cluster of images.

In order to better understand the performance of local and global features, we conduct a comparative study of SIFT and Gist-color features for image matching on a controlled dataset. We collected the dataset from the UK Recognition Benchmark [17], which consists of 10200 images of 2550 objects. Each object is captured from four distinct viewpoints (see Figure 3.7). The ground truth of image matches consists of all pairs of images of the same object. In this study, we randomly sampled 120 objects (480 images) to conduct pairwise image matching using SIFT and Gist-color features respectively.

First of all, we follow the procedure described in the previous sections with identical parameter settings to conduct pairwise image matching using SIFT features: we extract SIFT features from each image, match SIFT features among all pairs of images using BBF, verify matched image pairs by RANSAC on the fundamental matrix, and finally retain all the matched image pairs with at least 16 inlier SIFT matches. Secondly, we follow the parameter settings suggested by [37] to conduct pairwise image matching using Gist-color features: for each image, we compute the Gist feature over three scales (from coarse to fine) with 8, 8 and 4 orientations aggregated into 4×4 spatial bins. We augment each Gist feature with a subsampled and vectorized RGB image at 4×4 spatial resolution. Finally, we separately normalize the Gist part and the color part of the feature vector to unit L^1 norm. Two images are deemed to be a match if the L^1 distance between the corresponding Gist-color features is below a predefined threshold.

For each procedure of pairwise image matching, we evaluate its performance in terms of the precision and recall of image matches. Loosely speaking, precision measures how many image matches detected by the procedure are correct according to the ground truth, and recall measures how many image matches in the ground truth are detected by the procedure. Mathematical definitions are provided below:

$$precision = \frac{\sharp \text{ correct matches by procedure}}{\sharp \text{ matches by procedure}}, \tag{3.17}$$

$$recall = \frac{\sharp \text{ correct matches by procedure}}{\sharp \text{ matches by ground truth}}. \tag{3.18}$$

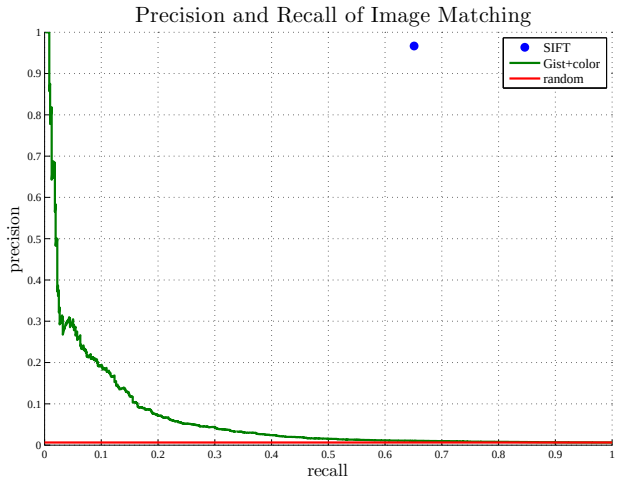


Figure 3.8: Precision and recall of image matching. The performance is compared among SIFT, Gist-color, and the random baseline. SIFT features outperform Gist-color features by a large margin.

The precision and recall achieved by SIFT and Gist-color features are shown in Figure 3.8. For Gist-color features, since we are not aware of a well-established distance threshold in the literature, we vary the distance threshold from the minimum to the maximum of the pairwise distances among all images, producing a complete precision-recall curve. It is easy to observe that SIFT features outperform Gist-color features by a large margin. SIFT features achieve a recall of image matches at 65.13%, while maintaining a near-perfect precision at 96.67%. On the other hand, Gist-color features only achieve a high precision when the distance threshold is set to be extremely low. As the distance threshold is increased for a higher recall, the precision of image matches drops sharply. At the same recall achieved by SIFT features (65.13%), the precision of Gist-color features is almost equivalent to the baseline where image matches are selected at random.

A similar observation of Gist-color features is reported by Hays and Efros – "scenes can only be trusted to be semantically similar if the distance between them is very small" [45]. In an image retrieval application such as [45], the low recall of image matches can be compensated by leveraging a huge database of images, so that regardless of the relatively low recall of image matches, the absolute number of image matches is still satisfactory. In a canonical view mining application, however, the low recall of image matches renders the similarity graph too sparse to reveal the wisdom of crowds on which views are canonical. Indeed, in the early experiments on canonical view mining, we tried replacing SIFT features by Gist-color features for the similarity metric among images, and the resulting canonical views are near-random.

CHAPTER 4

CANONICAL VIEWS: ALGORITHM DESCRIPTION

In this chapter, we describe the algorithm for canonical view mining. The problem is formalized as follows: given a collection of n images $\mathcal{P} = \{P_1, P_2, \cdots, P_n\}$, let $\mathcal{C}_k \subset \mathcal{P}$ be the subset of images that best summarizes $\mathcal{P}$ out of all subsets of size k. The goal is to compute a permutation of images $\mathcal{P}^* = \pi(\mathcal{P})$, such that the k-prefix $\mathcal{P}^*_{1,\cdots,k} \approx \mathcal{C}_k$ for $1 \leq k \leq n$. In other words, we would like to compute an ordering of the images, such that the top k images approximate the k-set of canonical views for the image collection. Once the permutation $\pi(\mathcal{P})$ is computed, canonical views of any size k can be retrieved for any subset $\mathcal{Q} \subseteq \mathcal{P}$ in real-time by taking the first k images in $\pi(\mathcal{P})$ that also appear in $\mathcal{Q}$.

For the purpose of illustration, this chapter shows step-by-step results and analysis for the algorithm on one of the experimental image collections. Experiments on other image collections can be found in Chapter 5. The image collection in question consists of 11959 images of Rome from 2493 photographers collected from Flickr (henceforth referred as the `Rome` collection).

Images are encoded and matched by SIFT features. The similarity between two images is measured by the number of inlier SIFT matches normalized by the total

number of SIFT features in both images:

$$similarity(P_i, P_j) = \frac{2|matches(P_i, P_j)|}{|features(P_i)| + |features(P_j)|}, \tag{4.1}$$

where $P_i \neq P_j \in \mathcal{P}$. The similarity values range from 0 to 1, where 0 indicates two images having no inlier SIFT matches, and 1 indicates two images having the same set of SIFT features (and therefore fully matched). See Chapter 3 for a discussion of SIFT feature matching. With the similarity metric defined, a similarity graph can be formed over the image collection, with vertices representing images and weighted edges indicating the similarity between images.

The ranking of canonical views is found in two phases. During the first phase, images are ranked by the representativeness of their photographed scene, which is measured by eigenvector centrality and solved using the power method [42]. During the second phase, a reranking scheme is applied to images to demote redundant views. The reranking scheme, which is based on adaptive non-maximal suppression [27], forces top-ranked images to be not only representative, but also diverse, and therefore serves as the ranking of canonical views.

Ranking Representative Views

Relying on the wisdom of crowds [96], we interpret a representative view as an image whose photographed scene is shared by many other images. In the similarity graph, such images are characterized by vertices that connect to many neighbors with high-weight edges. In graph theory, a class of metrics exists that quantitatively

measures the *centrality* of vertices [78], which coincides with our characterization of representative views. For example, the simplest metric of centrality, the *degree centrality*, measures the importance of a vertex by the number of incident edges. In a weighted graph, degree centrality can be generalized to the sum of weights carried by the incident edges to a vertex, which is very similar to our characterization of representative views in a similarity graph. In practice, however, this simple metric has a strong bias toward productive photographers. Imagine a photographer taking hundreds of images of the same random view. These images are connected by high-weight edges to each other and form a clique in the similarity graph. By the standard of degree centrality, they are all representative, even though none of them are.

We adopt the *eigenvector centrality* to measure representativeness for images. The principle of eigenvector centrality considers not only the number of neighbors of a vertex, but also the importance of the neighbors. A vertex close to important vertices may be valued higher than one close to unimportant vertices, even though the latter has a higher neighbor count. This is in contrast to degree centrality where all vertices are treated as equivalent. If we encode the similarity graph by adjacency matrix A, where A_{ij} stores the similarity between vertices v_i and v_j, and let x_i denote the eigenvector centrality for vertex v_i, then x_i is defined as follows:

$$x_i = \frac{1}{\lambda} \sum_j A_{ij} x_j, \tag{4.2}$$

which is a weighted sum of the eigenvector centralities of v_i's neighbors. The normalization factor λ is to ensure $\sum_i x_i = 1$. If we organize $\{x_i\}$ in vector format

$\boldsymbol{x} = (x_1, \cdots, x_n)^T$, then we can rewrite Equation 4.2 in matrix representation:

$$\boldsymbol{x} = \frac{1}{\lambda} A\boldsymbol{x}. \tag{4.3}$$

Rearranging the terms, we have the standard eigenvector equation $A\boldsymbol{x} = \lambda\boldsymbol{x}$, where $\boldsymbol{x}$ is an eigenvector for the adjacency matrix A, and the normalization factor λ is the corresponding eigenvalue (hence the name eigenvector centrality).

Eigenvector centrality has found a wide range of applications in network analysis. The most well-known of all is probably the PageRank algorithm, which is adopted by Google for ranking authority for webpages [25]. In the PageRank algorithm, the web is encoded by a directed graph where each edge is a hyperlink directed from one webpage to another. Intuitively, webpage authors tend to direct links to other webpages that are both relevant and authoritative. Therefore, a link can be treated as a vote from one webpage to another for the latter's authority. However, not all links carry equal weight. For example, a link directed from the homepage of Microsoft should be considered a much stronger evidence for authority than a link directed from a personal blog. Hence the PageRank algorithm weights each vote by the authority of the source and iteratively propagates weighted votes among the network and accordingly updates the authorities of webpages, which essentially converge to eigenvector centralities.

Inspired by the success of PageRank, Jing and Baluja propose to apply the algorithm to large-scale image search [53]. Their algorithm, termed VisualRank, provides the basis for our algorithm for ranking representative views. Unlike the web,

images are not naturally connected to form a network. When applying PageRank to the image domain (or VisualRank), hyperlinks between webpages are replaced by analogous "visual connections" between images, which, in our experiments, are specified by the similarity metric in Equation 4.1.

Once the similarity graph is formed, the ranking of representative views reduces to the solution for an eigenvector for the underlying adjacency matrix. In general, up to n eigenvector/eigenvalue pairs exist for an $n \times n$ adjacency matrix. However, by the Perron-Frobenius Theorem [85], if we constrain all eigenvector centralities to positive values, which is a desirable property for image representativeness, then only the dominant eigenvector can satisfy the requirement. Therefore, the problem is simplified to the solution for one eigenvector/eigenvalue pair for the adjacency matrix – the dominant one. In this case, we can avoid a full decomposition of the adjacency matrix, which is extremely inefficient when n is large, by using the power method [42].

The power method is an iterative algorithm. Following the same denotation as Equation 4.3, the power method initializes the solution for $\boldsymbol{x}$ to a random vector of unit L1-norm, and iteratively multiplies $\boldsymbol{x}$ by A and normalizes $\boldsymbol{x}$ back to unit L1-norm. Specifically, in the t^{th} iteration:

1. $\boldsymbol{x}^{(t+1)} = A\boldsymbol{x}^{(t)}$,

2. $\boldsymbol{x}^{(t+1)} = \frac{\boldsymbol{x}^{(t+1)}}{\|\boldsymbol{x}^{(t+1)}\|_1}$.

$\boldsymbol{x}^{(0)}$ is usually initialized to $(\frac{1}{n}, \cdots, \frac{1}{n})^T$, and $\boldsymbol{x}^{(\infty)}$ is the solution for $\boldsymbol{x}$. In practice, when the adjacency matrix A contains no negative entries, the second normalization

step can be eliminated by preprocessing A to form a stochastic matrix A^*, in which columns are normalized to unit L1-norm. By doing so, the L1-norm of $\boldsymbol{x}^{(t)}$ is never affected by multiplying by A. Now each iteration consists of a single matrix-vector multiplication:

1. $\boldsymbol{x}^{(t+1)} = A^*\boldsymbol{x}^{(t)}$.

In order for the power method to converge, the stochastic matrix A^* must be irreducible [66]. This is equivalent to the underlying similarity graph being connected. In practice, connectivity is forced by introducing a *damping factor* to the power iteration:

$$\boldsymbol{x}^{(t+1)} = dA^*\boldsymbol{x}^{(t)} + (1-d)(\frac{1}{n})_{n\times 1}, \tag{4.4}$$

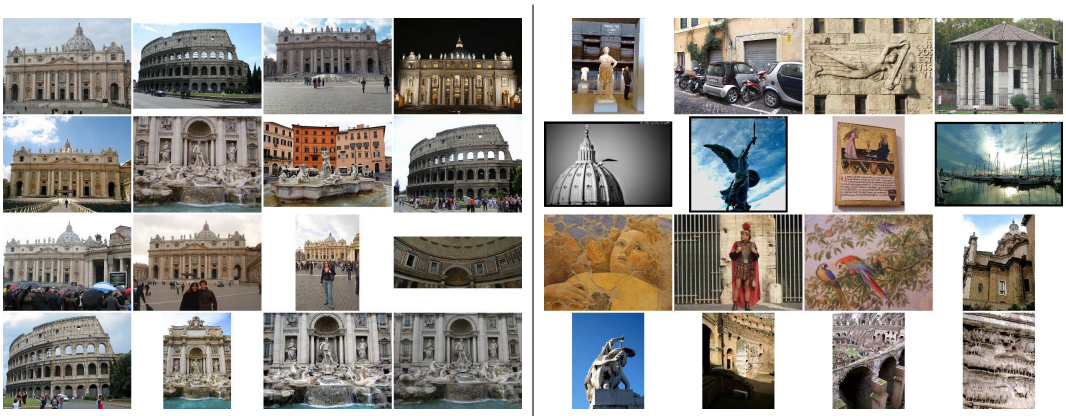
where $d \in (0,1)$ is the damping factor. Intuitively, the addition of the damping factor is equivalent to adding a low-weight edge between all pairs of vertices in the similarity graph, thereby enforcing connectivity. Previous applications of the power method have suggested a damping factor of $d = 0.85$ [25,53], which is adopted in this dissertation. In early experiments, we have tried varying d from 0.75 to 0.95 and we have observed little difference in the solution for image representativeness.

Following the power method with damping factor, the algorithm for ranking representative views is provided in Algorithm 3. The input to the algorithm is an $n \times n$ stochastic matrix A^* encoding the similarity graph over n images. The output from the algorithm is a vector $\mathbf{x}$ of n representativeness scores for n images.

The convergence rate for the power method is fast. In all of our experiments, the algorithm converges within 50 iterations. The high convergence rate, combined

Algorithm 3 Compute the representativeness scores $\mathbf{x}$ for n images, given the stochastic matrix A^*.

$\mathbf{x} \Leftarrow (\frac{1}{n})_{n\times 1}$
repeat
  $\mathbf{x}' \Leftarrow \mathbf{x}$
  $\mathbf{x} \Leftarrow dA^*\mathbf{x}' + (1-d)(\frac{1}{n})_{n\times 1}$
until $|\mathbf{x} - \mathbf{x}'| < \epsilon$



(a) Top-ranked representative views

(b) Bottom-ranked representative views

Figure 4.1: A comparison between the top-ranked and bottom-ranked representative views of the `Rome` collection. Bottom-ranked images hardly capture any recognizable views of Rome. In comparison, all of the top-ranked images indeed capture landmarks, such as the Colosseum and the Trevi Fountain. However, despite being representative, the top-ranked representative views are highly redundant. For example, in the top 16 representative views, 7 images capture St. Peter's Basilica within the top 16 images.

with simple matrix-vector multiplications in each power iteration, allows the algorithm to be highly efficient, processing more than ten thousand images in a matter of seconds. The power method is scalable, too, given that matrix-vector multiplications are often well optimized and easily parallelizable on a cluster of CPUs. Upon convergence of the power method, each image is assigned a representativeness score, by which representative views can be ranked.

In Figure 4.1, top-ranked and bottom-ranked representative views are shown for the `Rome` collection. As expected, bottom-ranked images hardly capture any rec-

ognizable views for Rome. In comparison, all of the top-ranked images indeed capture landmarks, such as the Colosseum and the Trevi Fountain.

However, despite being representative, the top-ranked representative views are highly redundant. For example, in the top 16 representative views for the `Rome` collection, 7 images capture St. Peter's Basilica. The high volume of redundancy is understandable: if the view of St. Peter's Basilica gains more representativeness during power iteration, then by definition of eigenvector centrality, all nearby vertices in the similarity graph (images sharing overlapping views) are likely to be assigned high representativeness scores. The redundancy issue is addressed in the next phase.

Ranking Canonical Views

In the similarity graph, redundant views can be characterized by vertices connected by high-weight edges. Unfortunately, the metric of eigenvector centrality tends to assign such vertices similar representativeness scores, resulting in a high volume of redundancy in top-ranked representative views. During the second phase, images are reranked by a scheme that demotes redundant views. The reranking scheme is inspired by adaptive non-maximal suppression [27]. We let a representative image suppress visually similar but less-representative images, and rerank images by the number of images that they suppress. Out of all images of the same redundant view, only the most representative image remains top-ranked and the rest are demoted.

Encoded by visual features, images can be considered as data points in a high-dimensional feature space. Each data point is associated with a representativeness

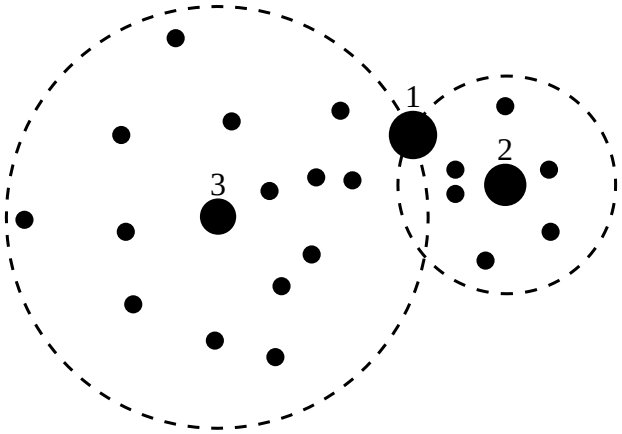


Figure 4.2: Demoting redundant views by adaptive non-maximal suppression (an illustration). Images are represented by solid dots. Size of dots indicates representativeness of corresponding images. The ranking of images is therefore (1, 2, 3, ...). Under the metric of visual dominance, each image has a suppression radius in the feature space determined by the minimum distance to a more representative image. Each image is scored by counting images that fall in its suppression radius. In this example, image 2 captures a redundant view of image 1 (close-by in feature space). Therefore image 2 has a small suppression radius and low count of images within its suppression radius. After adaptive non-maximal suppression, the ranking of images becomes (1, 3, 2, ...). Image 2 is demoted.

score computed during the previous phase. We define the *suppression radius* r_i for image $P_i \in \mathcal{P}$ as the minimum distance from P_i to a more representative image in $\mathcal{P}$:

$$r_i = \min_j distance(P_i, P_j), \text{ s.t. } x_i < x_j, P_j \in \mathcal{P}, \tag{4.5}$$

where x_i and x_j are representativeness scores for images P_i and P_j respectively. Given the similarity metric in Equation 4.1 and the range of similarity values between 0 and 1, the distance metric between images can be naturally defined as follows:

$$distance(P_i, P_j) = 1 - similarity(P_i, P_j). \tag{4.6}$$

The suppression radius for an image specifies a spherical neighborhood in the feature space in which the image is the most representative. According to the definition, for any pair of images $P_i \neq P_j \in \mathcal{P}$, the suppression radius for the less-representative image is bounded by $distance(P_i, P_j)$. If P_i and P_j capture redundant views ($distance(P_i, P_j)$ is small), then the less-representative image would be demoted for having a small suppression radius. However, reranking images solely based on suppression radius would be unwise, because images of random views, with which no other images share, will also have large suppression radiuses according to the definition. Therefore, we propose to rerank images by the number of images that fall in the suppression radius: $|\{P_j \text{ s.t. } distance(P_i, P_j) < r_i, P_j \in \mathcal{P}\}|$. In this way, top-ranked images are still representative, because they suppress many visually similar images in the feature space, but redundant views are demoted. Top-ranked images are not only representative, but also diverse, which fulfills our requirements for canonical views. The process of demoting redundant views by adaptive non-maximal suppression is illustrated by Figure 4.2. Sample images in the `Rome` collection that are suppressed during adaptive non-maximal suppression are shown in Figure 4.3.

A comparison between representative views and canonical views is illustrated in Figure 4.4. Representative views and canonical views are visualized in the similarity graph, which covers part of the `Rome` collection consisting of 1558 vertices (images) and 14198 edges (matching pairs). The lengths of edges are inversely proportional to their weights. Therefores close-by vertices are likely to share redundant views. A number of clusters can be clearly observed. Each cluster corresponds to images of a frequently photographed scene. Vertices corresponding to the top 16 representative

views and canonical views are visualized as red dots in the graph. It can be observed that the representative views reside in only a few clusters, and many of them are very close to each other (most of them are indeed redundant views as illustrated by Figure 4.1). On the other hand, the canonical views reside in all significant clusters with little redundancy (close-by vertices), providing a much more complete summary for all images. In Figure 4.5, top-ranked canonical views, in comparison with top-ranked representative views, are shown for the `Rome` collection. With redundant views demoted, the canonical views are both representative and diverse, covering within 16 images a much larger number of landmarks of Rome (see the Trevi Fountain, Colosseum – exterior and interior, St. Peter's Basilica – exterior and interior, Pantheon – exterior and interior). Compared to the first page of search results of Rome on Flickr (see Figure 1.1), the canonical views convey a much clearer impression of Rome within a smaller number of images.

The algorithm for adaptive non-maximal suppression and canonical view ranking is provided in Algorithm 4. In practice, having ranked images by descending order of representativeness, an image only needs to be compared to higher-ranked images to compute its suppression radius, and lower-ranked ones to count the number of images that it suppresses. Therefore the time complexity for the algorithm is in the order of $\mathcal{O}(n^2)$, where n is the number of images. In our experiments, ranking canonical views for more than ten thousand images finishes in a matter of seconds.

The ranking of canonical views has a number of applications. In the next section, we demonstrate an application of canonical views for large-scale image browsing.

Algorithm 4 Compute the ranking of canonical views $\mathcal{C}$, given an image collection $\mathcal{P}$ with representativeness scores $\mathbf{x}$.

```
P* ⇐ sort P in descending order of x
c ⇐ (0)_{n×1}
for i ⇐ 1 ··· n do
    r ⇐ ∞
    for j ⇐ i − 1 ··· 1 do
        if distance(P*_i, P*_j) < r then
            r ⇐ distance(P*_i, P*_j)
        end if
    end for
    for j ⇐ i + 1 ··· n do
        if distance(P*_i, P*_j) < r then
            c_i ⇐ c_i + 1
        end if
    end for
end for
C ⇐ sort P* in descending order of c
```

In Chapter 6, we discuss another application of canonical views for efficient object recognition.

Image Browsing with Canonical Views

The ranking of canonical views can assist in browsing large image collections. In the simplest form, one can present the top-ranked k canonical views as a succinct visual summary for a large image collection. Having removed noise and redundancy, the canonical views can facilitate an understanding of the essence of the large number of images without the need for extensive browsing. More importantly, once the ranking of canonical views has been computed for the entire image collection offline, any number of canonical views for any subset of the image collection can be retrieved in real-time. The method is straightforward: following the notation from previous sections, let us denote the entire image collection by $\mathcal{P}$, the ranking of canonical views

by $\pi(\mathcal{P})$, the subset of interest by $\mathcal{Q} \subseteq \mathcal{P}$, and the desired number of canonical views by k. In order to retrieve the k canonical views for $\mathcal{Q}$, we simply scan $\pi(\mathcal{P})$ from top to bottom and return the first k images that also appear in $\mathcal{Q}$.

The ability to retrieve canonical views for any subset of images in real-time is appealing to image search engines, because any query issued by the user specifies a subset of the entire image collection as search results. Therefore the user can browse the search results by their canonical views. We demonstrate this idea in Figures 4.6 and 4.7. In this demonstration, the `Rome` collection is considered as the superset of images. Once the ranking of canonical views has been computed for the superset, a user can specify constraints on image metadata to browse the canonical views for any subset of images of interest. In both cases, the constraints are specified for the subsets of images of the Trevi Fountain, the Colosseum, the Vatican and the Pantheon. In Figure 4.6, the constraints are specified by keywords. The subset of interest is therefore formed by images whose tags match all of the keywords. In Figure 4.7, the constraints are specified by geometric bounding boxes. Similar to TagMaps [15] (see Figure 2.3), we leverage the GPS tags associated with the images and enable a user to browse canonical views for a particular geographic region by zooming in/out or translating over a map view of Rome. The subset of interest is therefore formed by images whose GPS tags fall in the geographic bounding box. In both cases, the canonical views are retrieved in real-time. Notice that the canonical views for subsets of images are also representative and diverse, capturing the landmarks of interest from different iconic viewpoints. The qualities of the canonical views are systematically evaluated in Chapter 5.

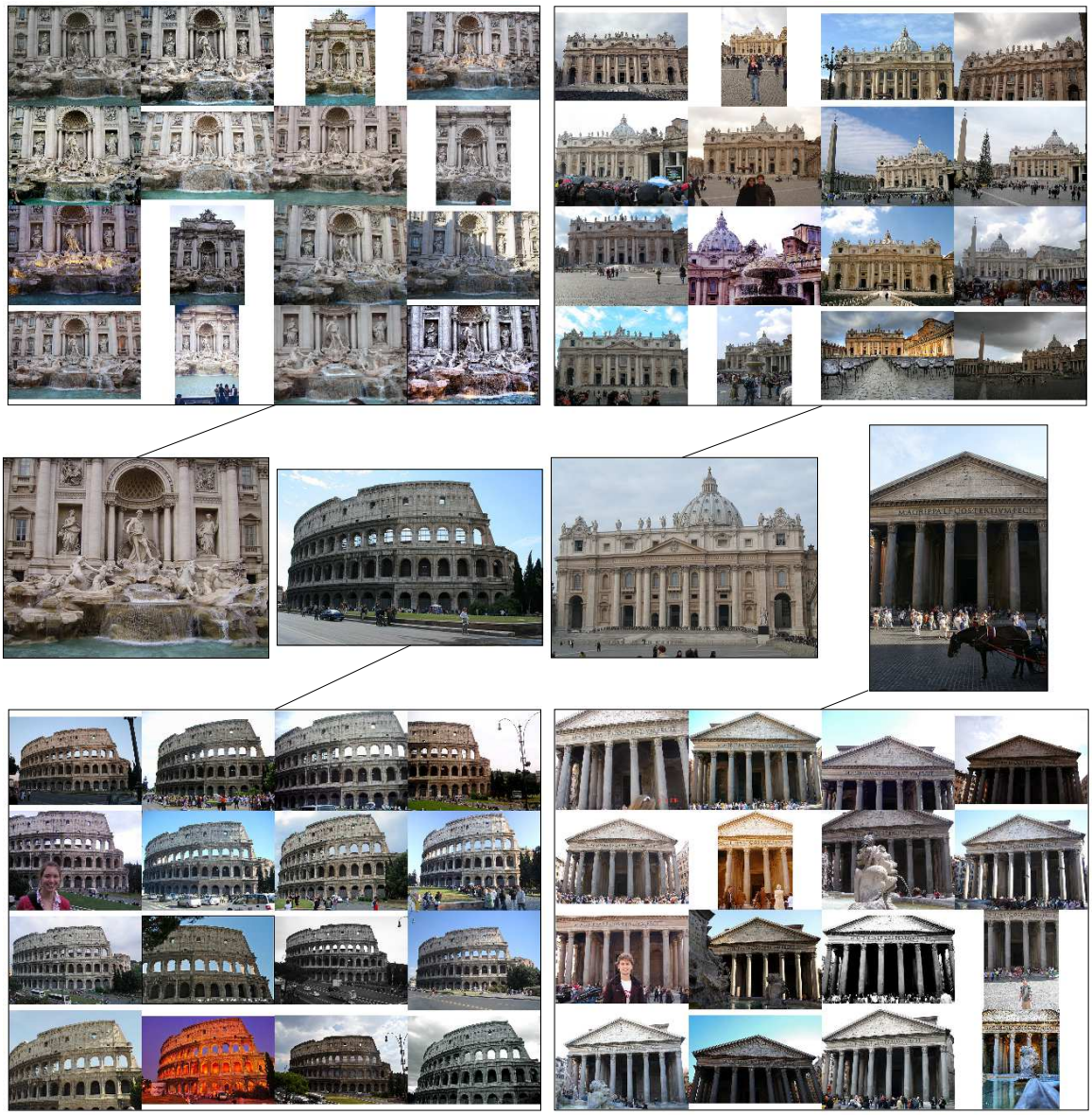
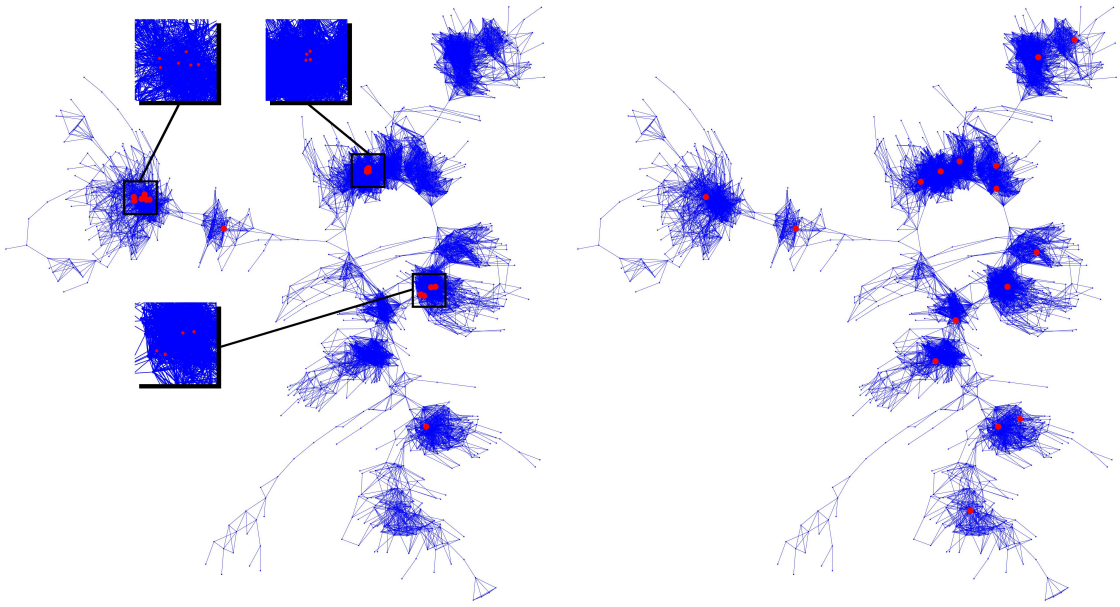


Figure 4.3: Demoting redundant views by adaptive non-maximal suppression (sample images). Sample images that are suppressed during adaptive non-maximal suppression are shown in the top and bottom rows. The corresponding "suppressing" images are shown in the central row. After adaptive non-maximal suppression, the suppressing images remain top-ranked as canonical views, while the suppressed images are demoted in the ranking. Therefore the canonical views are diversified.



(a) Representative views (b) Canonical views

Figure 4.4: A comparison between the representative views and canonical views visualized in the similarity graph. The similarity graph covers part of the `Rome` collection consisting of 1558 vertices (images) and 14198 edges (matching pairs). The lengths of edges are inversely proportional to their weights. Therefore close-by vertices are likely to share redundant views. A number of clusters can be clearly observed. Each cluster corresponds to images of a frequently photographed scene. Vertices corresponding to the top 16 representative views and canonical views are visualized as red dots in the graph. The representative views reside in only a few clusters, and many of them are very close to each other (most of them are indeed redundant views as illustrated by Figure 4.1). On the other hand, the canonical views reside in all significant clusters with little redundancy (close-by vertices), providing a much more complete summary for all images. The similarity graph is visualized by GraphViz [11]. This figure is best viewed in color.

(a) Representative views (b) Canonical views

Figure 4.5: A comparison between the top-ranked representative views and canonical views of the `Rome` collection. With redundant views demoted, the canonical views are both representative and diverse, covering within 16 images a much larger number of landmarks of Rome (see the Trevi Fountain, Colosseum – exterior and interior, St. Peter's Basilica – exterior and interior, Pantheon – exterior and interior).

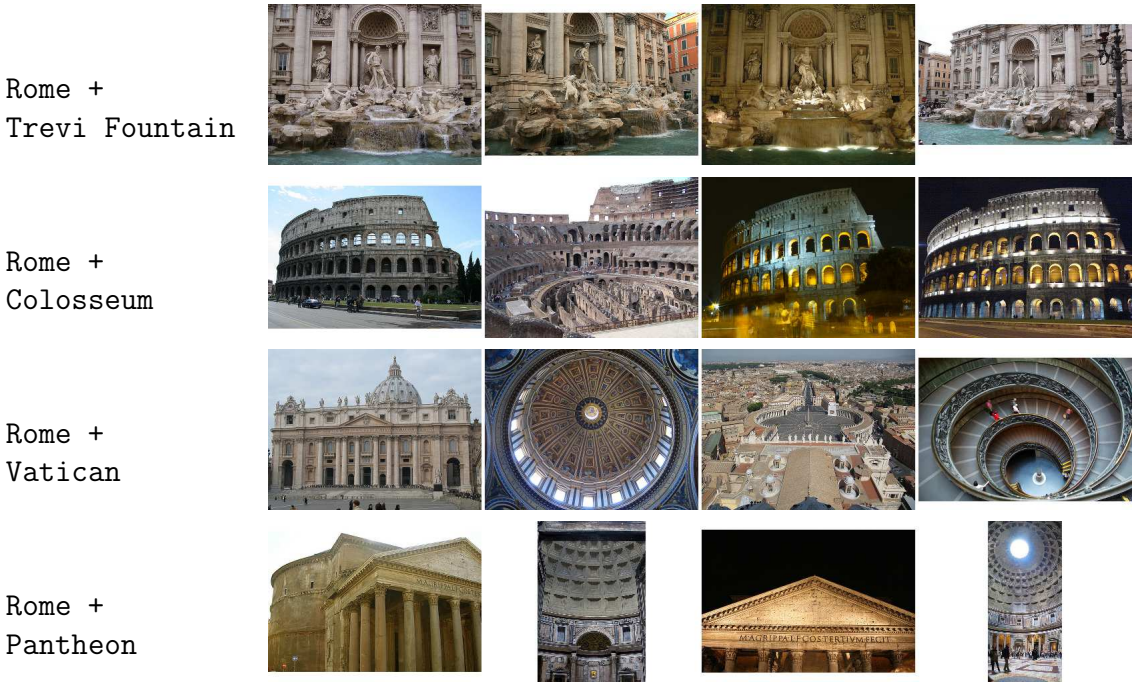


Figure 4.6: Canonical views for different subsets of the `Rome` collection specified by keywords. The left column shows the added keyword(s) to the original keyword `Rome`. Each set of keywords leads to a subset of the `Rome` collection. The right column shows the canonical views for the corresponding subsets. Canonical views for all subsets are retrieved in real-time.

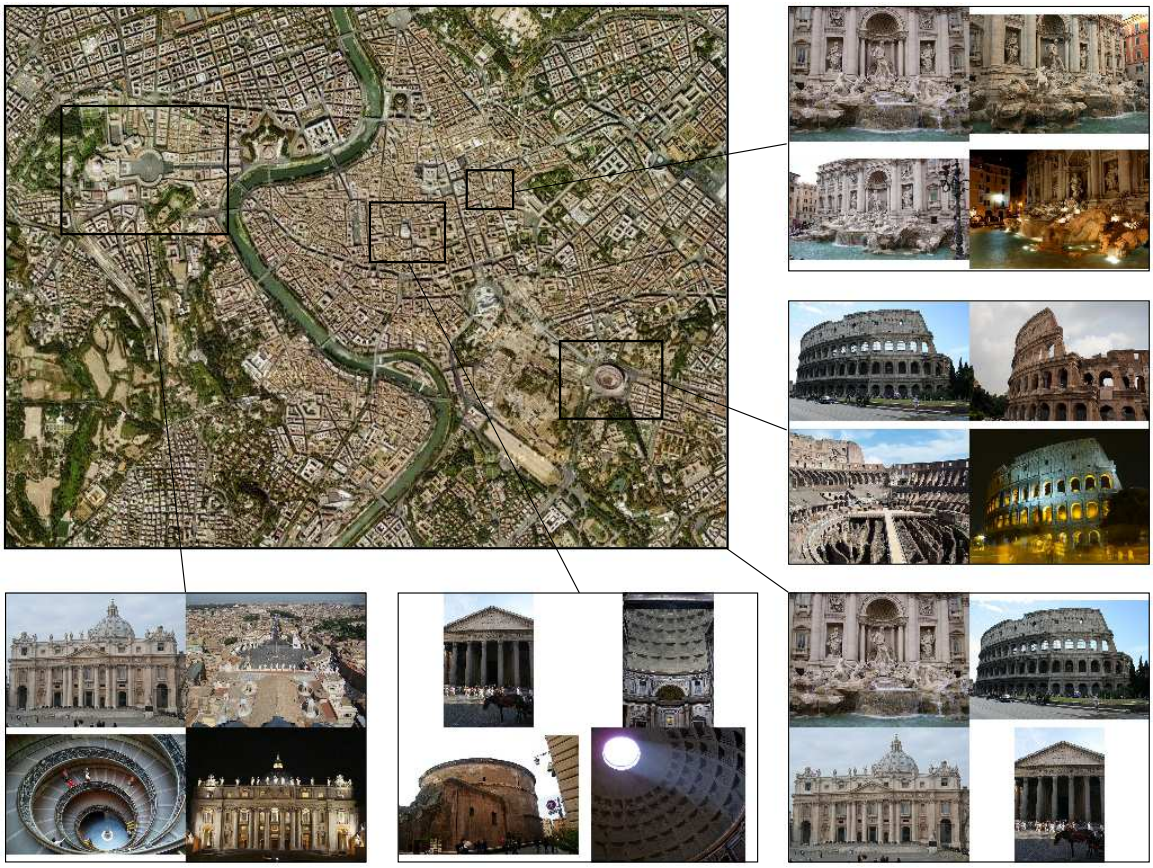


Figure 4.7: Canonical views for different subsets of the `Rome` collection specified by geographic bounding boxes. The top-left image shows a satellite image of Rome, with regions of interest specified by geographic bounding boxes. Each region leads to a subset of the `Rome` collection. The outer images show the canonical views for the corresponding subsets (from top-right clock-wise, the geographic bounding boxes cover the areas of the Trevi Fountain, the Colosseum, Rome, the Pantheon, the Vatican). Canonical views for all subsets are retrieved in real-time.

CHAPTER 5

CANONICAL VIEWS: EVALUATIONS

In this chapter, we demonstrate the quality of canonical views on various image collections. We develop quantitative measurements to evaluate the canonical views, and compare the proposed algorithm for canonical view mining to other methods, including image search engines, representative view ranking, and the previous work of Simon *et al.* [92].

Data Collection and Qualitative Results

We collected all the image collections from two sources: the photo sharing site Flickr [4], and the image search engine Google Images [9]. Both websites provide an interface for keyword-based image retrieval. We specify the following set of keywords for data collection:

- *places*: `Dubrovnik`, `Paris`, `Rome`, `Washington DC` and `Yosemite`,
- *products*: `Canon`, `iPod`, `Starbucks` and `Wii`,
- *artworks*: `da Vinci` and `Michelangelo`.

These keywords are selected based on several criteria. First of all, the volume of search results corresponding to a keyword must be large; otherwise, the images may be severely biased toward one or more productive photographers, rendering the basis

Table 5.1: Statistics of Flickr images.

Search keyword(s)	♯ images	♯ Flickr users
Places		
`Dubrovnik`	9350	1941
`Paris`	11997	3237
`Rome`	11959	2493
`Washington DC`	11991	2114
`Yosemite`	5756	1058

of canonical view mining – the wisdom of crowds – invalid. In practice, however, this criterion is the easiest to fulfill, since image search on almost any non-obscure keyword can retrieve a huge amount of search results from the above two sites. Secondly, the photographed objects corresponding to a keyword must be matchable across different images by SIFT features. This requirement is entailed by our definition of image similarity, which is based on SIFT feature matching. In general, SIFT demonstrates extremely high precision and recall for matching objects of *rigid appearances*. On the other hand, SIFT cannot match object classes such as animals and plants, whose appearances undergo nonrigid transformations (deformations) from one image to another. Nonetheless, images of rigid objects (*places* in particular) compose a significant portion of internet image collections [90], and have been the focus of much previous work on canonical view mining. Finally, the set of keywords is selected with a large overlap to previous work, so that the performance of canonical view mining can be compared among different methods on a common ground. All three object classes – *places*, *products* and *artworks* – have attracted great attention in the previous work on canonical view mining [30, 50, 54, 64, 92].

Table 5.2: Statistics of Google images.

Search keyword(s)	♯ images	♯ websites
Products		
`Canon`	672	288
`iPod`	793	395
`Starbucks`	779	482
`Wii`	669	370
Artworks		
`da Vinci`	245	162
`Michelangelo`	333	218

We collected images of *places* from Flickr, and images of *products* and *artworks* from Google Images. Flickr provides an API for massive download of images based on keywords [5]. We specified an upper limit of 12000 images per image collection (keyword search). Besides image files, we also downloaded the metadata attached to the images, including textual tags and GPS tags (geographic locations of capture). In total, we downloaded 51053 images from 10843 Flickr users. Image statistics are listed in Table 5.1. Different from photo sharing sites, Google Images indexes images not from personal photo albums, but from all the websites on the internet. It does not provide an API for massive download. We therefore parsed the search results returned by the search engine and followed the image URLs to download the images from their original websites. Google Images limits the number of search results to several hundred per query. In total, we downloaded 3491 images from 1915 websites. Image statistics are listed in Table 5.2. For stable performance of image matching, all large images in the collections are downscaled to a maximum dimension of 640 pixels.

The algorithm described in Chapter 4 was applied to each image collection to mine canonical views. Qualitative results of canonical views are shown in Figures 5.1, 5.3 and 5.4. For image collections of *places*, many of the canonical views capture popular landmarks of the places (listed in the caption of Figure 5.1). Moreover, almost all the landmarks are captured in their most iconic viewpoints. Images of near-identical viewpoints to these can be found on authoritative tourism websites such as the Wiki pages of these landmarks (see Figure 5.2). Apart from landmarks, some canonical views in the collections of `Dubrovnik` and `Paris` capture views of the cityscapes, because such views tend to connect many images in the similarity graph, and thereby gain large supports during canonical view ranking. The canonical views for *products* and *artworks* are also promising: almost all the canonical views for *products* capture either the popular products under the brand names or the brand logos; the canonical views for *artworks* capture most of the masterpieces of the two artists as listed on their Wiki pages (also listed in the caption of Figure 5.4). Some images in the collection of `da Vinci` are collages of multiple paintings. Similar to the cityscape views of `Dubrovnik` and `Paris`, collages also tend to connect many images in the similarity graph, and thereby gain large supports during canonical view ranking. Three of the four canonical views in the collection of `da Vinci` are collages of multiple paintings. In this dissertation, we do not treat collages differently from regular images, although it would not be very difficult to detect and remove collages based on the structure of the visual similarity graph and geometric constraints (intuitively, collages tend to connect many otherwise disconnected vertices in the visual similarity graph, and the camera parameters of collages cannot be uniquely resolved).

Table 5.3: Statistics of Dataset-II.

Original collection	Additional keywords	♯ images	♯ Flickr users
`Paris`	`Notre Dame`	637	337
`Rome`	`Colosseum`	604	306
`Rome`	`Pantheon`	422	237
`Rome`	`Trevi Fountain`	354	208
`Washington DC`	`Capitol`	795	362
`Washington DC`	`Lincoln`	505	256

Besides the image collections listed in Tables 5.1 and 5.2 (which are henceforth collectively referred to as Dataset-I), we also prepared a separate dataset, Dataset-II, which consists of *subsets* of images obtained from several collections in Dataset-I (Table 5.3). The subsets of images were obtained in the same manner as keyword-based image retrieval: we specified additional keywords on an image collection in Dataset-I, and retrieved all the images whose tags match the additional keywords. In order to maintain high volumes of images in the subsets, we only obtained subsets from collections of 10000+ images (`Paris`, `Rome`, `Washington DC`), and specified keywords that correspond to popular landmarks (Notre Dame, Colosseum, Pantheon, Trevi Fountain, U.S. Capitol and Lincoln Memorial). The additional keywords, along with image statistics of the retrieved subsets, are listed in Table 5.3. One advantage of the proposed algorithm for canonical view ranking is that, once the ranking of canonical views $\pi(\mathcal{P})$ has been computed for an image collection $\mathcal{P}$, the ranking of canonical views for any subset of image collection $\mathcal{Q} \subseteq \mathcal{P}$ can be retrieved in real-time by simply scanning $\pi(\mathcal{P})$ and returning ranked images in $\pi(\mathcal{P}) \cap \mathcal{Q}$ (see Chapter 4). Therefore the ranking of canonical views for each image collection in Dataset-II can be obtained from its corresponding superset. The top two canonical views are shown

for each image collection in Dataset-II in Figure 5.5. Notice that for Colosseum, Pantheon, U.S. Capitol and Lincoln Memorial, the top two canonical views capture each landmark from exterior and interior. For Notre Dame, the top two canonical views capture the landmark from front and back. Trevi Fountain is the only landmark for which the top two canonical views cover similar viewpoints, because there is only one aspect of the landmark that attracts photographing. The purposes of Dataset-II are two-fold. First of all, canonical view retrieval for subsets of images has important applications in large-scale image browsing (see Chapter 4). Therefore it is important to evaluate the quality of the canonical views retrieved for subsets of images. Secondly, as we shall see in the next section, one of the quantitative measurements to evaluate canonical views requires manual inspection of the canonical views to determine their relevance to the topic (keywords) of the image collection. It would be difficult to conduct this measurement on Dataset-I, since each set of keywords in Dataset-I (such as `Rome`) covers a myriad of possibilities in relevant photographed objects. Thus the relevance of images is ill-specified. On the other hand, the relevance of images in Dataset-II can be specified by a simple criterion: an image is relevant if and only if the corresponding landmark is recognizable in the image, which effectively reduces subjectivity in the manual inspection.

Quantitative Measurements

As we develop quantitative measurements to evaluate canonical views, it is worth noting that the evaluation of *summaries* in general is a well-studied topic in

the research of text summarization. A number of metrics exist. The most successful ones, such as BLEU (standing for Bilingual Evaluation Understudy) [82] and ROUGE (standing for Recall-Oriented Understudy for Gisting Evaluation) [70], operate on two pieces of text summaries, one generated by human, the other by machine, and evaluate the later by counting their co-occurrences of linguistic units (such as n-grams). These automatic procedures achieve high correlation with human judgements, and are routinely used in text summarization benchmarks. In [68], Li and Merialdo bring the idea to the domain of video summarization. Their metric, termed VERT (standing for Video Evaluation by Relevant Threshold), operates on a human-generated video summary and a machine-generated one. The underlying statistical procedures are identical to BLEU and ROUGE, while the co-occurrences of linguistic units are replaced by the visual similarities of video frames. Unfortunately, this class of metrics is not applicable to canonical views (summaries of internet image collections). While it is relatively easy to manually generate summaries for text documents or video clips, it is almost impossible to do so for a noisy and unstructured collection of tens of thousands of images. To merely browse all the images, which is unavoidable during the manual process of summarization, would be prohibitively time-consuming.

We therefore resort to simple measurements that operate on the set of canonical views alone without human-generated ground truth. We propose to evaluate canonical views by three quantitative measurements: *noise*, *redundancy*, and *coverage*. In the following discussion, we follow the notation from previous chapters and denote an internet image collection by $\mathcal{P}$, and its k-set of canonical views by $\mathcal{C}_k$.

Measurement of noise. As discussed in Chapter 1, the goal of canonical view mining is to remove noise and redundancy from internet image collections. Therefore quantitative measurements on noise and redundancy are well motivated. Similar metrics have been adopted in the previous work of Kennedy *et al.* [64] (termed *precision* and *uniqueness* tests in their work, which are simply negations of the measurements of noise and redundancy). Similar to [64], we define $noise(\mathcal{C}_k)$ to be the number of images in $\mathcal{C}_k$ with irrelevant content to the topic (keywords) of $\mathcal{P}$. The relevance of image content is inspected manually. We consider an image to be relevant to the topic of $\mathcal{P}$ if and only if the image contains recognizable views of the objects depicted by the keywords of $\mathcal{P}$. In order to reduce subjectivity in the manual inspection, we only conduct this measurement on Dataset-II, where the topics of image collections are well-specified by popular landmarks. By definition, $noise(\mathcal{C}_k)$ ranges from 0 to k, with a lower value being more desirable.

Measurement of redundancy. We define $redundancy(\mathcal{C}_k)$ to be the number of images of duplicate content in $\mathcal{C}_k$. To be exact, we scan the list of images in $\mathcal{C}_k$ from top to bottom, and count images whose content has been previously seen. In practice, we automate this process by leveraging the existing inlier SIFT matches among all pairs of images collected during canonical view mining. We define the ith image in $\mathcal{C}_k$ to be a duplicate if and only if it matches to at least one of the first $(i-1)$ images in $\mathcal{C}_k$ with at least 16 inlier SIFT matches. By definition, $redundancy(\mathcal{C}_k)$ ranges from 0 to $k-1$, with a lower value being more desirable.

Measurement of coverage. Finally, we measure the summarization power of $\mathcal{C}_k$ over $\mathcal{P}$. We define $coverage(\mathcal{C}_k)$ to be the percentage of images in $\mathcal{P}$ whose content

is captured by at least one of the images in $\mathcal{C}_k$. Again, we automate this process by leveraging the inlier SIFT matches among all pairs of images. We define an image $P \in \mathcal{P}$ to be covered by $\mathcal{C}_k$ if and only if image P is matched to at least one of the images in $\mathcal{C}_k$ with at least 16 inlier SIFT matches. By definition, $coverage(\mathcal{C}_k)$ ranges from $\frac{k}{|\mathcal{P}|}$ to 1, with a higher value being more desirable.

Notice that the measurements of redundancy and coverage are automated using SIFT feature matching. The automatic procedures allow canonical sets of various sizes for various image collections to be evaluated with ease. Otherwise, the measurement of coverage, at least, is impossible to conduct, for it would entail a manual inspection of potentially tens of thousands of images in $\mathcal{P}$. SIFT features have been demonstrated on a controlled dataset to achieve both high precision and recall in image matching (with the same threshold of 16 inlier SIFT matches; see Chapter 3). Therefore it is reasonable to expect these automatic procedures to faithfully simulate manual inspections. One concern in the measurements of redundancy and coverage is that the SIFT features are known to have poor coverage over textureless objects, thus resulting in a bias toward texture-rich objects in image matching. However, in our experiments, the measurements of redundancy and coverage are always conducted on the *same* image collection to compare different subsets of images obtained using different methods. Since the photographed objects are constrained by the topic of the image collection, we would expect little variance in the texture-richness across different subsets of images. Therefore the bias is insignificant in the comparative results.

Experiments

With datasets and quantitative measurements in place, we evaluate the canonical views and compare the proposed algorithm for canonical view ranking to three other methods:

- Search engines. The ranking of images returned by the search engine Flickr or Google Images is used as a baseline for comparison. The two search engines rely on text matching between query keywords and image metadata to determine the relevance of images and rank the images by descending relevance. In the case of Flickr, image metadata is provided by the image owners via tagging. In the case of Google Images, image metadata is automatically crawled from the text fields of the webpages that host the images. In both cases, no visual feature is considered in the ranking of images. Therefore a high volume of noise is expected in the top-ranked images.

- Representative view ranking. Representative views are ranked by eigenvector centralities in the visual similarity graph (see Chapter 4). Since the algorithm for representative view ranking is based on the previous work of Jing and Baluja, this is equivalent to a comparison to their work [53]. The ranking of representative views advances the baseline by taking into account the visual similarity among images. Therefore much less noise is expected in the top-ranked images. However, because images of similar views are assigned similar representativeness scores, a high volume of redundancy is expected in the top-ranked images.

- The previous work of Simon *et al.* [92]. Simon *et al.* present a clustering-based method to select canonical views for an image collection. They apply greedy k-means to the visual similarity graph and iteratively add images to the canonical set while maximizing a k-means-like objective function. The objective function is designed such that each image in the collection should be represented by one of the canonical views, while the content among the canonical views should not overlap. The algorithm is reviewed in Chapter 2. Little noise or redundancy is expected in the selected canonical views. However, being a clustering-based method, the algorithm can only generate a fixed number of canonical views in one run. Worse, the number of canonical views cannot be controlled explicitly. It can only be affected by adjusting the parameters α and β in the objective function. We follow the same parameter setting: $\alpha = 5.75$ and $\beta = 100$, as adopted in the original paper. On most image collections, the algorithm terminates with less than 10 canonical views selected.

In the subsequent experiments, the three methods are denoted as follows: search engines – SE, representative view ranking – RV, greedy k-means – GK. The proposed algorithm for canonical view ranking is denoted by CV.

Measurements of noise and redundancy. In this experiment, we conduct the measurements of noise and redundancy on Dataset-II. We evaluate three of the four methods – SE, RV and CV. Greedy k-means is omitted because it only manages to select 2 or 3 canonical views per image collection. Therefore there is not enough data to robustly evaluate its performance. We apply each of the three methods – SE, RV

and CV – to each of the six image collections in Dataset-II to retrieve top-ranked images. This yields a total of 18 sets of images. For each set of images, we compute the scores of noise and redundancy as defined in the previous section for the top k images in the set, where k varies from 1 to 20. Therefore we have computed a $3\times6\times20$ matrix, where the (i, j, k)th cell stores the noise and redundancy scores for the top k images retrieved from image collection j using method i. Finally, we average the noise and redundancy scores along the second dimension – across all the image collections in Dataset-II. In Figure 5.6, the average noise and redundancy scores are plotted against the top-ranked images for each of the three methods. As expected, the search engine results contain a high volume of noise (averaging 10.7 of the 20 images) while the representative views contain a high volume of redundancy (averaging 13.0 of the 20 images). The ranking of canonical views easily achieves the best performance out of the three, averaging only 2.3 noisy views and 2.0 redundant views in the top 20 images.

Measurement of coverage. In this experiment, we conduct the measurement of coverage on Dataset-I. We apply each of the four methods – SE, RV, GK and CV – to each of the eleven image collections in Dataset-I to retrieve top-ranked images. This yields a total of 44 sets of images. For each set of images, we compute the coverage score as defined in the previous section for the top k images in the set, where k varies over $\{1, 5, 10, 20, 40\}$. We divide the image collections in Dataset-I by their categories: *places*, *products* and *artworks*, and average the coverage scores across all the image collections within the same category. The reason not to average across different image categories is because image collections in different categories (especially

places vs. the other two) are drastically different in both volume and complexity of image content. In Figure 5.7, the results for the three image categories are presented separately. For each image category, the average coverage score is plotted against the top-ranked images obtained using four methods. Several observations can be drawn from the results. First of all, the ranking of canonical views consistently outperforms the other three methods. The ranking of representative views and greedy k-means achieve similar, slightly inferior performances. The search engines consistently yield the poorest performance. Since the measurement of coverage indicates summarization power, the ranking of canonical views is demonstrated to be the best among the four methods to compose a visual summary for the image collection. Another interesting observation is that the search engine of Google Images seems to compare favorably to that of Flickr, even though both rely on textual information to determine the relevance of images: in the Google categories (*products* and *artworks*), the performance of the search engine, albeit the poorest, is very close to the other three methods, while in the Flickr category (*places*), the performance of the search engine falls behind the other three methods by a large margin. One probable explanation is that the textual information leveraged by Google Images (webpage content) is much richer than that by Flickr (user-added tags), therefore the former stands a better chance in understanding the image content and determining its relevance to query keywords. Nonetheless, the best performance achieved by the text-based methods is still inferior than any of the content-based methods. Finally, notice that in the *places* category, as much as about 10% of the original image collection (1000+ images) is covered within the top 40 canonical views. The large compression ratio and excellent

summarization power of top-ranked canonical views open up new possibilities for efficient manipulation of large image databases. In Chapter 6, we shall demonstrate such an application of canonical views in the context of object recognition.

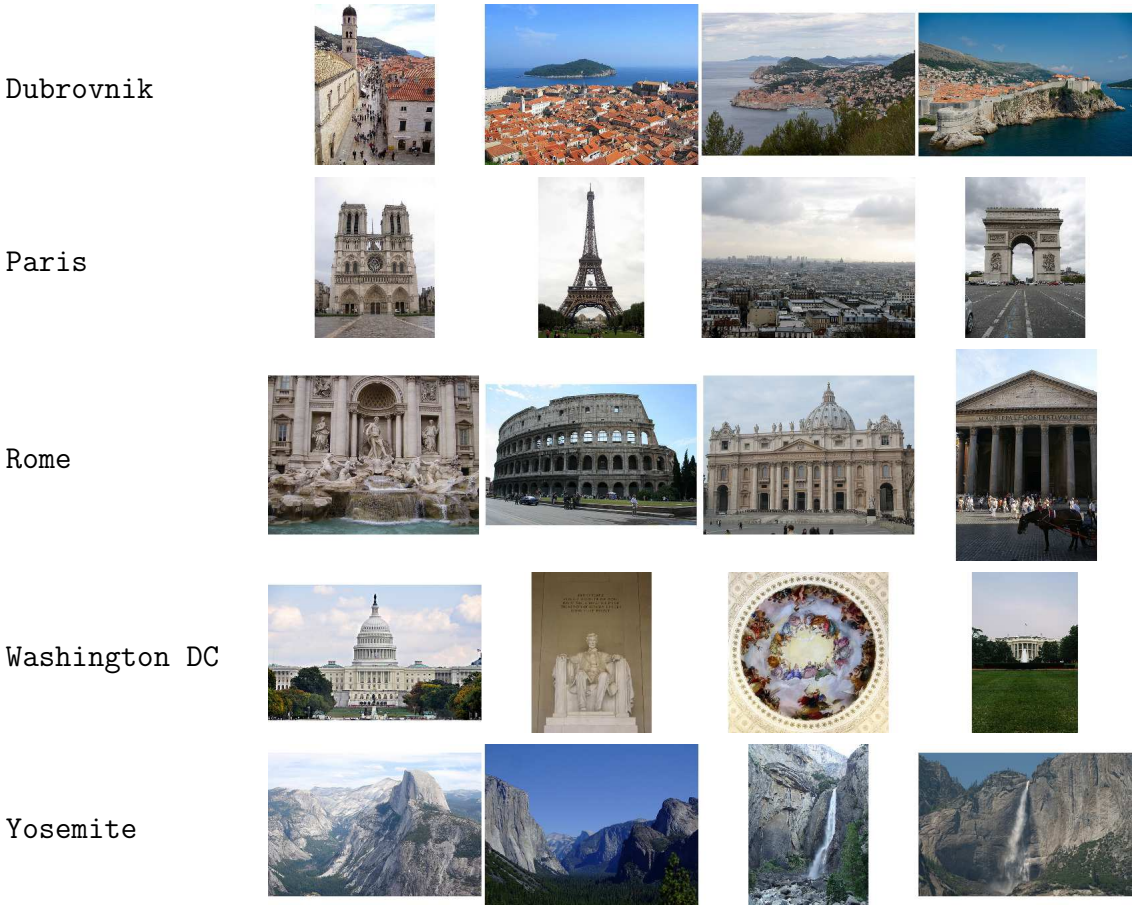


Figure 5.1: Canonical views for image collections of *places*. The left column shows the keywords and the right column shows the top four canonical views for corresponding image collections. Many of the canonical views capture popular landmarks of the places. For example, `Dubrovnik` – Stradun Street (first image); `Paris` – Notre Dame, Eiffel Tower, Arc de Triomphe (first, second and fourth images); `Rome` – Trevi Fountain, Colosseum, St. Peter's Basilica and Pantheon; `Washington DC` – U.S. Capitol, Lincoln Memorial, Capitol Dome, White House; `Yosemite` – Glacier Point, Yosemite Valley, Yosemite Falls (lower and upper sections). Moreover, almost all the landmarks are captured in their most iconic viewpoints. Images of near-identical viewpoints to these can be found on authoritative tourism websites such as the Wiki pages of these landmarks (see Figure 5.2). Apart from landmarks, some canonical views in the collections `Dubrovnik` and `Paris` capture views of the cityscapes, because such views tend to connect many images in the similarity graph, and thereby gain large supports during canonical view ranking.

(a) Stradun Street (b) Notre Dame (c) Eiffel Tower (d) Arc de Triomphe

(e) Trevi Fountain (f) U.S. Capitol (g) Glacier Point (h) Yosemite Valley

Figure 5.2: Comparisons of landmark viewpoints between the canonical views and the images collected from Wiki pages. We select several landmarks from the top-ranked canonical views in the `Dubrovnik`, `Paris`, `Rome`, `Washington DC` and `Yosemite` collections (see Figure 5.1). For each landmark, the image on the left shows the canonical view automatically selected by our algorithm, and the image on the right shows the image collected from the Wiki page of the landmark. For almost all these landmarks, the two images are captured from near-identical viewpoints.

Figure 5.3: Canonical views for image collections of *products*. The left column shows the keywords and the right column shows the canonical views for corresponding image collections. Almost all the canonical views capture either the popular products under the brand names or the brand logos.

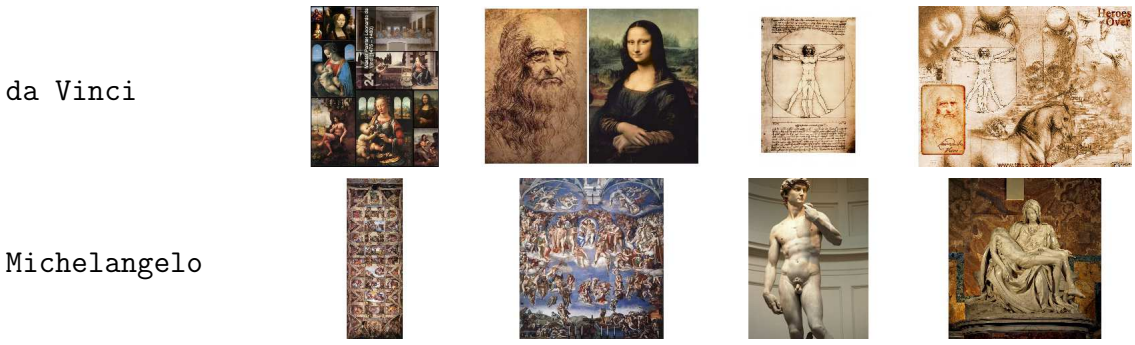
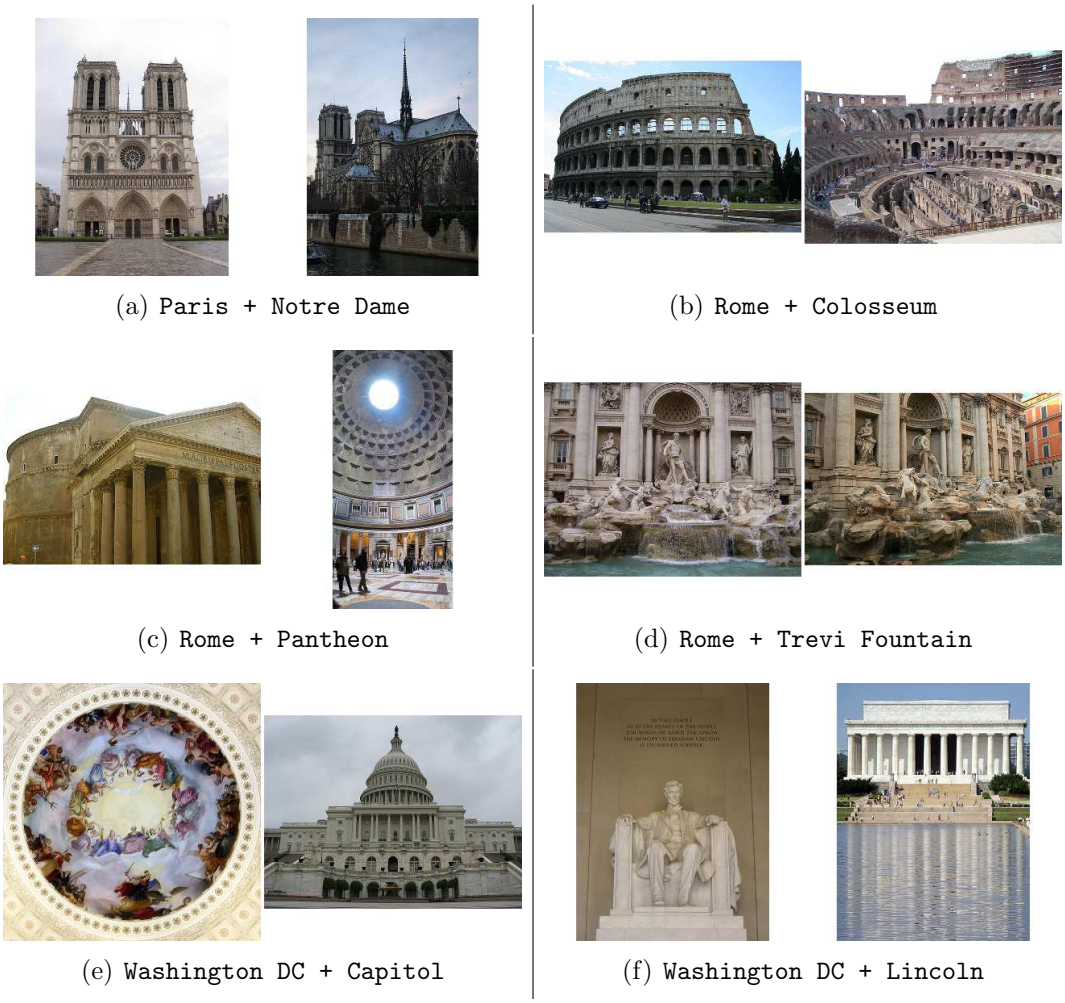


Figure 5.4: Canonical views for image collections of *artworks*. The left column shows the keywords and the right column shows the canonical views for corresponding image collections. The canonical views capture most of the masterpieces for the two artists as listed on their Wiki pages: `da Vinci` – self portrait, Mona Lisa, The Vitruvian Man, and several others in the first and fourth image collages; `Michelangelo` – the ceiling of the Sistine Chapel, The Last Judgment, David, Pietà.



(a) `Paris + Notre Dame`

(b) `Rome + Colosseum`

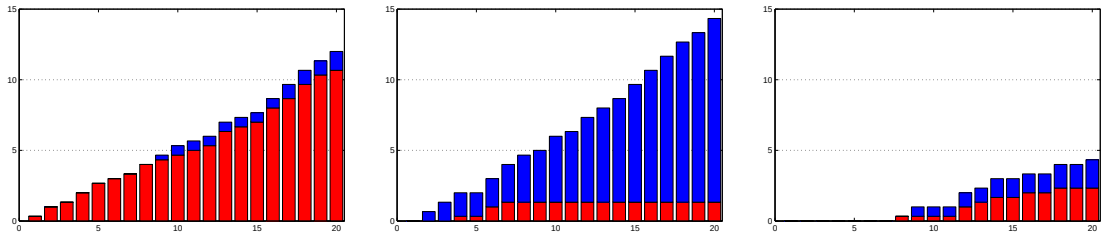
(c) `Rome + Pantheon`

(d) `Rome + Trevi Fountain`

(e) `Washington DC + Capitol`

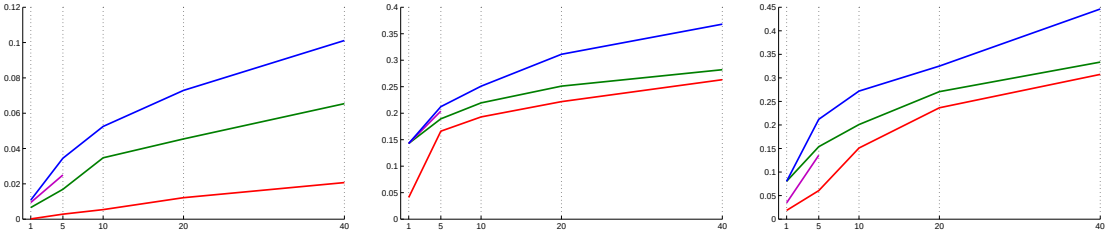
(f) `Washington DC + Lincoln`

Figure 5.5: Canonical views for Dataset-II. The image collections in Dataset-II are labeled by their keywords. The top two canonical views are shown for each image collection. Notice that for Colosseum, Pantheon, U.S. Capitol and Lincoln Memorial, the top two canonical views capture each landmark from exterior and interior. For Notre Dame, the top two canonical views capture the landmark from front and back. Trevi Fountain is the only landmark whose top two canonical views cover similar viewpoints, because there is only one aspect of the landmark that attracts photographing.



(a) SE (b) RV (c) CV

Figure 5.6: Measurements of noise and redundancy. The measurements of noise (red) and redundancy (blue) are averaged across all the image collections in Dataset-II and plotted against the number of top-ranked images obtained using three methods: SE – search engine (Flickr), RV – representative view ranking, CV – canonical view ranking. This figure is best viewed in color.



(a) *places* (b) *products* (c) *artworks*

Figure 5.7: Measurement of coverage. The measurement of coverage is averaged across all the image collections of *places* (left), *products* (center) and *artworks* (right), and plotted against the number of top-ranked images using four methods color-coded by: red – search engine (Flickr or Google Images), green – representative view ranking, purple – greedy k-means, blue – canonical view ranking. Notice that image collections in different categories are drastically different in both volume and complexity of image content. Therefore the measurement of coverage is not to be compared across different categories (notice that the Y coordinates are in different scales in the three plots). This figure is best viewed in color.

CHAPTER 6

CANONICAL VIEWS: AN APPLICATION FOR OBJECT RECOGNITION

The applications of canonical views extend beyond image browsing. With the proliferation of internet image collections, researchers have taken advantage of the excellent resource to tackle some of the most fundamental problems in computer vision, such as object recognition. A wave of work has been proposed that bypasses parametric modeling for object classes to interpret unseen images directly by their nearest neighbors in a large database of images of known interpretations [45,99]. This class of work has achieved promising results for many currently unsolvable problems such as general object recognition and scene understanding. However, the complexity of nearest neighbor search becomes prohibitive as the database scales. In this chapter, we argue that such methods can potentially benefit by the use of canonical views. By removing noise and redundancy from the database, we expect the set of canonical views to be compact while still preserving most of the representative power for the purpose of object recognition. We validate this hypothesis on the *place recognition* problem, in which we estimate the geographic location of an image by matching its photographed scene to a large database of images of known locations.

Introduction

In the study of cognitive psychology, *where* is considered to be one of the most important memory cues for recalling past events [104]. Since images are records of past events, it is not surprising that location information serves an important role in indexing and searching internet images [107]. On most photo sharing sites, it is a common practice to allow users to browse images on the map at locations of capture. We have already seen one example, TagMaps [15], in Chapter 2.

Location-based image browsing is an effective way to view images in context. However, in order to place an image on the map, the image must be associated with a GPS-tag, which is specified by a latitude/longitude coordinate. In general, the GPS-tags of images can be acquired in three ways. First of all, many cameras and smart phones nowadays have GPS devices embedded, so a GPS-tag is automatically saved to the EXIF fields of the image file upon capture. Secondly, many photographers (especially tourists) use external GPS devices to record their route of travel. Therefore the GPS-tags for the images captured along the route can be retained. Finally, most photo sharing sites provide a map-based tool where users can manually specify the locations of the images by drag-and-drop. Unfortunately, compared to the volume of internet image collections, the portion with GPS tags is still relatively small: by querying on Flickr using popular keywords and switching on the filter for GPS-tagged images, we estimate that GPS-tagged images make up fewer than 10% of all images on Flickr.

Although small in relative number, the absolute number of GPS-tagged images is gigantic, in the order of hundreds of millions. Moreover, the collection of GPS-tagged images provides an excellent coverage of the globe. Given an image of unknown location, it is very likely that there exist multiple GPS-tagged images of the same scene. This motivates a data-driven scene matching approach to acquiring GPS tags for images: having collected a database of images of known locations, a query image is matched to the database to retrieve images of matching scenes; the locations of the matched database images thereby provide an estimate of the location of the query image. This approach has demonstrated promising results in the literature [46, 88] (see next section for a review).

However, traditional strategies for image retrieval, which involve a linear scan of the database (such as [46]), lead to suboptimal performance on internet images collections. As discussed in Chapter 1, internet image collections contain massive amounts of noise and redundancy. For the purpose of place recognition, noise includes images focusing on people or objects and showing little background scene; redundancy includes near-duplicate images of the same scene (such as a landmark view). During a linear scan of the database, much of the computation is wasted either on matching to noisy images that reveal little geographic information, or on matching to images of redundant scenes over and over again. Little previous work on place recognition has addressed the noise and redundancy issues.

Inspired by the study of canonical views, we propose to minimize the impact of noise and redundancy on place recognition by compressing the original database into a compact representation by a small number of canonical views. Ideally, the set

of canonical views should eliminate noise and redundancy from the original database, while preserving a diverse set of representative views. Therefore, by restricting the scene matching of a query image to the canonical views, we expect to improve the efficiency of query processing significantly with minimal loss in recall rate.

Previous Work

This work is most closely related to [46, 88]. In [88], Schindler *et al.* propose a system for place recognition on the scale of a city. They collect a database of 30K GPS-tagged streetside images of a city. SIFT features are extracted from all database images and organized by a vocabulary tree [79] for efficient matching. The location of a query image is given by its top match in the database using SIFT features. In IM2GPS [46], Hays and Efros leverage a database of over 6 million GPS-tagged internet images for place recognition on the scale of the globe. A query image is matched to all database images using a combination of visual features. Mean shift [28] is applied to the locations of the top k matched images in the database to find the modes (density centers) in their geographic distribution. The locations of the density centers serve as the estimated locations for the query image. In [58], IM2GPS is extended to predict the locations for a sequence of images with timestamps. A prior distribution of travel patterns is trained on a large database of images. Therefore the location of an image can be predicted not only based on its scene matching results, but also based on the location predictions of temporally adjacent images in the sequence. Our work adopts the data-driven scene matching approach of [46, 88], but improves

the efficiency of query processing by compressing the original database into a compact representation by canonical views.

With the maturity of large-scale image-based modeling [18], a wave of work adopts structure from motion (SfM) techniques to reconstruct a 3D point cloud from internet image collections, and uses the point cloud as a basis for place recognition [49, 67, 69]. By registering images in 3D and reconstructing the point cloud, various statistics can be accumulated such as the view count for each 3D point. Therefore, the scene structure can be exploited (*e.g.*, which 3D points and associated 2D views appear more frequently in the images) to improve the efficiency of query processing by prioritizing 3D points for matching [69], compressing the 3D point cloud to a minimal cover of the location [49], and building an iconic scene graph for matching [67]. Our work is similar to this class of work in that we also exploit the scene distribution of database images, but our work does not rely on SfM, which often entails more requirements on the images (such as EXIF fields with the focal length information) and a higher computational cost.

Algorithm

The algorithm consists of offline and online stages. During the offline stage, a large database of GPS-tagged images is compressed into a small subset of canonical views. During the online stage, the query image is matched to each canonical view until a matching scene is found, upon which the location for the query image can be predicted.

Canonical View Selection

The algorithm for canonical view ranking is described in Chapter 4. Given input images $\mathcal{P} = \{P_1, P_2, \cdots, P_n\}$, the output of the algorithm is a permutation π of $\mathcal{P}$ such that the top k images in the $\pi(\mathcal{P})$ approximate the k-set of canonical views for $\mathcal{P}$. In the sequel, we add two post-processing steps to finalize a subset of canonical views from $\pi(\mathcal{P})$.

Redundancy removal. Top-ranked images in $\pi(\mathcal{P})$ are usually diverse. However, redundant views start appearing towards the middle of $\pi(\mathcal{P})$. This is not surprising – the original algorithm does not *remove* any redundant views; it only *demotes* them in the canonical view ranking. For the purpose of place recognition, redundant views bring little new information to a database. The first post-processing step aims to remove redundant views by the following procedure: we scan the list of images in $\pi(\mathcal{P})$ in order, and remove all subsequent images that have at least 16 inlier SIFT matches to the current one, until the list is exhausted. After this procedure, few redundant views remain.

Noise removal. Top-ranked images in $\pi(\mathcal{P})$ usually have large supports in their suppression radiuses, but the support drops sharply towards the middle of $\pi(\mathcal{P})$. Towards the end of $\pi(\mathcal{P})$, there is a long tail of images with tiny supports in their suppression radiuses, which indicates only a handful of images sharing a same scene. The popularity of the scenes are so low that these images are rarely matched by query images in place recognition. In the second post-processing step, we treat the long tail of images as noise and remove any image with a support less than a certain threshold

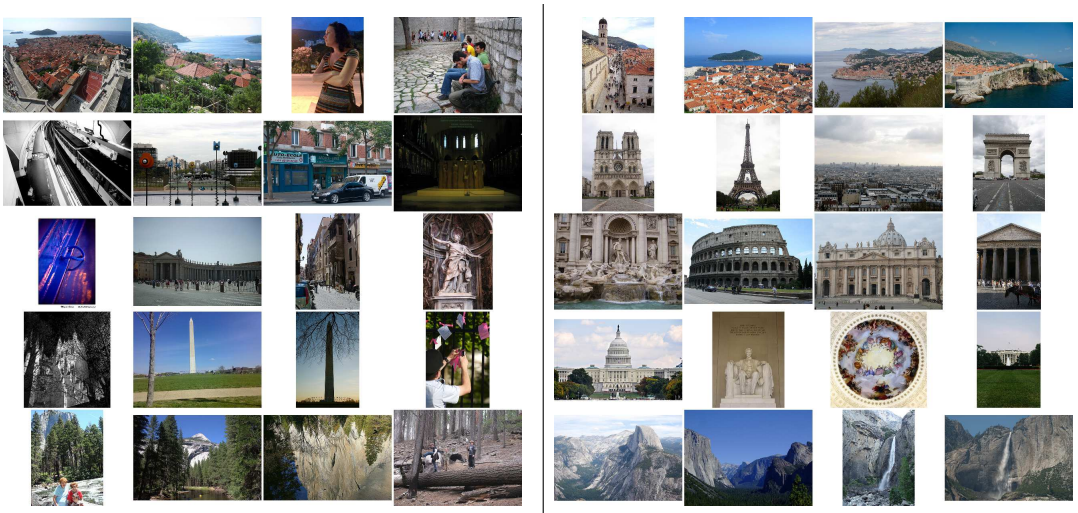
t in its suppression radius. In our experiments, we empirically set $t = 2$, which offers a good tradeoff between the compactness and coverage of canonical views. The threshold indicates that any canonical view must represent a minimum of 3 images (2 in the suppression radius plus 1 for the canonical view itself).

Noise and redundancy abound in internet image collections. The two post-processing steps lead to about 96% compression of a database in our experiments. The remaining 4% of database images are selected as canonical views and ordered by the canonical view ordering π for place recognition.

Scene Matching

Having a database of images of known locations and an ordered list of canonical views, estimating the location for a query image is straightforward: we scan the list of canonical views from top to bottom, matching each canonical view to the query image. If a matching scene is found, the scan is terminated, and the location of the matching scene is reported as the estimated location for the query image. If we exhaust the list of canonical views without a match, the query image is rejected as not at any location covered by the database.

Since query processing is terminated as soon as a match is found, we aim for a low false positive rate during the scene matching between canonical views and a query image. As discussed in Chapter 3, we use SIFT features to match images, followed by a geometric verification using RANSAC [35] on the fundamental matrix [44]. After the geometric verification, if the number of remaining SIFT matches exceeds a certain



(a) Random views (b) Canonical views

Figure 6.1: A comparison between random views and canonical views for place recognition. Each row shows four random views and top-ranked canonical views for a site (from top down: Dubrovnik, Paris, Rome, Washington DC and Yosemite). The random views shed light on the amount of noise in internet image collections and justify our approach of canonical views, in which little noise or redundancy is observed.

threshold (16 in our experiments), the pair of images is deemed to be a match. In a robustness test where we match images from different sites, we observe zero false positives using the described procedure and threshold.

Experiments

We evaluate place recognition on the same dataset as canonical view evaluation in Chapter 5. The database contains 51053 images of five sites: Dubrovnik, Paris, Rome, Washington DC and the Yosemite National Park. The top canonical views for place recognition are shown in Figure 6.1. We keep track of the number of canonical views after the first and second post-processing steps are applied to the original output. As shown in Table 6.1, there is a significant reduction in the number of

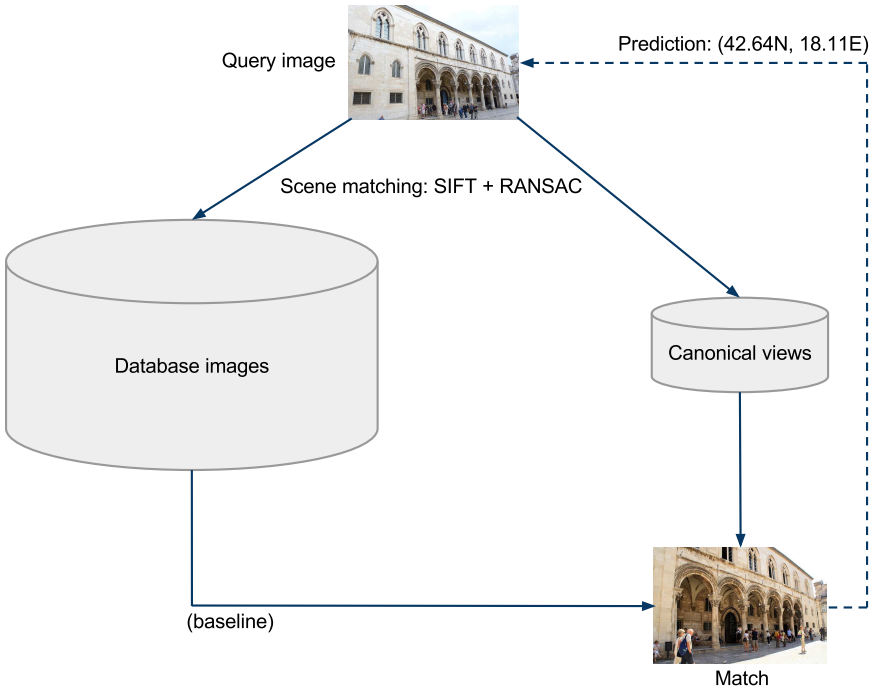


Figure 6.2: The experimental setup for place recognition. Each test image is matched to the database images as well as the canonical views. In both sets of images, the first scene match to the query image, if any, predicts the GPS tag for the query image. This process is repeated for each test image. The performance of the canonical views is compared to that of the database images in terms of efficiency and recall.

canonical views after each step is applied. Together, a 96% reduction is attained. By restricting query processing to this set of canonical views, the maximum processing time for a query image (proportional to the number of scene matching operations) is also reduced by about 96%.

The significant reduction in processing time leads to some loss in recall rate: some images that could have been matched to the database may fail to do so because all the corresponding database images are missing from the canonical set. The loss in recall rate is quantitatively measured by conducting place recognition on a test set of

Table 6.1: Statistics of database images and canonical views. The last two columns show the number of canonical views after the first and second post-processing steps are applied to the original output. Notice that a consistent $94 - 98\%$ reduction of images is attained after both steps.

Dataset	♯ images	P1	P2	% reduction
Dubrovnik	9350	6059	520	94.44%
Paris	11997	9854	407	96.61%
Rome	11959	8951	433	96.38%
Washington DC	11991	10528	295	97.54%
Yosemite	5756	3923	257	95.54%

images against both the original database and the canonical views. The experimental setup is illustrated in Figure 6.2.

The test set consists of 400 GPS-tagged images for each of the five sites, yielding 2000 query images in total. The GPS tags of query images are only used for localization error analysis. Images in the test set are downloaded from Flickr in the same manner as database images. However, a filtering is applied to ensure that the database and the test set share no image/user in common.

Notice that the ground-truth recall rate will not be 100%. Since the query images are just as noisy as database images, a majority of them cannot be matched even by a full scan of the database. Since we are interested in the loss of recall caused by canonical views, our ground-truth recall rate is the one where query images are matched to the entire database.

For each query image, we match it against all database images in random order. This provides ground truth data. Then we apply the scene matching algorithm to match the query image against the canonical views for all sites – not only the canonical views of the same site, but those of the other sites as well – to test the robustness

Table 6.2: Comparison of efficiency, precision, and recall. GT refers to the ground-truth method; CV refers to the method with canonical views. Efficiency is measured by the average number of scene matching operations per query.

	GT	CV	CV gain
efficiency (♯ ops)	2540.76	65.216	97.43%
	GT	CV	CV gain
precision (% correct)	98.21%	100%	1.79%
	GT	CV	CV loss
recall (♯ correct matches)	275	206	25.09%

of place recognition. In both the database images and the canonical views, the first scene match to the query image, if any, predicts the GPS tag for the query image.

Out of 2000 query images, 280 have at least one match to the original database, and 206 have at least one match to the canonical views. In the sequel, we analyze efficiency, precision, recall, and localization error in more depth.

Efficiency. Efficiency is measured by the average number of scene matching operations per query. A comparison of efficiency is shown in Table 6.2, where the method with canonical views saves as much as 97% of scene matching operations, reducing the average time for query processing from several minutes to a few seconds. Notice that the canonical views are ordered by descending representativeness. This means that a majority of query images are expected to match to the top fraction of canonical views, which results in extremely efficient processing. Figure 6.3 plots the growth of matched query images as more canonical views are scanned.

Precision. Precision of scene matching is measured by the percentage of correctly matched query images among all matched query images. For each matched query image, we manually inspect its first match to both the entire database and

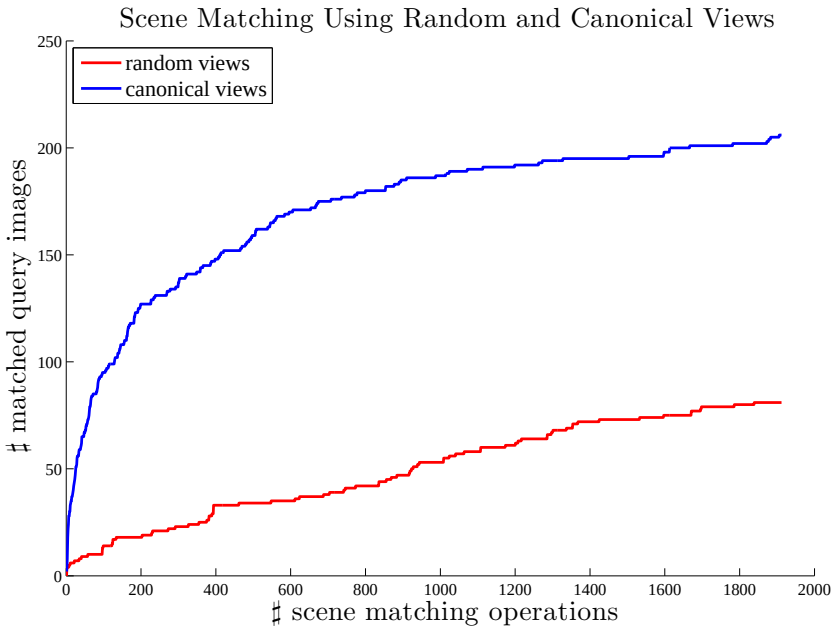


Figure 6.3: A comparison between the random views and canonical views for scene matching. The number of matched query images is plotted against the number of scene matching operations required by using random views and canonical views. Notice that a majority of query images are matched by the top fraction of canonical views, which results in extremely efficient query processing.

the canonical views (if any) and determine if the matched images indeed share the same view of the same place with the query image. Of the 280 matches to the entire database, 5 query images are matched on indoor objects or street signs of *different places*, causing incorrect estimates of their geographic locations. Therefore the precision of scene matching to the entire database is 98.21%. In comparison, those 5 query images are all absent in the canonical views. All 206 matches to the canonical views share the same view of the same place as the query images, leading to a 100% precision. In particular, no query image is matched to any image of a *different* site, which demonstrates the robustness of the strict procedure of SIFT matching and geometric verification.

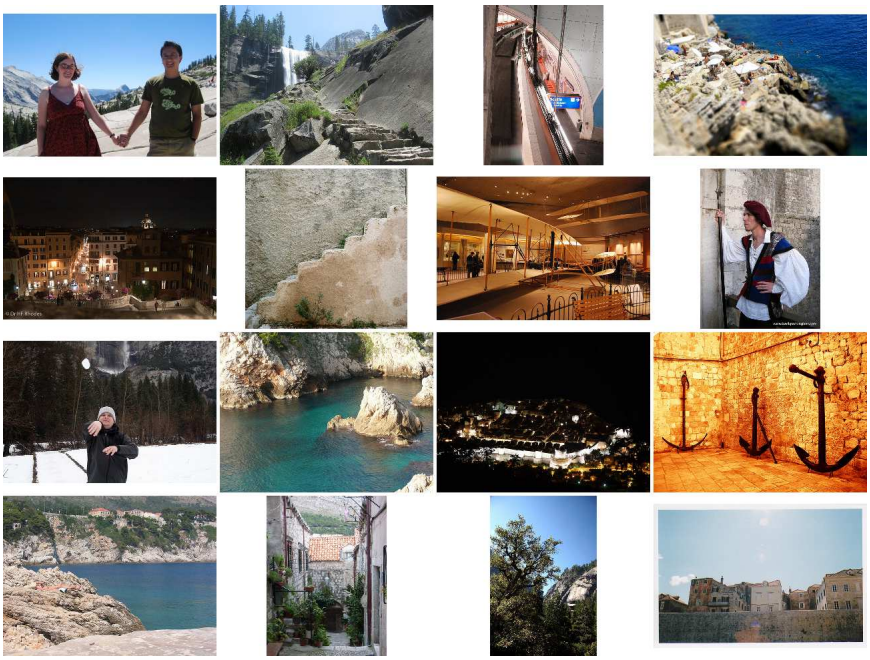


Figure 6.4: Samples of difficult query images. 16 random samples are shown of query images having $\leq$ 2 matches in the database, which indicate rarely photographed scenes.

Recall. We are interested in the *relative* recall: among all query images that can be matched by the entire database, the percentage that are matched by the set of canonical views. The recalls are shown in Table 6.2, where the method with canonical views suffers a 25% loss. The loss in recall is low considering that the size of the canonical set is only 4% of the entire database. Moreover, by inspecting the query images corresponding to the loss, we find that more than 95% of such images have ≤ 2 matches in the entire database, which indicate rarely photographed scenes (see Figure 6.4). Place recognition for rarely photographed scenes is inherently a difficult problem. A small change in the database may result in different match/reject decisions for such images. Therefore, we believe it is worth making a sacrifice on such images in exchange for a significant improvement in efficiency.

Localization error. Finally, we analyze the localization error of the scene matching approach to place recognition. Each matched query image has two GPS tags: one of its own (treated as ground truth) and the other predicted by a matched database image. Localization error is measured by the great-circle distance between the two GPS tags. Among all matched query images, the median localization error is 30.82m, which is promising given that the typical precision of civilian GPS devices is about 20m [6]. The lower and upper quartiles of localization errors are 14.57m and 103.98m respectively. Only a few predicted GPS tags are far off their ground truth (up to 6km), all of which are caused by incorrect GPS-tagging of either the query image or the matched database image, not by scene matching.

Summary

An efficient method for place recognition is proposed. The principal novelty of the method is in compressing a database of images into a compact representation by canonical views. The use of canonical views eliminates noise and redundancy in the database, and enables efficient place recognition with a reasonable recall rate. The data also suggests that the precision of place recognition is slightly improved, thanks to the removal of noise in the canonical views. This experiment validates our hypothesis that the set of canonical views, albeit small, preserves most of the representative power of the original database for the purpose of object recognition. Therefore canonical views can potentially benefit a number of techniques that rely on large-scale nearest neighbor search for object recognition.

CHAPTER 7

CANONICAL VIEWS: SCALABILITY

We have proposed a pipeline for canonical view mining from internet image collections. We have evaluated the quality of the canonical views, and demonstrated their applications for large-scale image browsing and efficient object recognition. In this chapter, we discuss the scalability of the pipeline. We single out pairwise image matching as the scalability bottleneck, and propose an approximation algorithm to remove the bottleneck. We evaluate the approximation algorithm by efficiency and accuracy for pairwise image matching. We demonstrate that the approximation algorithm speeds up pairwise image matching (and the entire pipeline) by two orders of magnitude with low impact on the resulting canonical views.

Time Analysis

The pipeline for canonical view mining is recapitulated in Figure 7.1. As we shall see, the pipeline demands significant amounts of computational resources, yielding poor scalability to large image collections. In the sequel, we walk through the main stages in the pipeline, inspect their time complexities, and recognize the scalability bottleneck. For the purpose of comparison, we report the runtime of each stage on the `Rome` collection of 11959 images (summarized in Table 7.1). The runtime distributions for other image collections are similar.

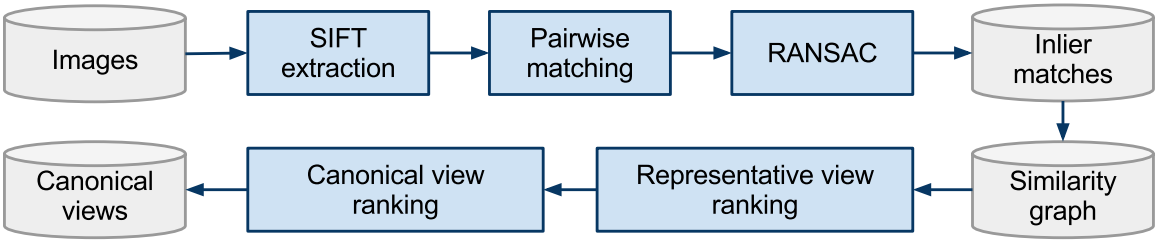


Figure 7.1: The pipeline of canonical view mining.

Table 7.1: Runtime of canonical view mining on the `Rome` collection.

Stage	Time
SIFT feature extraction	$\approx$ 5.5 CPU hours
Pairwise image matching	$\approx$ 2212 CPU hours
Geometric verification	$\approx$ 40 minutes
Representative view ranking	13.94 seconds
Canonical view ranking	4.43 seconds

SIFT feature extraction. On average, SIFT feature extraction takes about 1.6 seconds per image. The exact runtime is largely dependent on the resolution and texture-richness of the image. In our experiments, we have downscaled all the large images in the collections to a standard resolution (640 pixels on the maximum dimension). Therefore we observe little deviation in the runtime among different images. The total runtime of this stage grows linearly in the number of images. On the `Rome` collection, SIFT feature extraction for 11959 images amounts to about 5.5 CPU hours. Notice that SIFT feature extraction operates on each image independently. Therefore, a large image collection can be easily partitioned and processed by a cluster of CPUs. In our experiments, we parallelize SIFT feature extraction on 6 CPUs to reduce the wall clock time of this stage to less than one hour.

Pairwise image matching using SIFT features. In Chapter 3, we have addressed the efficiency of matching two images using SIFT features by approximate nearest neighbor search, and successfully reduced the runtime of image matching to about 0.1 second per pair. Yet, this stage proves to be the most time-consuming of the entire pipeline, because the requirement for *pairwise* image matching causes the total runtime to grow *quadratically* in the number of images. For the `Rome` collection of 11959 images, pairwise image matching translates into matching $(11959 \times (11959 - 1))/2 = 71,502,861$ pairs of images! Even with efficient image matching (about 0.1 second per pair), this amounts to about 2212 CPU hours. In our experiments, we parallelize pairwise image matching on a cluster of 40 CPUs. Still, this stage takes the longest wall clock time, about 3 days per large collection of 10000+ images.

Geometric verification using RANSAC. Like image matching, geometric verification using RANSAC also operates on pairs of images. Therefore, in the worst case, the total runtime of this stage could be quadratic in the number of images. In practice, however, geometric verification is only triggered on image pairs with at least 16 SIFT matches (see Chapter 3). Due to the massive amount of noise in internet image collections and the low false positive rate of SIFT feature matching, only a tiny portion of image pairs are qualified for geometric verification. Out of the 71,502,861 image pairs in the `Rome` collection, geometric verification is triggered on 239,990 pairs, which takes about 40 minutes runtime on a single CPU (averaging about 0.01 second per pair). After geometric verification, 22262 image pairs remain.

Representative view and canonical view ranking. Compared to the previous stages, the runtime of representative view and canonical view ranking is negligible.

On a collection of N images, representative view ranking entails a small number of matrix-vector multiplications between an $N \times N$ stochastic matrix and an $N \times 1$ vector of eigenvector centralities. Canonical view ranking entails scanning the list of representative views N times. On the `Rome` collection ($N = 11959$), both stages finish in a matter of seconds on a single CPU.

From the time analysis, we can easily single out pairwise image matching as the scalability bottleneck of the pipeline. On a large collection of 10000+ images, pairwise image matching takes thousands of CPU hours, accounting for more than 99% of the total runtime of the pipeline. Even worse, due to the massive amount of noise in internet image collections, true image matches are extremely sparse: in the `Rome` collection, only 22262 image pairs survive the SIFT feature matching and geometric verification, which accounts for about 0.03% of the total 71,502,861 image pairs. In other words, about 99.97% of the runtime in pairwise image matching is spent on images that do not match.

Approach

We propose a *large kd-tree* for *approximate image matching*. A large kd-tree indexes feature descriptors for a database of N images. By querying the large kd-tree, a query image can be matched to all N images in one run. We enable approximate nearest neighbor search on the large kd-tree to reduce the time complexity of 1-to-N image matching from $\mathcal{O}(N)$ to $\mathcal{O}(log(N))$. On a database of 10000+ images, 1-to-N image matching achieves a significant speedup over the conventional N times

1-to-1 image matching (henceforth referred to as *exact image matching*). Due to the approximation in nearest neighbor search, the retrieved feature matches to the database images may be sparse. However, we shall demonstrate that the feature matches can predict with a high recall which of the N images are likely to match to the query image.

Based on a large kd-tree, we present a prediction-verification scheme for pairwise image matching (Figure 7.2). Given a database of N images for pairwise matching, we build a large kd-tree to index the feature descriptors from all N images. In the prediction stage, we issue each database image as a query to the large kd-tree and collect a set of predictions of image matches. In the verification stage, we verify the correctness of each prediction by exact image matching. False image matches are discarded, and true image matches are retained. Due to the high sparsity of image matches in internet image collections, we expect the output from the prediction stage (the input to the verification stage) to be much smaller than the full set of $N(N-1)$ image pairs. Therefore the prediction-verification scheme can achieve a significant speedup over pairwise image matching.

In the literature, a wave of work has focused on indexing feature descriptors for a database of images, thereby enabling 1-to-N image matching (see next section for a review). Our work is different from previous work in two aspects. First of all, previous work requires all the feature descriptors that are to be indexed and searched to reside in the main memory. This requirement imposes a high memory consumption on the kd-tree and a poor scalability to large image databases. Thus previous work can hardly scale beyond several thousand regular-sized images (about

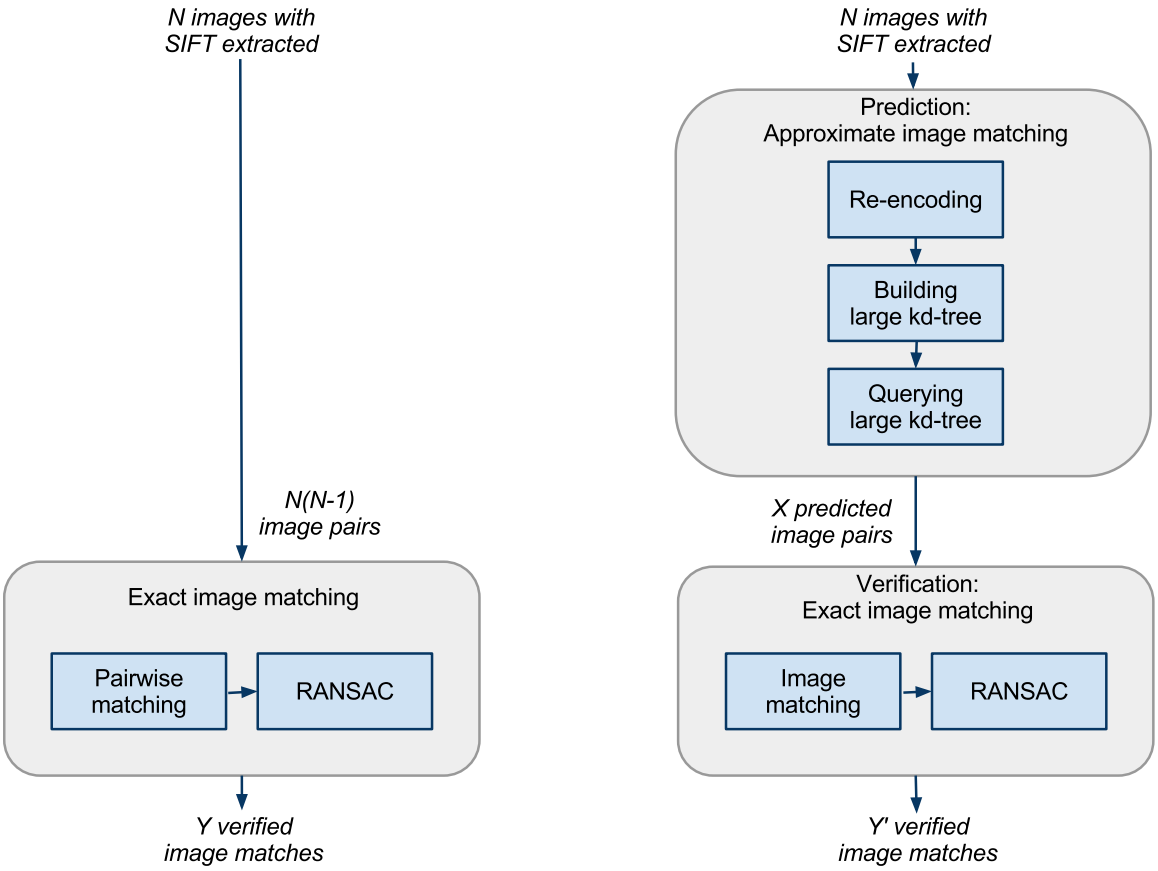


Figure 7.2: The prediction-verification scheme for pairwise image matching. The left diagram shows the conventional scheme for pairwise image matching, where all $N(N-1)$ pairs of images are processed by exact image matching, and qualified image pairs (with at least 16 feature matches) are subject to geometric verification using RANSAC. The right diagram shows the prediction-verification scheme for pairwise image matching. A large kd-tree is built to index the feature descriptors from all N images. In the prediction stage, each database image is issued as a query to the large kd-tree and a set of predictions of image matches is collected. In the verification stage, the correctness of each prediction is verified by exact image matching. False image matches are discarded, and true image matches are retained. Due to the high sparsity of image matches in internet image collections, we expect the output from the prediction stage (the input to the verification stage) to be much smaller than the full set of $N(N-1)$ image pairs (*i.e.*, $X \ll N(N-1)$). Therefore the prediction-verification scheme can achieve a significant speedup over pairwise image matching.

several million SIFT features). On the other hand, we propose disk-based kd-tree construction and nearest neighbor search, which allows a majority of the feature

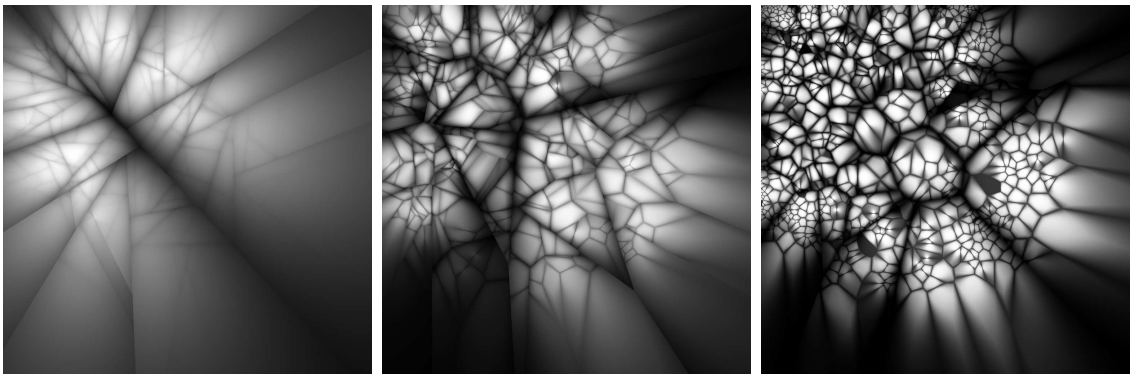


Figure 7.3: Projections of vocabulary trees with different branching factors. The three vocabulary trees are constructed using the same set of 10 thousand SIFT features. From left to right, the branching factors are 2, 10, and 100. Hierarchical k-means forms a hierarchy of Voronoi diagrams in the 128-dimensional feature space. The images show their projections onto two random dimensions. Pixel intensities indicate the ratio of the distances to the first and second nearest nodes on each level of the vocabulary tree. Therefore the darkest values (ratio ≈ 1) indicate the boundaries of the Voronoi cells (k-means clusters). The visualization uses the technique of [88].

descriptors that are to be indexed and searched to reside on the disk. This effectively reduces the memory consumption of the kd-tree and improves the scalability by an order of magnitude. Hence we term our work as *large kd-tree* to distinguish with the conventional kd-tree. Secondly, previous work treats the retrieved image matches as *finalized*, whereas we treat them as *predictions*. Since the predicted image pairs are subject to verification, they need not be highly precise. This allows us to conduct a large degree of approximation in nearest neighbor search, and trade precision for a significant improvement in efficiency.

Previous Work

Our work is closely related to [54, 73, 76], in which the authors propose to index feature descriptors for a database of images, thereby enabling 1-to-N image

matching. In [73], Lowe leverages a variant of kd-tree to index hundreds of thousands of SIFT features for efficient object recognition. In [54], Jing *et al.* construct a spill tree [72] to index the SIFT features for up to 1000 small-sized images (400 pixels in the maximum dimension) for scalable pairwise image matching. In [76], Muja and Lowe experiment with several tree structures for SIFT feature indexing, including randomized kd-tree [91] and hierarchical k-means tree [79]. They evaluate the performance of different tree structures on various datasets of up to millions of SIFT features. They also propose an automatic procedure to infer the optimal tree structure and parameter setting given a specific dataset, by sampling the parameter space in a coarse-to-fine manner while minimizing a cost function. The previous work has achieved promising results in indexing and matching databases of up to several thousand regular-sized images (about several million SIFT features). However, the previous work can hardly scale to larger databases due to the requirement for memory-based tree construction and nearest neighbor search.

Another class of algorithms approaches scalable image matching by quantizing the space of feature descriptors into a finite set of states (*visual words*) and converting each image to a bag of visual words [79, 93]. The bag-of-words model has been extensively studied in the domain of text retrieval [87]. In [93], Sivic and Zisserman introduce the model to the image domain by clustering SIFT features from all the images by k-means. The finite set of k-means centers serves as the set of visual words, based on which SIFT features are quantized and images are matched. In [79], Nister and Stewenius improve the performance of visual words by recursively clustering SIFT features to form a hierarchical k-means tree (*vocabulary tree*). Figure 7.3 illus-

trates the projections of several vocabulary trees constructed using the same set of 10 thousand SIFT features with different k values for k-means clustering (also called *branching factors*). With little overhead in tree construction and feature quantization, the vocabulary tree offers a much larger set of visual words (up to millions). The performance of vocabulary tree is evaluated on a dataset of 40000 images, and superior efficiency and quality is demonstrated for image retrieval. In [18], Agarwal *et al.* adopt the vocabulary tree as a building block in a scalable structure from motion (SfM) pipeline. Given a large internet image collection, the pipeline relies on the vocabulary tree to predict a small number of image pairs that are likely to match. Only the predicted pairs are verified by exact image matching. However, Agarwal *et al.* did not measure the prediction accuracy for vocabulary tree, probably because the ground truth for image matches is too time-consuming to collect. In our experiments, we compare the performance between large kd-tree and vocabulary tree on a controlled dataset, and demonstrate the superior performance of large kd-tree in terms of prediction accuracy.

A Review of Kd-Tree and Optimizations

In this section, we briefly review kd-tree and a few optimizations, which serve as the basis for large kd-tree. A kd-tree is a general data structure for indexing data points in k-dimensional space. The kd-tree is discussed in detail in Chapter 3; see Figure 3.4 and surrounding discussion. The data structure is proposed by Bentley [23] and refined by Friedman *et al.* for nearest neighbor search [39]. Given a set of data

points in k-dimensional space, a kd-tree is built by recursively partitioning the space into two disjoint axis-aligned subspaces until the number of data points in each subspace falls below a predefined threshold. The recursion yields a balanced binary tree, where the leaf buckets form a complete partition of the k-dimensional space and the data points. Given a query data point, the search for the nearest neighbor is reduced to a tree traversal with pruning: the leaf bucket that contains the query data point is visited first; adjacent leaf buckets are visited by backtracking; tree branches are pruned if their distances to the query data point are larger than that of the current nearest neighbor. The traversal terminates when all adjacent leaf buckets have been either visited or pruned, at which point the current nearest neighbor is verified as the true nearest neighbor for the query data point.

The performance of kd-tree is examined in low dimensions where $k \leq 6$ [39]. The expected time for nearest neighbor search is reported to be logarithmic in the number of data points indexed by the kd-tree. Unfortunately, the performance of kd-tree degrades quickly as the dimensionality of the data points grows. In high dimensions, the leaf bucket that contains a query data point has a huge number of adjacent leaf buckets, most of which must be visited to ensure a true nearest neighbor. In this case, kd-tree offers little speedup over a naive linear search.

Great effort has since been devoted to the optimization of kd-tree for nearest neighbor search in high dimensions [19, 22, 51, 91]. In this work, we leverage three optimizations to improve the performance of kd-tree: data alignment, prioritized search and approximate search.

Data Alignment. Silpa-Anan and Hartley propose to preprocess the data points by principal component analysis (PCA) [57] and align the principal axes of the data points to the coordinate axes [91]. In this way, partitioning tends to take place along the direction that exhibits the largest variance in the data points, which reduces the likelihood of distant data points falling into the same leaf bucket of the kd-tree.

Prioritized Search. Arya and Mount propose *prioritized search* [19]. Instead of visiting the leaf buckets by backtracking, prioritized search visits the leaf buckets in ascending order of distance to the query data point. This is achieved with a little overhead by maintaining a priority queue that holds all the untaken branches during the tree traversal. Prioritized search proves to be particularly useful for *approximate search* (see below), for it increases the likelihood of visiting true nearest neighbors in an early stage of the tree traversal.

Approximate Search. No known algorithm achieves sub-linear time complexity for exact nearest neighbor search in high dimensions. Therefore approximate nearest neighbor search is a promising alternative. Beis and Lowe propose Best Bin First (BBF) for approximate nearest neighbor search [22]. The algorithm follows the prioritized search scheme as in [19] but *stops early* after a predefined number of leaf buckets have been visited, enforcing a logarithmic time complexity. In [73], Lowe leverages BBF to index hundreds of thousands of SIFT features for efficient object recognition.

Other optimizations to kd-tree, such as augmenting the set of partition axes for optimal space partition [51] and building multiple kd-trees for simultaneous nearest

neighbor search [91], are also well-received in the literature. However, these optimizations alter the kd-tree structure and introduce a non-constant overhead of memory consumption. Therefore they are not adopted for large kd-tree, where memory consumption is a major constraint.

Large Kd-Tree for Approximate Image Matching

A large kd-tree is structurally identical to a conventional kd-tree. It follows the same set of rules for partitioning the k-dimensional space and processing a query data point for nearest neighbors. The performance of large kd-tree therefore benefits from the same set of optimizations: data alignment, prioritized and approximate search (see previous section).

However, the fact that a large kd-tree indexes feature descriptors from *multiple* images raises new challenges in its implementation. Specifically, as the volume of feature descriptors outgrows the capacity of the main memory, the construction and search procedures for large kd-tree must proceed with a majority of the feature descriptors residing on disk.

In the sequel, we discuss the implementation of large kd-tree in greater detail. First of all, we introduce a preprocessing of the image database, in which we re-encode the feature points by a different type of feature descriptors to optimize data alignment. Secondly, we describe the procedures for disk-based kd-tree construction and nearest neighbor search. We conclude the discussion with implementation details of large kd-tree.

Re-Encoding Feature Points

As a preprocessing to the image database, we re-encode the SIFT feature points from all images by PCA-SIFT feature descriptors [59]. A PCA-SIFT feature descriptor operates on the same input as a SIFT feature descriptor: the location, scale-space parameter and dominant orientation of a feature point (see Chapter 3). PCA-SIFT computes a feature descriptor by cropping the local image patch at the feature point (scaled according to its scale-space parameter and rotated according to its dominant orientation). It computes a gradient map over the image patch and projects the vectorized gradient map to a 36-dimensional eigenspace to form the feature descriptor. The eigenspace is trained offline on a large collection of image patches using PCA. The overhead incurred by PCA-SIFT is insignificant since its input (SIFT feature points) is readily available from the previous stage of canonical view mining.

The eigenspace projection of PCA-SIFT effectively aligns the principal axes of the resulting feature descriptors with the coordinate axes, thereby optimizing data alignment. Notice that one could also follow the procedure described by Silpa-Anan and Hartley, and apply PCA directly to SIFT feature descriptors with dimension reduction to achieve a similar effect to PCA-SIFT [91]. However, the choice of the eigenspace dimensionality would be untested. We prefer PCA-SIFT because its choice of eigenspace dimensionality (36) is supported by a systematic evaluation on a controlled dataset [59].

Finally, we group the PCA-SIFT feature descriptors from all images and store them in an external file F. Later procedures rely on F for disk-based kd-tree construction and nearest neighbor search. For optimal read/write speeds, we store the feature descriptors in binary format where each feature descriptor spans 144 bytes on disk (36 floating point numbers). Meanwhile, we keep track of the image ID for each feature descriptor in F. During nearest neighbor search, the image IDs are used to aggregate feature matches and predict image matches.

Disk-Based Kd-Tree Construction

The construction of a large kd-tree is formalized as follows: given n data points in k dimensions stored in an external file F, construct a kd-tree to index these data points *under a memory constraint* – at anytime during the kd-tree construction, at most ω data points can be loaded into the main memory. In practice, the value of ω can be easily calculated according to the desirable memory consumption and the size of each data point. Notice that the conventional procedure for kd-tree construction (Algorithm 2 of Chapter 3, Page 42) dose not work under the memory constraint (unless $\omega \geq n$), because it requires all the data points to be loaded into the main memory.

We propose a hybrid approach to large kd-tree construction (Figure 7.4): nodes on the low levels (toward the root) of the kd-tree are constructed by a disk-based procedure (Algorithm 5 below); nodes on the high levels (toward the leaves) of the kd-tree are constructed by a memory-based procedure (Algorithm 6 below). In the

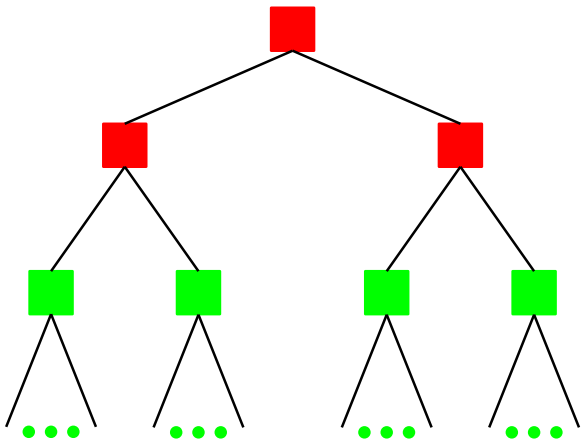


Figure 7.4: A hybrid approach to large kd-tree construction. In this illustration, we assume that $\omega = \frac{1}{4}n$. That is, at anytime during the kd-tree construction, at most $\frac{1}{4}n$ data points can be loaded into the main memory. On the first two levels of the kd-tree (red nodes), the number of data points to be indexed at each node exceeds ω. Therefore the construction of these nodes is handled by a disk-based procedure (Algorithm 5 below). As soon as the recursion of kd-tree construction reaches the third level (green nodes), the number of data points to be indexed at each node drops below ω. Therefore the construction of all the nodes on the third and subsequent levels is handled by a memory-based procedure (Algorithm 6 below).

following discussion, let us assume that the levels of a kd-tree are numbered from root to leaves and the numbering is zero-based.

First of all, let us focus on nodes at the high levels of the kd-tree. Notice that a kd-tree is a balanced binary tree. Along any path, the number of data points assigned to the nodes is progressively halved. On the lth level of the kd-tree, the number of data points assigned to each node is about $\frac{1}{2^l}n$. Therefore, even when ω is tiny compared to n, we shall soon reach a level ($\lceil log_2(\frac{n}{\omega}) \rceil$, to be exact), where all the nodes on the current and subsequent levels are each assigned $\leq \omega$ data points. From this level onward, we can safely load all the required data points into the main memory and construct the subtrees by a memory-based procedure (Algorithm 6).

Algorithm 5 Build kd-tree on the data points stored in file F and return the root.

```
procedure BuildKdTreeOnDisk(F, IDX)
    if |IDX| ≤ ω then
        P ⇐ Load the indexed data points from F
        v ⇐ BuildKdTreeInMemory(P, IDX)                    ▷ Algorithm 6.
        free P
        return v
    else
        P ⇐ Sample and load ω of the data points from F
        axis ⇐ Select the partition axis along which P exhibits the largest variance
        threshold ⇐ Compute the median projection of the indexed data points in
                    F on axis
        IDX_LE, IDX_GT ⇐ Split the indexed data points by threshold
        v ⇐ new InternalNode
        v.axis ⇐ axis
        v.threshold ⇐ threshold
        v.leftChild ⇐ BuildKdTreeOnDisk(F, IDX_LE)
        v.rightChild ⇐ BuildKdTreeOnDisk(F, IDX_GT)
        return v
    end if
end procedure
```

Algorithm 6 Build kd-tree on data points P and return the root.

```
procedure BuildKdTreeInMemory(P, IDX)
    if |IDX| < φ then
        v ⇐ new LeafBucket
        v.indices ⇐ IDX
        return v
    else
        axis ⇐ Select a partition axis along which P exhibits the largest variance
        threshold ⇐ Compute the median projection of P on axis
        P_LE, P_GT ⇐ Split P by threshold
        IDX_LE, IDX_GT ⇐ Split IDX according to P_LE, P_GT
        v ⇐ new InternalNode
        v.axis ⇐ axis
        v.threshold ⇐ threshold
        v.leftChild ⇐ BuildKdTreeInMemory(P_LE, IDX_LE)
        v.rightChild ⇐ BuildKdTreeInMemory(P_GT, IDX_GT)
        return v
    end if
end procedure
```

Algorithm 6 assumes a similar form to the conventional procedure for kd-tree construction (Algorithm 2), except that at the leaf level, the leaf buckets contain the *indices* of the data points on disk, instead of the data points themselves. This change is necessary for a large kd-tree, since upon construction, the leaf buckets will form a complete partition of all n data points, which may not fit into the main memory.

Now let us focus on nodes at the low levels (0 through $\lceil log_2(\frac{n}{\omega})\rceil - 1$) of the kd-tree. Upon close examination of Algorithm 2, we can see that the construction of each node requires the data points to be in memory for two reasons: (1) it relies on the data points to select the partition axis – the coordinate axis along which the data points exhibit the largest variance; (2) it relies on the data points to compute the cutoff value – the median projection of the data points on the partition axis. The partition axis and the cutoff value together define the split for all the data points at the current node. Under the memory constraint, we approximate the selection of partition axis by randomly sampling a subset of ω data points and loading the sampled data points into main memory. We select the partition axis as the coordinate axis along which the *sampled data points* exhibit the largest variance. After the partition axis is selected, we scan the external file F to compute the projections of the data points on the partition axis, and compute the median projection as the cutoff value. Notice that the projections are computed for *all* the assigned data points at the current node, not just for the sampled ones in memory. This is to ensure an equal split of all the assigned data points at the current node; otherwise the kd-tree would not be balanced. During the computation of the cutoff value, only the *projections* of

the data points are stored in memory. The data points themselves are not loaded. Therefore the memory constraint is not violated.

The disk-based procedure (Algorithm 5) is the entry point of large kd-tree construction. Since the data points are never fully loaded into main memory, we keep track of the assigned data points at each node by a list of indices to the data points on disk. At the root node, the list of indices is initialized to $(1, \cdots, n)$. During the recursive kd-tree construction, the list of indices is recursively split in half and passed to child nodes. As soon as the recursion reaches level $\lceil log_2(\frac{n}{\omega}) \rceil$, kd-tree construction enters the memory-based procedure (Algorithm 6), in which all the assigned data points are loaded into the main memory to construct the entire subtree at the current node. The recursion proceeds in a depth-first manner. After the entire subtree is constructed at the current node, the memory allocated for the assigned data points is immediately freed, leaving no effect on the construction of other branches (see Algorithm 5).

Notice that the hybrid approach introduces certain approximations to large kd-tree construction: during the disk-based procedure, the partition axis is determined by a *subset* of the assigned data points, and therefore may not be optimal. In practice, however, the degree of approximation is minimal. We study the approximation of large kd-tree construction on the `Rome` collection (11959 images, $n = 16,393,211$ feature descriptors). In this study, we define the memory constraint ω as a function of n: $\omega = \epsilon n$, where ϵ varies over $\{\frac{1}{10}, \frac{2}{10}, \cdots, 1\}$. At the high extreme of $\epsilon = 1$, the hybrid approach is equivalent to the memory-based procedure with zero approximation. As ϵ decreases, we would expect a higher degree of approximation in the resulting large

kd-tree. However, it turned out that all 10 large kd-trees were structurally *identical*. That is, the ever-tighter memory constraint had no effect on the selection of partition axis at *any* node in the large kd-tree. Not until we reduced ω to $\frac{1}{100}n$ did we observe a difference in the resulting large kd-tree. There are two reasons behind the minimal degree of approximation. First of all, the disk-based procedure is only applied to construct the first several levels of the kd-tree (levels 0 through $\lceil log_2(\frac{n}{\omega})\rceil - 1$). Even when ω is as tiny as $\frac{1}{16}n$, approximation can only take place on the first 4 levels of the kd-tree (15 nodes). Secondly, we have already projected all the data points to a low-dimensional eigenspace. Therefore, the data points exhibit drastically different variances along different coordinate axes. Even though this property is diminished by the recursive split of the data points, it generally holds true for the first several levels of the kd-tree (where approximation takes place). Therefore a subset of the data points usually suffices to detect the true partition axis at these nodes.

The runtime of the hybrid approach is satisfactory. Compared to the conventional kd-tree construction, the only additional cost of the hybrid approach comes from the disk-based procedure, where the external file F needs to be read at each node on levels 0 through $\lceil log_2(\frac{n}{\omega})\rceil$. In practice, this additional cost is insignificant, because (1) disk reads only happen at a small number of nodes ($2^{\lceil log_2(n/\omega)\rceil+1} - 1$, to be exact), and (2) each disk read involves scanning and parsing a binary file, which is efficient. In the above study on the `Rome` collection, each disk scan of the external file F (about 2.4GB) takes about 50 seconds. Even when ω is as low as $\frac{1}{10}n$, all the 31 disk reads on the first 5 levels of the large kd-tree cost only about 25 minutes of

additional runtime. Considering that the large kd-tree is constructed only once but is queried by tens of millions of feature descriptors, the additional cost is negligible.

Disk-Based Nearest Neighbor Search

Before we dive into the details of disk-based nearest neighbor search on large kd-tree, let us briefly overview the procedure for 1-to-N image matching. Given a query image, we obtain all the PCA-SIFT features from the image, and issue each feature as a query to the large kd-tree to retrieve nearest neighbors, which correspond to candidate feature matches to database images. We follow the suggestion of Ke and Sukthankar [59], and verify the robustness of each candidate feature match by a *distance test*: a candidate feature match is robust if and only if the Euclidean distance between the two features is below a predefined threshold $\theta = 3000$. After all the features from the query image have been processed for nearest neighbor search and the distance test, we have collected a set of feature matches between the query image and the database images. Finally, we group the feature matches by distinct database image. For each database image, the more feature matches it contains, the more likely it is that it matches to the query image.

The core operation of 1-to-N image matching is the nearest neighbor search on large kd-tree. The conventional procedure for nearest neighbor search on kd-tree (optimized by prioritized and approximate search) can be summarized as follows: given a query feature $\boldsymbol{q}$, we visit the leaf buckets of the kd-tree in ascending order of distance to $\boldsymbol{q}$. As we visit a leaf bucket, we examine all the database features in the

leaf bucket and compute their distances to $\boldsymbol{q}$. We maintain a list of database features $\mathcal{M}$ whose distances to $\boldsymbol{q}$ fall below the predefined threshold $\theta = 3000$. The search of nearest neighbors terminates after γ database features have been visited. Upon termination, we report $\mathcal{M}$ as the list of matching features to $\boldsymbol{q}$. Obviously, a higher value of γ leads to a higher recall of feature matches. At the high extreme of $\gamma = n$, all the database features would be processed for the distance test, achieving a 100% recall of feature matches. However, the time complexity would also grow to $\mathcal{O}(n)$, offering no advantage over a naive linear search. Fortunately, due to the effective space partition of kd-tree and prioritized search scheme, the true nearest neighbors of $\boldsymbol{q}$ are likely to be visited during the early stages of tree traversal. Therefore a low value of γ often suffices to achieve a reasonable recall of feature matches. In this dissertation, we empirically set $\gamma = 200$ for all the image collections. Below this threshold, we observe a fast improvement of recall as we increase γ; above this threshold, the improvement slows down, and further increase of γ is less cost-effective. By visiting γ database features per query, the time complexity of nearest neighbor search is reduced to $\mathcal{O}(log(n))$, with a constant factor proportional to γ.

The conventional procedure for nearest neighbor search requires all the database features to reside in main memory, so that random access to any database feature for distance computation is fast (in the order of 10^{-4}ms). In the context of large kd-tree, the volume of database features may outgrow the capacity of the main memory. Therefore we need to conduct nearest neighbor search with the database features residing on disk. If we naively follow the conventional procedure, but replace the memory access to database features by disk access, the runtime of nearest neighbor

search would suffer tremendously. A random disk access is very expensive, in the order of 10ms. When $\gamma = 200$, processing a query feature would incur about 2 seconds just on disk I/O; processing a query image with about 1500 features would take about 50 minutes, which is even slower than matching the query image to the entire database of images by exact image matching.

In [60], Ke *et al.* have faced a similar situation of massive random access to disk data. They present a simple but effective strategy to remove the bottleneck of disk I/O. The key operation is to postpone individual random disk accesses, and group them into one *sequential* disk access. In this dissertation, we employ the same strategy. We process not a single query image, but a batch of M query images at a time. For each query image, we process each query feature following the conventional procedure to traverse the large kd-tree and retrieve a list of $\gamma = 200$ database features as candidate feature matches. However, during this process, we do not perform the distance test on any of the candidate feature matches. Since no distance computation is required, no random disk access is triggered. We postpone the distance test until all the features from all M query images have been processed. By this time we have accumulated a large number of candidate feature matches:

$$\begin{array}{rcl} QF_1 & \Rightarrow & (DF_{11}, \cdots, DF_{1\gamma}) \\ QF_2 & \Rightarrow & (DF_{21}, \cdots, DF_{2\gamma}) \\ & \cdots & \\ QF_i & \Rightarrow & (DF_{i1}, \cdots, DF_{i\gamma}) \\ & \cdots & \end{array}$$

For each query feature QF_i, the retrieved database features $(DF_{i1}, \cdots, DF_{i\gamma})$ are stored in a linked list. In the next step, we reorganize the data structure into an

inverted index, where the retrieved database features serve as the index, each followed by the corresponding query features stored in a linked list:

$$\begin{array}{rcl} DF'_1 & \Rightarrow & (QF'_{11}, \cdots, QF'_{1m_1}) \\ DF'_2 & \Rightarrow & (QF'_{21}, \cdots, QF'_{2m_2}) \\ & \cdots & \\ DF'_i & \Rightarrow & (QF'_{i1}, \cdots, QF'_{im_i}) \\ & \cdots & \end{array}$$

After the reorganization, $\{DF'\}$ contains the indices of all the database features that we need to access on disk. Finally, we sort $\{DF'\}$ in ascending order and thereby access the database features on disk *sequentially*. When we access DF'_i, we load the referenced database feature $\boldsymbol{p}_i$ into main memory and perform the distance test between DF'_i and all the corresponding query features in $(QF'_{i1}, \cdots, QF'_{im_i})$ and retain robust feature matches among them. We free the memory allocated for $\boldsymbol{p}_i$ immediately after the distance test has been performed for all the corresponding feature matches, before we access DF'_{i+1}. Therefore the sequential access of database features imposes practically zero overhead in memory consumption.

By grouping a large number of random disk accesses into one sequential disk access, we effectively minimize the amount of seek time between consecutive reads, which is the bottleneck in disk access. In practice, the amount of speedup is tremendous. In this dissertation, we empirically set $M = 100$. In the `Rome` collection, an average image consists of about 1500 features. After a batch of $M = 100$ query images have been processed, we have accumulated about $100 \times 1500 \times 200 = 30$ million candidate feature matches (where $200 = \gamma$). A sequential scan of the external file F (about 2.4GB) takes about 50 seconds. Therefore the average access time for a database feature on disk drops from about 10ms by individual random disk access

to (50 seconds)/(30 million) $\approx 2 \times 10^{-3}$ms by grouped sequential disk access. As a result, the average processing time for a query image drops from about 50 minutes to about 1.6 seconds to match to the entire database of 11959 images!

Another way to understand this strategy is that, since disk access is expensive, we would like to perform the distance test on as many candidate feature matches as possible on one disk access. In the conventional procedure for nearest neighbor search, we could only verify one candidate feature match on one disk access. Thus the efficiency is extremely poor. In the new strategy, when $M = 100$, we can verify about 30 million candidate feature matches on one disk access. Thus the efficiency is greatly improved. Notice that a larger value of M would further reduce the average access time for a database feature, and thereby further reduce the average processing time for a query image. However, it would also increase the memory consumption of query processing, because we need to store a larger set of query features and a larger inverted index in the main memory. When $M = 100$, the bottleneck of query processing has already shifted from disk I/O (about 0.5 seconds per image) to in-memory operations of kd-tree traversal and numerical computation (about 1.1 seconds per image). Therefore we do not increase M any further.

After we have performed the distance test for all the candidate feature matches, we have collected a set of robust feature matches among the M query images and the database images. We then look up the image IDs corresponding to the matching features, and group the feature matches by distinct (*query*, *database*) image pair. In the context of pairwise image matching, we issue each database image as a query image to the large kd-tree. Therefore after we have processed each query image, we

have two sets of matching features $\mathcal{M}$ and $\mathcal{M}'$ for each image pair (I, J), one with I as the query image, and the other with J as the query image. We merge $\mathcal{M}$ and $\mathcal{M}'$ to achieve a further (marginal) improvement in the number of feature matches between the two images. Finally, we determine the likelihood of the two images I and J being a match based on the number of merged feature matches $|\mathcal{M} \cup \mathcal{M}'|$.

Notice that we do not perform geometric verification on the feature matches using RANSAC on the fundamental matrix as we have done during exact image matching (Chapter 3). This is because the feature matches extracted during 1-to-N image matching are sparse. Some true image matches have too few feature matches even to be qualified for geometric verification (the 8-point algorithm requires at least 8 feature matches to start with). Instead, we use the number of raw feature matches to predict if two images are likely to match. The lack of geometric verification is reasonable, since the purpose of this stage is to generate *predictions* of image matches, which are still subject to *verification* by exact image matching. For this purpose, we shall see that the feature matches, albeit sparse, can indeed predict with a high recall which images are likely to match.

Implementation of Large Kd-Tree

Finally, we present the implementation details of large kd-tree. Leveraging the fact that a kd-tree is a balanced binary tree, we implement the kd-tree in an array representation: the nodes in the kd-tree are mapped in a depth-first manner and stored in an array bounded by $2n$ elements, where n is the number of data points

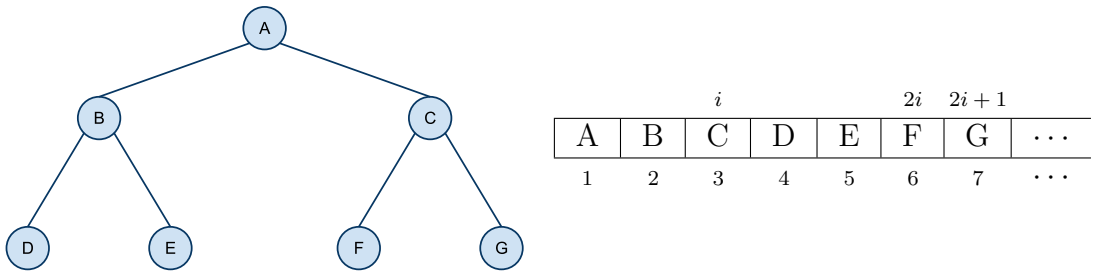


Figure 7.5: An array representation of kd-tree. The nodes in the kd-tree are mapped in a depth-first manner and stored in an array; the two child nodes of node i are therefore located at $2i$ and $2i + 1$ in the array.

indexed by the kd-tree; the two child nodes of node i are therefore located at $2i$ and $2i+1$ in the array (Figure 7.5). By not storing the pointers among the nodes, we save a significant amount of memory (about $2n$ pointers $\times$ 4 bytes per pointer $= 8n$ bytes). At each internal node, we store the partition axis (1 byte for an integer ranging from 1 to 36) and the cutoff value (4 bytes for a floating point number). At each leaf node, we store the indices to the data points (4 bytes per index). Both the number of internal nodes and the number of leaf nodes are bounded by n. In total, the kd-tree structure takes about $(1+4)n$ bytes to store the internal nodes and about $4n$ bytes to store the leaf nodes, totaling about $9n$ bytes of memory. On the other hand, the size of the data points indexed by the kd-tree is $144n$ bytes (36 floating point numbers per PCA-SIFT feature descriptor)! By allowing the data points to reside on disk, large kd-tree achieves a significant reduction in memory consumption (from $(144 + 9)n$ to $9n$). For example, within 1GB of memory, a large kd-tree can index over 100 million data points (about 80 thousand regular-sized images), while a conventional kd-tree, which requires all the data points to reside in the main memory, can only index

Table 7.2: Statistics of the experimental dataset. For each image collection, we show the number of images in the collection, the number of SIFT features extracted from the images, and the size of the re-encoded PCA-SIFT feature descriptors stored on disk.

Collection	♯ images	♯ features	size of F
`Dubrovnik`	9350	12,236,285	1.8GB
`Paris`	11997	16,257,199	2.3GB
`Rome`	11959	16,393,211	2.4GB
`Washington DC`	11991	14,242,285	2.1GB
`Yosemite`	5756	8,978,015	1.3GB

about 7 million data points (less than 5 thousand regular-sized images). Hence the scalability is greatly improved.

Experiment-I: Approximate Image Matching

In this section, we evaluate the performance of large kd-tree for approximate image matching. We compare the performance of large kd-tree and vocabulary tree in terms of precision and recall of image matches, and demonstrate the superior performance of large kd-tree.

Experimental Setup

Experimental dataset. We conduct Experiment-I on the same dataset as canonical view evaluation in Chapter 5. The dataset consists of five image collections of different sites: Dubrovnik, Paris, Rome, Washington DC and Yosemite National Park. Each image collection contains between 5000 and 12000 images. On average, each image generates about 1500 SIFT features; each image collection generates about

15 million features. On average, the re-encoded PCA-SIFT feature descriptors for each image collection spans about 2GB on disk. Detailed statistics can be found in Table 7.2.

Approximate image matching with large kd-tree. For each image collection, we follow the procedure described in the previous section to conduct approximate image matching. During large kd-tree construction, we set the memory constraint $\omega = 0.1n$, where n is the number of feature descriptors in the image collection. During nearest neighbor search, we set $\gamma = 200$ (number of database features to visit per query) and $M = 100$ (number of query images to process in a batch). Upon output, each pair of images is assigned a likelihood of being a match, which is determined by the number of feature matches between the two images.

Approximate image matching with vocabulary tree. In order to put large kd-tree into context, we also conduct approximate image matching using the widely adopted vocabulary tree (unfamiliar readers are referred to [79] for a detailed description of vocabulary tree). For each image collection, we group the SIFT feature descriptors from all images and build a hierarchical k-means tree (vocabulary tree) with a branching factor of 10 and a depth of 6, and thereby obtain a codebook of $10^6 = 1$ million visual words. Each feature descriptor is quantized into a visual word by traversing down the vocabulary tree. Each image is converted to a bag of visual words, represented by a (sparse) vector in 1 million dimensions. Finally, images are pairwise matched, and the distance between two images is defined by the L^1 distance between their bag-of-words vectors, with each dimension weighted by the entropy

of the corresponding visual word. Upon output, each pair of images is assigned a distance score (negation of similarity).

Ground truth and measurements. For each image collection, we obtained the ground-truth image matches by exact image matching followed by geometric verification (Chapter 3). With ground truth in place, we evaluate the performance of approximate image matching in terms of precision and recall of image matches. Loosely speaking, precision measures how many image matches predicted by approximate image matching are correct according to the ground truth, and recall measures how many image matches in the ground truth are predicted by approximate image matching. Mathematical definitions are provided below:

$$precision = \frac{\sharp \text{ correct matches by approx.}}{\sharp \text{ matches by approx.}}, \tag{7.1}$$

$$recall = \frac{\sharp \text{ correct matches by approx.}}{\sharp \text{ matches by ground truth}}. \tag{7.2}$$

Results

First, we compare the performance of large kd-tree and vocabulary tree in terms of precision and recall of image matches. The performance of both algorithms is shown in Figure 7.6. Notice that the output from both algorithms is a set of scalar values, each measuring the likelihood of two images being a match. In order to convert a likelihood score to a concrete match/no-match prediction, we need to

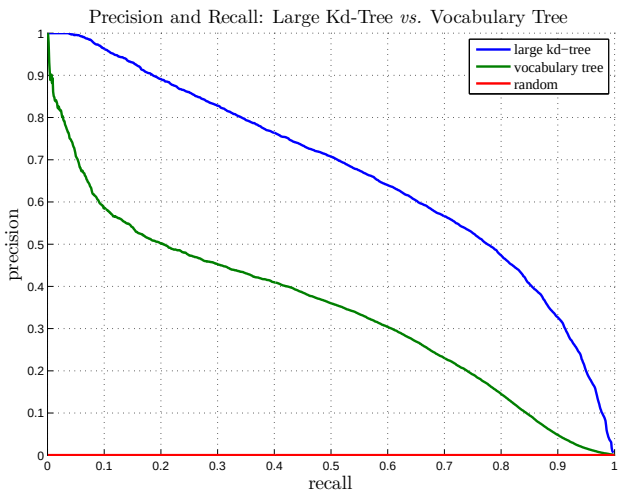


Figure 7.6: Precision and recall of approximate image matching across a spectrum of matching thresholds. The performance is compared among large kd-tree, vocabulary tree and the random baseline. Large kd-tree outperforms vocabulary tree by a large margin. Both algorithms achieve significant improvements over the random baseline.

impose a threshold ϑ on the likelihood scores:

$$match(I, J) = \begin{cases} \text{true} & likelihood(I, J) \geq \vartheta, \\ \text{false} & \text{otherwise.} \end{cases} \tag{7.3}$$

At a fixed threshold ϑ, we can obtain a set of predictions from both algorithms and thereby compute their precision and recall of image matches. We vary ϑ over all possible values of likelihood scores and generate a complete precision-recall curve to compare the performance between the two algorithms (Figure 7.6).

It can be observed that, while both algorithms achieve significant improvements over the random baseline (where image matches are predicted at random), large kd-tree outperforms vocabulary tree by a large margin. At almost all recall lev-

Table 7.3: Efficiency of pairwise image matching and recall of feature matches. Statistics are compared between exact image matching and approximate image matching (using large kd-tree) on the `Rome` collection. For each method, the corresponding column shows the runtime, the total number of feature pairs checked for distance test, and the total number of feature pairs passing the distance test (output feature matches). See text for analysis.

	Exact matching	Approximate matching
runtime	2212 hours	5.62 hours
pairs of features checked	21451 billion	4 billion
pairs of features matched	75 million	8 million

els, the precision achieved by large kd-tree consistently advances that by vocabulary tree by about 30%.

The most significant observation from the precision-recall curve is that, even at a high recall of image matches (such as 90%), large kd-tree is still able to maintain a reasonable precision (32.65%). At first sight, this precision may seem low. However, given the extreme sparsity of image matches in internet image collections (about 0.03%), this precision is already impressive, up to 1000 times better than chance.

Next, we present more statistics on feature-level matches. Table 7.3 shows the statistics of pairwise image matching on the `Rome` collection using approximate image matching (large kd-tree) and exact image matching. It can be observed that approximate image matching misses a large fraction (about 90%) of the feature matches that are extracted by exact image matching. On the other hand, approximate image matching is also about 400 times faster, because it only checks a tiny fraction (about 0.02%) of the feature pairs that are checked by exact image matching. Fortunately, the sparsity of feature matches of approximate image matching does not pose much problem in predicting image matches, because true image matches often have hun-

dreds of feature matches, while false ones barely have more than a few. Therefore we can afford to lose a majority of the feature matches while still being able to differentiate true image matches from false ones. As long as the correct image matches are predicted, the verification stage can extract dense feature matches among them. Therefore the final output of the prediction-verification scheme can achieve a high recall on both the image and the feature levels.

Experiment-II: Canonical View Mining with Approximate Image Matching

In this section, we incorporate the prediction-verification scheme for pairwise image matching into the pipeline of canonical view mining. We demonstrate that the prediction-verification scheme speeds up the pipeline by two orders of magnitude with low impact on the resulting canonical views.

Experimental setup

Canonical view mining with approximate image matching. We conduct canonical view mining on all five image collections in Experiment-I. This time, we replace the conventional exact image matching by the prediction-verification scheme. Since canonical view mining relies on the density of the image similarity graph, a high recall of image matches is desirable. We empirically set $\vartheta = 6$, which yields an average of 33% precision and 90% recall of image matches (see Experiment-I).

Ground truth and measurements. We have collected ground-truth canonical view ranking for each image collection in Chapter 5. In order to study the impact of

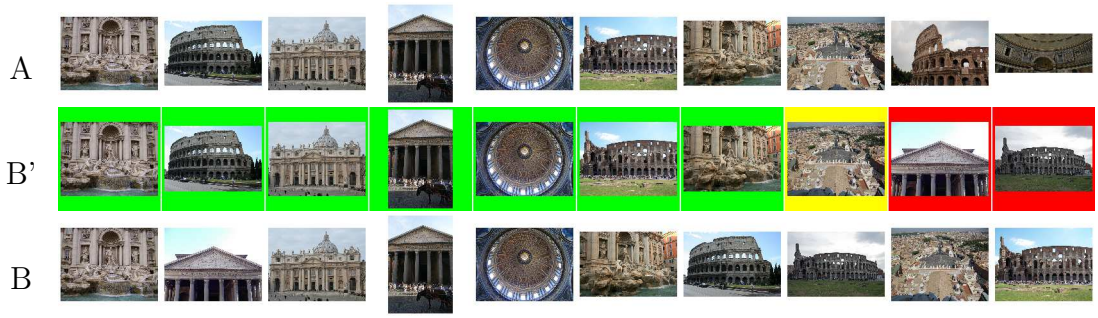


Figure 7.7: An illustration of the maximum bipartite matching between two sets of images of the `Rome` collection. Let the two sets of images be labeled by A (top row) and B (bottom row). Maximum bipartite matching computes a one-to-one mapping between the images of A and B that maximizes the total number of inlier SIFT matches. For a better illustration, images of B are reordered to B' (central row) such that the paired images between A and B are in the same columns. The reordered images in B' are shown with a green border if they are the same images as A, with a yellow border if they are different images but share at least 16 inlier SIFT matches to A (near-duplicate views), or with a red border if they are different images with less than 16 inlier SIFT matches to A. In the measurement of recall, we count both identical views and near-duplicate views. In this example, $recall_A(B) = \frac{8}{10}$.

approximate image matching on the resulting canonical views, we need a measurement to quantify the difference between two sets of canonical views, one generated with exact image matching (ground truth), and the other generated with approximate image matching. Inspired by the work of Jing *et al.* [55], we compare two sets of canonical views using maximum bipartite matching [105]. Given two sets of canonical views of equal size $\mathcal{C}$ and $\mathcal{C}'$, we form a bipartite graph $G(\mathcal{C} \cup \mathcal{C}', E)$ where edges $E \in \mathcal{C} \times \mathcal{C}'$ are weighted by the number of inlier SIFT matches between the corresponding images. The maximum bipartite matching computes a 1-to-1 mapping between the images in $\mathcal{C}$ and $\mathcal{C}'$ that maximizes the total number of inliers SIFT matches. Afterwards, we measure $\mathcal{C}'$ by its recall of images in $\mathcal{C}$. We count both identical views and near-duplicate views (defined as different images sharing at least

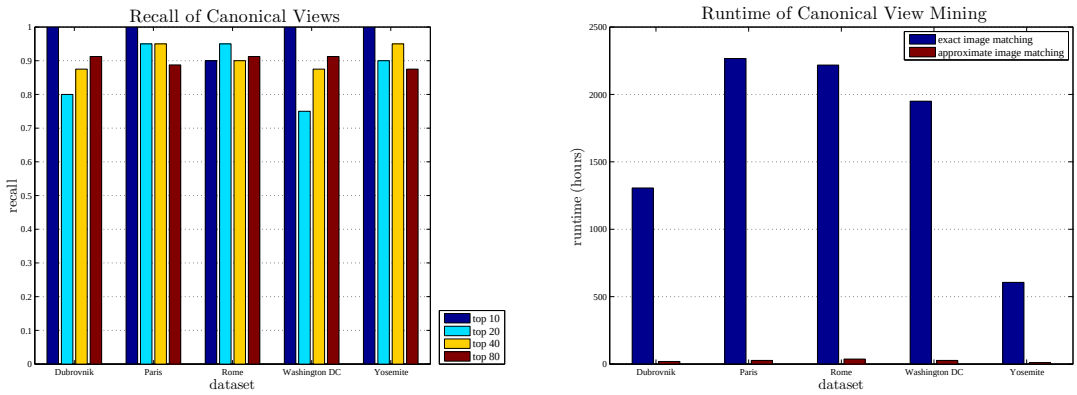


Figure 7.8: Impacts of approximate image matching on canonical view mining. The left figure shows the recall of the top canonical views computed with approximate image matching. The average recall of the top 10/20/40/80 canonical views across all image collections is 98%/87%/91%/90% respectively. The right figure compares the runtime between exact image matching and the prediction-verification scheme for canonical view mining. The latter speeds up the pipeline by two orders of magnitude. On average, the runtime of canonical view mining is reduced by about 99%, from about 2000 CPU hours to about 20 CPU hours per large collection of 10000+ images.

16 inlier SIFT matches) that are mapped by maximum bipartite matching toward the recall of $\mathcal{C}'$. This measurement is illustrated in Figure 7.7.

Results

The impact of approximate image matching on the resulting canonical views is shown in Figure 7.8. For each image collection, we compute the recall of ground truth in the top $k \in \{10, 20, 40, 80\}$ canonical views. The average recall across all image collections varies over $\{90\%, 87\%, 91\%, 90\%\}$ respectively. That is, within the top $k \in \{10, 20, 40, 80\}$ canonical views computed with approximate image matching, about $0.9k$ of the images are either identical or near-duplicate views to the ground truth. In Figure 7.9, we present qualitative results of the top 10 canonical views

Table 7.4: Comparison of runtime (in hours) on the `Rome` collection. The runtime of canonical view mining is divided into individual steps and shown for the conventional pipeline (with exact pairwise image matching) and the scalable pipeline (with the prediction-verification scheme for pairwise image matching). See text for analysis.

	Approximate matching			Exact matching		Rest	Total
	encode	tree	query	match	ransac		
conventional				2212	0.68	5.50	2218.18
predict-verify	4.98	0.55	5.62	1.97	0.61	5.50	19.23

computed under approximate image matching for each image collection, along with the results of maximum bipartite matching to the ground truth. Only one image in the `Rome` collection is not found in the top 10 ground-truth canonical views (it ranks 12th in the ground truth). All the other canonical views computed with approximate image matching are either identical or near-duplicate views to the ground-truth ones.

On the other hand, the speedup brought about by approximate image matching is significant. On average, the runtime of canonical view mining (mostly pairwise image matching) is reduced by about 99%, from about 2000 CPU hours to about 20 CPU hours per large collection of 10000+ images. In Table 7.4, we show the detailed runtime of canonical view mining on the `Rome` collection. The runtime distributions for other image collections are similar. In the scalable pipeline (with the prediction-verification scheme for pairwise image matching), the prediction stage takes 4.98 CPU hours to re-encode all the SIFT features by PCA-SIFT feature descriptors, 0.55 CPU hours to build the large kd-tree on about 16 million features from all images, and 5.62 CPU hours to query each of the 11959 images against the large kd-tree and predict image matches. In total, the runtime of the prediction stage is about 11.15 CPU

hours. This stage generates 60715 predictions of image matches (with 20036 true positives and 40679 false positives). The 60715 predictions are subject to verification by exact image matching. Notice that the set of predictions is much smaller than the full set of $N(N-1) = 71,502,861$ image pairs. As a result, the verification stage proceeds much faster, incurring only 1.97 CPU hours for SIFT feature matching and 0.61 CPU hours for geometric verification. The rest of the pipeline (SIFT feature extraction, representative view and canonical view ranking) takes about 5.5 CPU hours. In total, the runtime for canonical view mining drops from over 2200 CPU hours using the conventional pipeline to 19.23 CPU hours using the scalable pipeline. Meanwhile, the scalable pipeline still maintains 90% recall of image matches (20036 *vs.* 22262). Thus the impact on the resulting canonical views is insignificant.

Summary

A prediction-verification scheme has been presented for scalable image matching. During the prediction stage, image matches are predicted by approximate image matching. During the verification stage, the correctness of each prediction is verified by exact image matching. The principal novelty is the use of large kd-tree for approximate image matching. We proposed disk-based kd-tree construction and nearest neighbor search, which enable large kd-tree to scale to databases of tens of thousands of images. We evaluated approximate image matching both separately and integrated with canonical view mining, and demonstrated its superior performance in comparison to previous work.

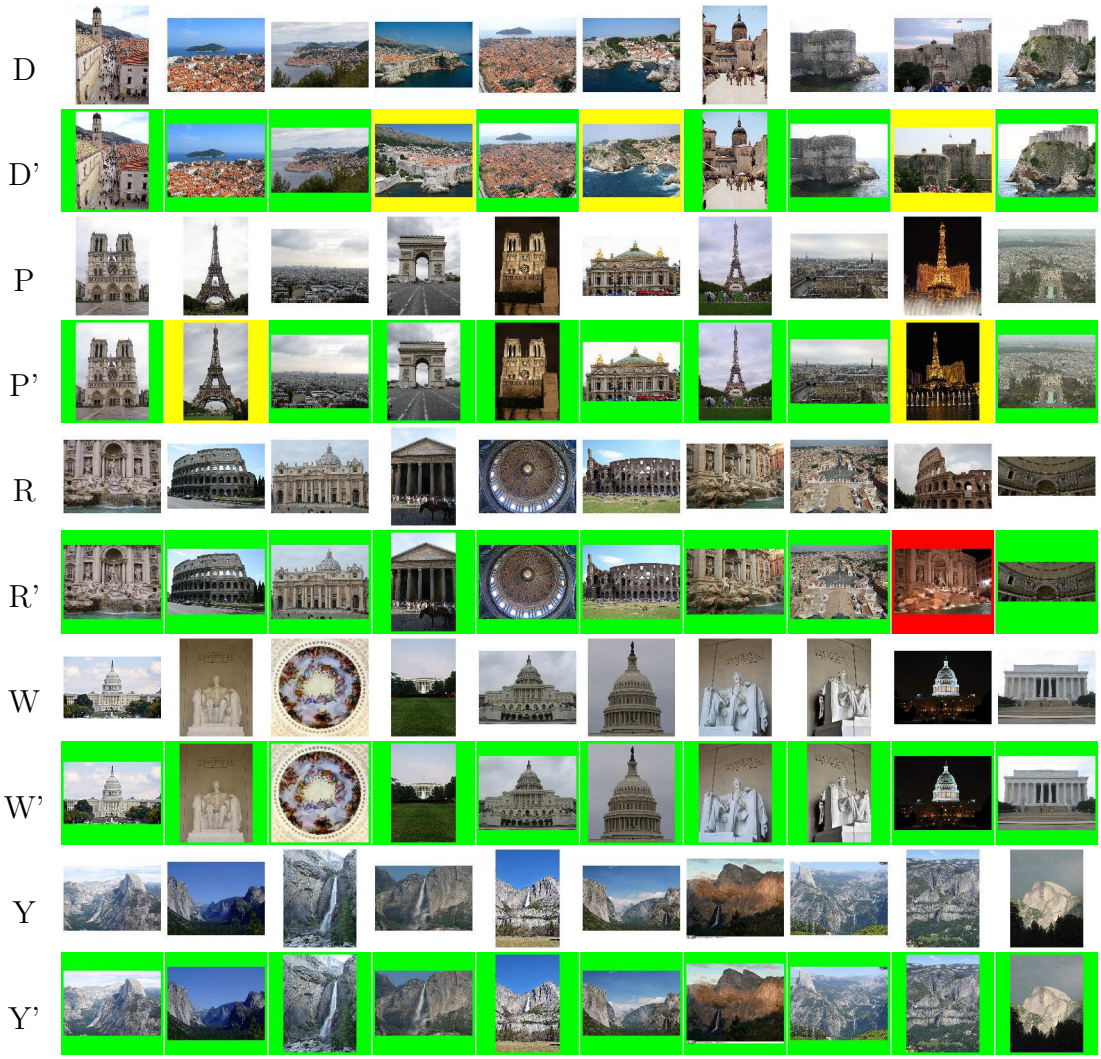


Figure 7.9: Maximum bipartite matching between the top 10 canonical views computed with exact image matching (labeled by X) and the prediction-verification scheme (labeled by X'). The labels are D for `Dubrovnik`, P for `Paris`, R for `Rome`, W for `Washington DC` and Y for `Yosemite`. The two sets of canonical views in each image collection are paired by maximum bipartite matching. Paired canonical views in X' are reordered to be in the same columns as X. The reordered canonical views in X' are shown with a green border if they are the same images as X, with a yellow border if they are different images but share at least 16 inlier SIFT matches to X, or with a red border if they are different images with less than 16 inlier SIFT matches to X. This figure is best viewed in color.

CHAPTER 8

CONCLUSIONS AND FUTURE WORK

In this dissertation, we develop a fully automatic pipeline for canonical view mining from internet image collections. We evaluate the quality of the canonical views and demonstrate their application to large-scale image browsing and efficient object recognition. Finally, we analyze the scalability of the pipeline, and propose an approximate algorithm that effectively removes the scalability bottleneck with low impact on the resulting canonical views. In this chapter, we summarize the contributions of this dissertation, and explore possibilities for future work.

Summary of Contributions

In Chapter 4, we developed the pipeline for canonical view mining. The pipeline relies on the wisdom of crowds to automatically infer representative views and canonical views. It is completely data-driven and requires no user input. Specifically, it does not require parameter tuning across different input datasets, which would be a painstaking process given the enormous amount of input datasets that can be obtained from internet image collections. Moreover, the pipeline does not require the number of canonical views to be known *a priori*. Instead, it computes a ranking of canonical views such that the top-ranked images are both representative and diverse, thereby approximating canonical views at a range of granularities. Once the ranking

is computed offline, any number of canonical views for any subset (including the full set) of the image collection can be retrieved in real-time. We also discussed the incorporation of the pipeline for canonical view mining with current image search engines, so that canonical views can be retrieved from image search results in *real-time* and presented to the user for enhanced image browsing.

In Chapter 5, we evaluated the quality of canonical views. Besides showing qualitative results of canonical views, we introduce three quantitative measurements to evaluate the canonical views by quantifying the amount of noise, redundancy, and summarization power for the top-ranked canonical views. Based on the quantitative measurements, we evaluated the pipeline for canonical view mining on a variety of datasets, and demonstrate that the proposed pipeline compares favorably to several other methods, including the search engines of Flickr [4], Google Images [9] and the previous work of [53, 92].

In Chapter 6, we extended the applications of canonical views beyond image browsing, to non-parametric object recognition (object recognition based on nearest neighbor search instead of parametric modeling for object classes). By removing noise and redundancy from the database, we expect the set of canonical views to compress the database into a compact representation while still preserving most of the representative power for the purpose of object recognition. We validated this hypothesis on the place recognition problem, in which we estimate the geographic location of a query image by scene matching to a large database of images of known locations. By leveraging canonical views, we observe a significant improvement in the efficiency of query processing with minimal loss in the success rate of place recognition.

In Chapter 7, we analyzed the scalability of the pipeline for canonical view mining. We singled out the stage of pairwise image matching as the scalability bottleneck. To this end, we presented a prediction-verification scheme for pairwise image matching. During the prediction stage, image matches are predicted by approximate image matching. During the verification stage, the correctness of each prediction is verified by exact image matching. The principal novelty is the use of large kd-tree for approximate image matching. We proposed disk-based kd-tree construction and nearest neighbor search, which enable large kd-tree to scale to databases of tens of thousands of images. We evaluated approximate image matching both separately and integrated with canonical view mining, and demonstrated its superior performance in comparison to previous work.

Future Work

The work presented in this dissertation is just a first step toward the effective organization and presentation of internet image collections. In this section, we explore several possibilities for future work.

Web-scale image matching. Scalability is still a significant problem in canonical view mining. In Chapter 7, we presented a prediction-verification scheme to improve the scalability of image matching by an order of magnitude, from thousands to tens of thousands of images. However, for a popular keyword, we can easily retrieve a collection of images that is two or three orders of magnitudes larger than the current scale. For example, by searching for the keyword `Rome` on Flickr, one gains

nearly three million images in the search results. How do we scale image matching (and thus canonical view mining) to the scale of the web? Of course, parallelization would play a significant role in this system. Nonetheless, progress must be made in the core algorithm of image matching; otherwise the quadratic complexity of pairwise image matching simply cannot scale to huge numbers of images.

Image matching for non-rigid objects. Another major constraint of the current pipeline for canonical view mining is that, we rely on SIFT feature matching to establish the similarity among images, which can only handle objects of rigid appearances. A promising extension to the current pipeline would be an enhanced similarity metric that is able to match non-rigid objects (such as animals and plants) across different views. One possible direction along this thread would be to relax the global geometric constraint and instead detect subsets of features that are consistent with local geometric constraints (such as [24]).

Visual quality assessment. Visual quality provides an orthogonal dimension for the selection of canonical views. Ideally, if multiple images exist for one canonical view (which is almost always true considering the volume of internet images), we would like to present the one with the highest visual quality (or at least bias the ranking of canonical views by visual quality). In the literature of computer vision, there have been several attempts to assess the visual quality of images based on visual features and statistical models [31, 61]. Nonetheless, there is still sufficient room for improvement. On the other hand, the timing for this research is ideal: photo contest websites such as DPChallenge [2] and Photo.net [13] not only host huge amounts of photos of various visual qualities, but also have a large community of photographers

to rate the visual quality of these photos, thereby providing excellent training data for the development of automatic algorithms for visual quality assessment.

Leveraging interesting negatives in the canonical views. In our experiments, we also observed interesting negatives in the canonical views. As a typical example, in the `Paris` collection, the 9th canonical view turns out to be a replica of the Eiffel tower at hotel Paris Las Vegas. Because we collected the images tagged by keyword `Paris`, many images are actually of hotel Paris Las Vegas. These images form a nontrivial cluster in the visual space, and one of them (the replica of the Eiffel tower) becomes top-ranked in the canonical views. Such negatives in the canonical views often reveal major ambiguations in the dataset. In this case, the keyword `Paris` has ambiguous interpretations and leads to poor image search results. It would be interesting to leverage such negatives, are use them as feedback to refine the query for image search.

The wisdom of crowds in other domains. The success of canonical view mining is a powerful demonstration of the wisdom of crowds: the aggregation of opinions within a crowd results in information that is otherwise difficult to obtain. Intuitively, the internet and its billions of users provide an excellent resource for the extraction of crowd intelligence. Beside images, the web hosts a huge amount of data across multiple domains: textual documents, audios, videos, *etc.*. Inevitably, huge amounts of noise and redundancy are prevalent in these domains as well. It would be interesting to adapt the proposed algorithms to these domains for the removal of noise and redundancy, and the extraction of crowd intelligence.

LIST OF REFERENCES

[1] Affine covariant features. http://www.robots.ox.ac.uk/~vgg/research/affine/.

[2] Dpchallenge. http://www.dpchallenge.com/.

[3] Facebook. http://www.facebook.com/.

[4] Flickr. http://www.flickr.com/.

[5] Flickr API. http://www.flickr.com/services/api/.

[6] Global positioning system. http://en.wikipedia.org/wiki/Global_Positioning_System.

[7] Google. http://www.google.com/.

[8] Google Image Swirl. http://image-swirl.googlelabs.com/.

[9] Google Images. http://images.google.com/.

[10] Google Street View. http://maps.google.com/help/maps/streetview/.

[11] Graphviz. http://www.graphviz.org/.

[12] Keypoint detector. http://www.cs.ubc.ca/~lowe/keypoints/.

[13] Photo.net. http://www.photo.net/.

[14] Picasa Web Albums. http://picasa.google.com/.

[15] TagMaps. http://tagmaps.research.yahoo.com/.

[16] ANN: A library for approximate nearest neighbor searching. http://www.cs.umd.edu/~mount/ANN/.

[17] UK recognition benchmark. http://www.vis.uky.edu/~stewe/ukbench/.

[18] S. Agarwal, N. Snavely, I. Simon, S. Seitz, and R. Szeliski. Building rome in a day. In *IEEE International Conference on Computer Vision (ICCV)*, pages 72–79, 2009.

[19] S. Arya and D. M. Mount. Algorithms for fast vector quantization. In *Data Compression Conference*, pages 381–390, 1993.

[20] R. A. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval*. Addison-Wesley Longman Publishing Co., Inc., 1999.

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

[22] J. Beis and D. Lowe. Shape indexing using approximate nearest-neighbour search in high-dimensional spaces. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1000–1006, 1997.

[23] J. L. Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, 1975.

[24] P. Bhat, K. Zheng, N. Snavely, A. Agarwala, M. Agrawala, M. Cohen, and B. Curless. Piecewise image registration in the presence of multiple large motions. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2491–2497, 2006.

[25] S. Brin and L. Page. The anatomy of a large-scale hypertextual web search engine. *Computer Networks and ISDN Systems*, 30(1–7):107–117, 1998.

[26] M. Brown and D. Lowe. Automatic panoramic image stitching using invariant features. *International Journal of Computer Vision*, 74(1):59–73, 2007.

[27] M. Brown, R. Szeliski, and S. Winder. Multi-image matching using multi-scale oriented patches. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 510–517, 2005.

[28] D. Comaniciu, P. Meer, and S. Member. Mean shift: A robust approach toward feature space analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 24(5):603–619, 2002.

[29] G. Cornuejols, M. L. Fisher, and G. L. Nemhauser. Location of bank accounts to optimize float: An analytic study of exact and approximate algorithms. *Management Science*, 23(8):789–810, 1977.

[30] D. J. Crandall, L. Backstrom, D. Huttenlocher, and J. Kleinberg. Mapping the world's photos. In *ACM International Conference on World Wide Web (WWW)*, pages 761–770, 2009.

[31] R. Datta, D. Joshi, J. Li, and J. Wang. Studying aesthetics in photographic images using a computational approach. In *European Conference on Computer Vision (ECCV)*, pages 288–301, 2006.

[32] S. Deerwester, S. T. Dumais, G. W. Furnas, T. K. Landauer, and R. Harshman. Indexing by latent semantic analysis. *Journal of the American Society for Information Science*, 41(6):391–407, 1990.

[33] I. S. Dhillon and D. S. Modha. Concept decompositions for large sparse text data using clustering. *Machine Learning*, 42(1–2):143–175, 2001.

[34] J. Fan, Y. Gao, H. Luo, D. A. Keim, and Z. Li. A novel approach to enable semantic and visual image summarization for exploratory image search. In *ACM International Conference on Multimedia Information Retrieval (MIR)*, pages 358–365, 2008.

[35] M. A. Fischler and R. C. Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, 1981.

[36] D. A. Forsyth and J. Ponce. *Computer Vision: A Modern Approach*. Prentice Hall, 2002.

[37] J.-M. Frahm, P. Fite-Georgel, D. Gallup, T. Johnson, R. Raguram, C. Wu, Y.-H. Jen, E. Dunn, B. Clipp, S. Lazebnik, and M. Pollefeys. Building rome on a cloudless day. In *European Conference on Computer Vision (ECCV)*, pages 368–381, 2010.

[38] B. J. Frey and D. Dueck. Clustering by passing messages between data points. *Science*, 315:972–976, 2007.

[39] J. H. Friedman, J. L. Bentley, and R. A. Finkel. An algorithm for finding best matches in logarithmic expected time. *ACM Transactions on Mathematical Software (TOMS)*, 3(3):209–226, 1977.

[40] B. Georgescu, I. Shimshoni, and P. Meer. Mean shift based clustering in high dimensions: a texture classification example. In *IEEE International Conference on Computer Vision (ICCV)*, pages 456–463, 2003.

[41] J. Goldberger and T. Tassa. A hierarchical clustering algorithm based on the hungarian method. *Pattern Recognition Letter*, 29(11):1632–1638, 2008.

[42] G. H. Golub and C. F. Van Loan. *Matrix Computations*. Johns Hopkins University Press, 1996.

[43] M. H. Harries, D. I. Perrett, and A. Lavender. Preferential inspection of views of 3-D model heads. *Perception*, 20(5):669–680, 1991.

[44] R. I. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, 2000.

[45] J. Hays and A. A. Efros. Scene completion using millions of photographs. *ACM Transactions on Graphics (TOG)*, 26(3), 2007.

[46] J. Hays and A. A. Efros. IM2GPS: estimating geographic information from a single image. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–8, 2008.

[47] T. Hofmann. Probabilistic latent semantic indexing. In *ACM International Conference on Research and Development in Information Retrieval (SIGIR)*, pages 50–57, 1999.

[48] C.-C. Hsieh, W.-H. Cheng, C.-H. Chang, Y.-Y. Chuang, and J.-L. Wu. Photo navigator. In *ACM International Conference on Multimedia (MM)*, pages 419–428, 2008.

[49] A. Irschara, C. Zach, J.-M. Frahm, and H. Bischof. From structure-from-motion point clouds to fast location recognition. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2599–2606, 2009.

[50] A. Jaffe, M. Naaman, T. Tassa, and M. Davis. Generating summaries and visualization for large collections of geo-referenced photographs. In *ACM International Workshop on Multimedia Information Retrieval (MIR)*, pages 89–98, 2006.

[51] Y. Jia, J. Wang, G. Zeng, H. Zha, and X.-S. Hua. Optimizing kd-trees for scalable visual descriptor indexing. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3392–3399, 2010.

[52] Y. Jia, J. Wang, C. Zhang, and X.-S. Hua. Finding image exemplars using fast sparse affinity propagation. In *ACM International Conference on Multimedia (MM)*, pages 639–642, 2008.

[53] Y. Jing and S. Baluja. Visualrank: Applying pagerank to large-scale image search. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 30(11):1877–1890, 2008.

[54] Y. Jing, S. Baluja, and H. Rowley. Canonical image selection from the web. In *ACM International Conference on Image and Video Retrieval (CIVR)*, pages 280–287, 2007.

[55] Y. Jing, M. Covell, and H. A. Rowley. Comparison of clustering approaches for summarizing large populations of images. In *IEEE International Conference on Multimedia and Expo (ICME)*, pages 1523–1527, 2010.

[56] Y. Jing, H. Rowly, and C. Rosenberg. Visualizing web images via google image swirl. In *NIPS Workshop on Statistical Machine Learning for Visual Analytics*, 2009.

[57] I. Jolliffe. *Principal Component Analysis*. Springer, 2002.

[58] E. Kalogerakis, O. Vesselova, J. Hays, A. Efros, and A. Hertzmann. Image sequence geolocation with human travel priors. In *IEEE International Conference on Computer Vision (ICCV)*, pages 253–260, 2009.

[59] Y. Ke and R. Sukthankar. PCA-SIFT: a more distinctive representation for local image descriptors. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 506–513, 2004.

[60] Y. Ke, R. Sukthankar, and L. Huston. An efficient parts-based near-duplicate and sub-image retrieval system. In *ACM International Conference on Multimedia (MM)*, pages 869–876, 2004.

[61] Y. Ke, X. Tang, and F. Jing. The design of high-level features for photo quality assessment. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 419–426, 2006.

[62] L. Kennedy and S.-F. Chang. Internet image archaeology: automatically tracing the manipulation history of photographs on the web. In *ACM International Conference on Multimedia (MM)*, pages 349–358, 2008.

[63] L. S. Kennedy, S.-F. Chang, and I. V. Kozintsev. To search or to label?: predicting the performance of search-based automatic image classifiers. In *ACM International Workshop on Multimedia Information Retrieval (MIR)*, pages 249–258, 2006.

[64] L. S. Kennedy and M. Naaman. Generating diverse and representative image search results for landmarks. In *ACM International Conference on World Wide Web (WWW)*, pages 297–306, 2008.

[65] H. W. Kuhn. The Hungarian method for the assignment problem. *Naval Research Logistic Quarterly*, 2:83–97, 1955.

[66] A. N. Langville and M. C. D. *Internet Mathematics*, 1(3):335–380, 2003.

[67] X. Li, C. Wu, C. Zach, S. Lazebnik, and J.-M. Frahm. Modeling and recognition of landmark image collections using iconic scene graphs. In *European Conference on Computer Vision (ECCV)*, pages 427–440, 2008.

[68] Y. Li and B. Merialdo. VERT: automatic evaluation of video summaries. In *ACM International Conference on Multimedia*, pages 851–854, 2010.

[69] Y. Li, N. Snavely, and D. P. Huttenlocher. Location recognition using prioritized feature matching. In *European Conference on Computer Vision (ECCV)*, pages 791–804, 2010.

[70] C.-Y. Lin and E. Hovy. Automatic evaluation of summaries using n-gram co-occurrence statistics. In *Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology*, pages 71–78, 2003.

[71] T. Lindeberg. Scale-space theory: A basic tool for analysing structures at different scales. *Journal of Applied Statistics*, 21(2):224–270, 1994.

[72] T. Liu, A. W. Moore, A. Gray, and K. Yang. An investigation of practical approximate nearest neighbor algorithms. In *Conference on Advances in Neural Information Processing Systems (NIPS)*, pages 825–832, 2005.

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

[74] K. Mikolajczyk and C. Schmid. Scale & affine invariant interest point detectors. *International Journal of Computer Vision*, 60(1):63–86, 2004.

[75] K. Mikolajczyk and C. Schmid. A performance evaluation of local descriptors. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 27(10):1615–1630, 2005.

[76] M. Muja and D. G. Lowe. Fast approximate nearest neighbors with automatic algorithm configuration. In *International Conference on Computer Vision Theory and Applications*, pages 331–340, 2009.

[77] H. Murase and S. K. Nayar. Visual learning and recognition of 3-D objects from appearance. *International Journal of Computer Vision (IJCV)*, 14(1):5–24, 1995.

[78] M. E. J. Newman. *Networks: An Introduction*. Oxford University Press, 2010.

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

[80] A. Oliva and A. Torralba. Modeling the shape of the scene: A holistic representation of the spatial envelope. *International Journal of Computer Vision*, 42(3):145–175, 2001.

[81] S. Palmer, E. Rosch, and P. Chase. Canonical perspective and the perception of 40 objects. In *Attention and Performance IX*, pages 135–151, 1981.

[82] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu. Bleu: a method for automatic evaluation of machine translation. In *Annual Meeting on Association for Computational Linguistics*, pages 311–318, 2002.

[83] D. I. Perrett and M. H. Harries. Characteristic views and the visual inspection of simple faceted and smooth objects: tetrahedra and potatoes. *Perception*, 17(6):703–720, 1988.

[84] D. I. Perrett, M. H. Harries, and S. Looker. Use of preferential inspection to define the viewing sphere and characteristic views of an arbitrary machined tool part. *Perception*, 21(4):497–515, 1992.

[85] D. Poole. *Linear Algebra: A Modern Introduction*. Brooks Cole, 2002.

[86] R. Raguram and S. Lazebnik. Computing iconic summaries of general visual concepts. In *IEEE Computer Vision and Pattern Recognition Workshops*, pages 1–8, June 2008.

[87] G. Salton and M. J. McGill. *Introduction to Modern Information Retrieval*. McGraw-Hill, 1986.

[88] G. Schindler, M. Brown, and R. Szeliski. City-scale location recognition. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–7, 2007.

[89] J. Shi and J. Malik. Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 22:888–905, 1997.

[90] B. Sigurbjörnsson and R. van Zwol. Flickr tag recommendation based on collective knowledge. In *ACM International Conference on World Wide Web (WWW)*, pages 327–336, 2008.

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

[92] I. Simon, N. Snavely, and S. Seitz. Scene summarization for online image collections. In *IEEE International Conference on Computer Vision (ICCV)*, pages 1–8, 2007.

[93] J. Sivic and A. Zisserman. Video google: a text retrieval approach to object matching in videos. In *IEEE International Conference on Computer Vision (ICCV)*, pages 1470–1477, 2003.

[94] N. Snavely, R. Garg, S. M. Seitz, and R. Szeliski. Finding paths through the world's photos. *ACM Transactions on Graphics*, 27(3):1–11, 2008.

[95] N. Snavely, S. M. Seitz, and R. Szeliski. Photo tourism: exploring photo collections in 3D. *ACM Transactions on Graphics (TOG)*, 25(3):835–846, 2006.

[96] J. Surowiecki. *The Wisdom of Crowds: Why the Many Are Smarter Than the Few and How Collective Wisdom Shapes Business, Economies, Societies and Nations*. Doubleday, 2004.

[97] M. J. Swain and D. H. Ballard. Color indexing. *International Journal of Computer Vision (IJCV)*, 7(1):11–32, 1991.

[98] R. Szeliski. Image alignment and stitching: a tutorial. *Foundations and Trends in Computer Graphics and Vision*, 2(1):1–104, 2006.

[99] A. Torralba, R. Fergus, and W. Freeman. 80 million tiny images: A large data set for nonparametric object and scene recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 30(11):1958–1970, 2008.

[100] B. Triggs, P. F. McLauchlan, R. I. Hartley, and A. W. Fitzgibbon. Bundle adjustment - a modern synthesis. In *International Workshop on Vision Algorithms: Theory and Practice*, pages 298–372, 2000.

[101] M. Turk and A. Pentland. Face recognition using eigenfaces. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 586–591, 1991.

[102] T. Tuytelaars and K. Mikolajczyk. Local invariant feature detectors: A survey. *Foundations and Trends in Computer Graphics and Vision*, 3(3):177–280, 2008.

[103] R. H. van Leuken, L. Garcia, X. Olivares, and R. van Zwol. Visual diversification of image search results. In *ACM International Conference on World Wide Web (WWW)*, pages 341–350, 2009.

[104] W. Wagenaar. My memory: A study of autobiographical memory over six years. *Cognitive Psychology*, 18:225–252, 1986.

[105] D. B. West. *Introduction to Graph Theory*. Prentice Hall, second edition, 2000.

[106] Y. H. Yang, P. T. Wu, C. W. Lee, K. H. Lin, W. H. Hsu, and H. H. Chen. ContextSeer: context search and recommendation at query time for shared consumer photos. In *ACM International Conference on Multimedia (MM)*, pages 199–208, 2008.

[107] Y.-T. Zheng, Z.-J. Zha, and T.-S. Chua. Research and applications on georeferenced multimedia: a survey. *Multimedia Tools and Applications*, 51(1):77–98, 2011.

[108] J. Zobel and A. Moffat. Inverted files for text search engines. *ACM Computing Surveys*, 38(2):6, 2006.

APPENDIX A

IMAGE CREDITS

We would like to thank the following Flickr users for allowing us to reproduce their images under Creative Commons licenses:

A bloke called Jerm	http://www.flickr.com/photos/sharkbait/
Adri	http://www.flickr.com/photos/cafebabe/
Alastair Taylor	http://www.flickr.com/photos/alastairtaylor/
Alessandra Del Gos	http://www.flickr.com/photos/11737229@N07/
Alex Barrow	http://www.flickr.com/photos/alexbarrow/
Alex Hart	http://www.flickr.com/photos/eahart/
Andrea Bandelli	http://www.flickr.com/photos/maphutha/
Andrea Marutti	http://www.flickr.com/photos/afeman/
Andrea	http://www.flickr.com/photos/sacherfire/
Andrew Churches	http://www.flickr.com/photos/churcho/
Andrew & Suzanne	http://www.flickr.com/photos/andrew_suzanne/
Andy Schultz	http://www.flickr.com/photos/aschultz/
Anita Mitchell	http://www.flickr.com/photos/nitz/
Antonio Olmo	http://www.flickr.com/photos/oncho88/
Bérenger ZYLA	http://www.flickr.com/photos/bzyla/
Backpacking Bex	http://www.flickr.com/photos/backpackingbex/
Bethany Weeks	http://www.flickr.com/photos/washuugenius/
Bill and Cathy	http://www.flickr.com/photos/billandcathyfuchs/
Br Lawrence Lew, O.P.	http://www.flickr.com/photos/paullew/
Breanne Kato	http://www.flickr.com/photos/guillotinehead/
Brendan Costigan	http://www.flickr.com/photos/63269749@N00/
Brian Garrett	http://www.flickr.com/photos/28122162@N04/
Brian Herzog	http://www.flickr.com/photos/herzogbr/
Brian Johnson & Dane Kantner	http://www.flickr.com/photos/danebrian/
Charles Kaiser	http://www.flickr.com/photos/ckaiserca/
Charlie Phillips	http://www.flickr.com/photos/savingfutures/
Chris Clayson	http://www.flickr.com/photos/theclaytaurus/
Chris Huffee	http://www.flickr.com/photos/chuffee/
Christopher Chan	http://www.flickr.com/photos/chanc/
Claudio Gennari	http://www.flickr.com/photos/claudiogennari/
Costas Tavernarakis	http://www.flickr.com/photos/taver/
Dale Musselman	http://www.flickr.com/photos/dalem/
Damien du Toit	http://www.flickr.com/photos/coda/
David Ohmer	http://www.flickr.com/photos/the-o/
David Poblador i Garcia	http://www.flickr.com/photos/nirvanis/
Davide Lonigro	http://www.flickr.com/photos/d1988/
David	http://www.flickr.com/photos/davidandnalini/
Davie Dunn	http://www.flickr.com/photos/daviedunn/
Dean Dietro Vun	http://www.flickr.com/photos/deanvun/
Dimitri N.	http://www.flickr.com/photos/dimi3/
Dimitry B.	http://www.flickr.com/photos/ru_boff/

Dom Hine	http://www.flickr.com/photos/domhuk/
Donna	http://www.flickr.com/photos/donna-andrew/
Dr Hilary Rhodes	http://www.flickr.com/photos/hilofoz/
Ed Yourdon	http://www.flickr.com/photos/yourdon/
Edwin Lee	http://www.flickr.com/photos/edwin11/
Emanuele	http://www.flickr.com/photos/zakmc/
Enzo D.	http://www.flickr.com/photos/enzod/
Eric Borda	http://www.flickr.com/photos/eric81/
Eustaquio Santimano	http://www.flickr.com/photos/eustaquio/
Fr Matthew Green	http://www.flickr.com/photos/mehjg/
Gabrielle Ludlow	http://www.flickr.com/photos/gabludlow/
Gary	http://www.flickr.com/photos/garyong/
Harald Hoyer	http://www.flickr.com/photos/hhoyer/
Heather Cowper	http://www.flickr.com/photos/heatheronhertravels/
InterNational Carmen*	http://www.flickr.com/photos/unionbig/
Ivor Brands	http://www.flickr.com/photos/ivornl/
James Stringer	http://www.flickr.com/photos/jamesstringer/
Jan&Marion	http://www.flickr.com/photos/49274163@N03/
Jared Hawkins	http://www.flickr.com/photos/jaredhawkins/
Jason Scragz	http://www.flickr.com/photos/scragz/
Jean-Noël Dollé	http://www.flickr.com/photos/jinod/
Jess J	http://www.flickr.com/photos/jessicajuriga/
Jingjing	http://www.flickr.com/photos/jingjing/
Joe Baz	http://www.flickr.com/photos/joebaz/
Johan Biré	http://www.flickr.com/photos/donemat/
John G. Walter	http://www.flickr.com/photos/8037063@N02/
Jonathan Ziapour	http://www.flickr.com/photos/jonathanziapour/
Jose Maria Cuellar	http://www.flickr.com/photos/cuellar/
Joseph Younis	http://www.flickr.com/photos/strike1/
Kevin Hoogheem	http://www.flickr.com/photos/khoogheem/
Kevin Tostado	http://www.flickr.com/photos/tostie14/
Konrad Glogowski	http://www.flickr.com/photos/teachandlearn/
Latte_Art	http://www.flickr.com/photos/latte_art/
Lava	http://www.flickr.com/photos/lavaland/
Leandro Demori	http://www.flickr.com/photos/32360435@N02/
Lee Kindness	http://www.flickr.com/photos/lkindness/
Leonardo	http://www.flickr.com/photos/sgatto/
Lo van den Berg	http://www.flickr.com/photos/blondavenger/
Lucian Bickerton	http://www.flickr.com/photos/lucianbickerton/
Luigi Guarino	http://www.flickr.com/photos/luigi_and_linda/
Luis Rubio	http://www.flickr.com/photos/luisrubio/
Lydia	http://www.flickr.com/photos/miss_l/
M Soli	http://www.flickr.com/photos/msoli/
M Trombone	http://www.flickr.com/photos/min0n/
Mandi Lindner	http://www.flickr.com/photos/msquarter/

Manuel Faisco	http://www.flickr.com/photos/arteurbana/
Marcel	http://www.flickr.com/photos/marcelix/
Mark Bayes	http://www.flickr.com/photos/32087030@N07/
Mark Pozzobon	http://www.flickr.com/photos/mpozzobon/
Martijn van de Streek	http://www.flickr.com/photos/treenaks/
Martin Wippel	http://www.flickr.com/photos/martin_julia/
Mathew F	http://www.flickr.com/photos/canvy/
Mathias Mildenberger	http://www.flickr.com/photos/mathias_m/
Matt Conrad	http://www.flickr.com/photos/conrad_matt/
Matteo Piotto	http://www.flickr.com/photos/namuit/
Matthew Bradley	http://www.flickr.com/photos/mjb/
Matthias Läßig	http://www.flickr.com/photos/oscailte/
Maurice	http://www.flickr.com/photos/mauricedb/
Mete Dönmez	http://www.flickr.com/photos/mdcworks/
Michael Dawes	http://www.flickr.com/photos/tk_five_0/
Michel Guilly	http://www.flickr.com/photos/13945579@N03/
Michela Simoncini	http://www.flickr.com/photos/comunicati/
Mike & Maggie	http://www.flickr.com/photos/fetchpics/
Moyan Brenn	http://www.flickr.com/photos/aigle_dore/
Nenita Casuga	http://www.flickr.com/photos/nenita_casuga/
Nicolas et Clémence ETIQUE	http://www.flickr.com/photos/clemetnic/
Padraic Woods	http://www.flickr.com/photos/padraicwoods/
Paolo	http://www.flickr.com/photos/raselased/
Patrick Morgan	http://www.flickr.com/photos/pmorgan67/
Pedro Lopez	http://www.flickr.com/photos/pedroqtc/
Peter & Joyce Grace	http://www.flickr.com/photos/gracefamily/
Philip Milne	http://www.flickr.com/photos/pamilne/
Pierre Châtel	http://www.flickr.com/photos/chatelp/
Princess Purple	http://www.flickr.com/photos/thepurpleempire/
Rafal Kiermacz	http://www.flickr.com/photos/kiermacz/
René Querin	http://www.flickr.com/photos/querin/
Rizzie	http://www.flickr.com/photos/rizziemelb/
Robert Liu	http://www.flickr.com/photos/robertliu/
Roberto	http://www.flickr.com/photos/corner_of_my_eye/
Ron	http://www.flickr.com/photos/rollingstone2009/
Ross	http://www.flickr.com/photos/syder/
Ryan Kelly	http://www.flickr.com/photos/rpkelly/
Sébastien Bertrand	http://www.flickr.com/photos/tiseb/
Saket Vora	http://www.flickr.com/photos/saket_vora/
Sally Taylor	http://www.flickr.com/photos/sataylor/
Sam Holloway	http://www.flickr.com/photos/aliasrex/
Samantha Ayres	http://www.flickr.com/photos/14270088@N06/
Sara	http://www.flickr.com/photos/nastrina/
Scott Trulock	http://www.flickr.com/photos/strulock/
Scott	http://www.flickr.com/photos/scottcwebster/

Serge Melki	http://www.flickr.com/photos/sergemelki/
Seth Chandler	http://www.flickr.com/photos/sethbc/
Shannon Lynch	http://www.flickr.com/photos/killahpoopface/
Shawn Stilwell	http://www.flickr.com/photos/shawnstilwell/
Sian Evans	http://www.flickr.com/photos/diaphanus/
Simon Bisson	http://www.flickr.com/photos/sbisson/
Simon Laird	http://www.flickr.com/photos/swarve/
Steven	http://www.flickr.com/photos/erwan2007/
Stewart Butterfield	http://www.flickr.com/photos/stewart/
Tahbepet	http://www.flickr.com/photos/86214920@N00/
Temari 09	http://www.flickr.com/photos/34053291@N05/
Terry Dunn	http://www.flickr.com/photos/terrydel/
Thomas Hawk	http://www.flickr.com/photos/thomashawk/
Toby Garden	http://www.flickr.com/photos/toby_garden/
Tom Hargett	http://www.flickr.com/photos/22218618@N08/
Tom Mendalka	http://www.flickr.com/photos/mendalka/
Tommaso Masetti	http://www.flickr.com/photos/leekelso/
Valentin Likyov	http://www.flickr.com/photos/vlx/
Vicki Potts	http://www.flickr.com/photos/kiwi_vic/
Ville Miettinen	http://www.flickr.com/photos/wili/
Ville Misaki	http://www.flickr.com/photos/vlumi/
Vyacheslav Argenberg	http://www.flickr.com/photos/argenberg/
Wally Gobetz	http://www.flickr.com/photos/wallyg/
Will & Becky Munns	http://www.flickr.com/photos/will-and-becky/
Yongming Hou	http://www.flickr.com/photos/cupman/
Zach Flanders	http://www.flickr.com/photos/zachflanders/
alex de carvalho	http://www.flickr.com/photos/adc/
ammog	http://www.flickr.com/photos/50379127@N08/
andrew wales	http://www.flickr.com/photos/stanrandom/
boris doesborg	http://www.flickr.com/photos/batigolix/
cfcall02	http://www.flickr.com/photos/56482997@N07/
cobalt	http://www.flickr.com/photos/cobalt/
david aouita	http://www.flickr.com/photos/elaouita/
dirk huijssoon	http://www.flickr.com/photos/dicknella/
emc	http://www.flickr.com/photos/elderc/
jaysun093	http://www.flickr.com/photos/jaysun093/
jtop2007	http://www.flickr.com/photos/7831033@N08/
karmacamilleeon	http://www.flickr.com/photos/karmacamilleeon/
kperkins14	http://www.flickr.com/photos/keithperkins/
lamele	http://www.flickr.com/photos/lamele/
nickgubbins	http://www.flickr.com/photos/nickgubbins/
ruudwessels	http://www.flickr.com/photos/49756869@N04/
speedygroundhog	http://www.flickr.com/photos/speedygroundhog/
theodorahhh	http://www.flickr.com/photos/theodorahhh/
tripu	http://www.flickr.com/photos/tripu/

We would like to thank the following Wikipedia users for allowing us to reproduce their images under Creative Commons licenses:

László Szalai	http://commons.wikimedia.org/wiki/User:Beyond_silence
Sanchezn	http://commons.wikimedia.org/wiki/User:Sanchezn
Benh LIEU SONG	http://commons.wikimedia.org/wiki/User:Benh
Incola	http://commons.wikimedia.org/wiki/User:Incola
Ra Boe	http://de.wikipedia.org/wiki/Benutzer:Raboe001
Sanjay Acharya	http://commons.wikimedia.org/wiki/User:Sanjay_ach
chensiyuan	http://en.wikipedia.org/wiki/Yosemite_Valley