Learning Distance Functions for Exemplar-Based Object Recognition

by

Andrea Lynn Frome

B.S. (Mary Washington College) 1996

A dissertation submitted in partial satisfaction
of the requirements for the degree of

Doctor of Philosophy

in

Computer Science

in the

GRADUATE DIVISION

of the

UNIVERSITY OF CALIFORNIA, BERKELEY

Committee in charge:

Professor Jitendra Malik, Chair
Professor Michael I. Jordan
Professor Bruno Olshausen

Fall 2007

The dissertation of Andrea Lynn Frome is approved:

Professor Jitendra Malik, Chair	Date
Professor Michael I. Jordan	Date
Professor Bruno Olshausen	Date

University of California, Berkeley

Fall 2007

Abstract

Learning Distance Functions for Exemplar-Based Object Recognition

by

Andrea Lynn Frome

Doctor of Philosophy in Computer Science

University of California, Berkeley

Professor Jitendra Malik, Chair

This thesis investigates an exemplar-based approach to object recognition that learns, on an image-by-image basis, the relative importance of patch-based features for determining similarity. We borrow the idea of "family resemblances" from Wittgenstein's Philosophical Investigations and Eleanor Rosch's psychological studies to support the idea of learning the detailed relationships between images of the same category, which is a departure from some popular machine learning approaches such as Support Vector Machines that seek only the boundaries between categories.

We represent images as sets of patch-based features. To find the distance between two images, we first find for each patch its nearest patch in the other image and compute their inter-patch distance. The weighted sum of these inter-patch distances is defined to be the distance between the two images. The main contribution of this thesis is a method for learning a set-to-set distance function specific to *each training image* and demonstrating the use of these functions for image browsing, retrieval, and classification. The goal of the learning algorithm is to assign a non-negative weight to each patch-based feature of the image such that the most useful patches are assigned large weights and irrelevant or confounding patches are given zero weights. We formulate this as a large-margin optimization, related to the soft-margin Support

Vector Machine, and discuss two versions: a “focal” version that learns weights for each image separately, and a “global” version that jointly learns the weights for all training images. In the focal version, the distance functions learned for the training images are not directly comparable to one another and can be most directly applied to in-sample applications such as image browsing, though with heuristics or additional learning, these functions can be used for image retrieval or classification. The global approach, however, learns distance functions that are globally consistent and can be used directly for image retrieval and classification. Using geometric blur and simple color features, we show that both versions perform as well or better than the best-performing algorithms on the Caltech 101 object recognition benchmark. The global version achieves the best results, a 63.2% mean recognition rate when trained with fifteen images per category and 66.6% when trained with twenty.

Professor Jitendra Malik, Chair　　　　Date

Acknowledgements

This accomplishment would not be possible without many turns of good fortune, the aggregate support of many, many people, and the efforts of those who have helped me find my way along a winding path.

First must come my advisor, Jitendra Malik. In addition to being a dedicated advisor, he has been an impassioned advocate, supporter, and teacher. From our many discussions I have been inspired to aim for the grander goals of Computer Vision, and his far-reaching knowledge of the field and its trends has enabled me to see the broader picture that emerges from the research details. This is truly a gift as I leave to apply my expertise. Thank you also to Michael Jordan and Bruno Olshausen, for serving both on my qualifying and thesis committees. Additional thanks to Mike for being an amazing teacher and resource. Since my first year of graduate school, I've been inspired by his amazing ability to interrelate and organize concepts and communicate them in a way that left me with a deep sense of awe.

My professional development also played an instrumental role in getting me where I am today, and I owe a debt of gratitude to Blue Mug, Inc. for giving me my first real software engineering job, and a big thank you to Dave and Penny Loftessness for being such inspiring leaders. A big thank you to Google and in particular Chuck Rosenberg, Radhika Malpani, and Yoram Singer for many amazing opportunities and collaborations the last two years.

In my education, I have been incredibly fortunate; somehow I had all the best teachers and TAs, thank you en masse for your dedication and for passing along your knowledge and excitement. There have been several individuals that have each made a singular difference. Dr. Michael Bass, my undergraduate advisor in Environmental Science, was a fantastic advisor and advocate when I was at Mary Washington College and has since become a dear friend. Thank you, Dr. Bass, for continuing to stand by my side and support me, even as I moved away from Environmental Science into Computer Science. I am deeply grateful to Barbara Hightower and all those that made the Computer Science Reentry Program possible; without it, my graduate career would never have been possible. Thank you to Jerome Feldman for taking me on as a research assistant prior to my starting as a graduate student; I learned an amazing amount during my short time in his group. Thank you to Stuart Russell and Nello Cristianini, who were my advisors my first two years as a Masters student. What I learned those first two years set the stage for the work that followed.

Thank you to my coauthors and collaborators, Ravi Kolluri, Thomas Bülow, Daniel Huber, Yoram Singer, and Fei Sha, and to those at the Sarnoff Corporation with whom I worked on E3D, including Harpreet Sawney and Bogdan Matei. I have also received invaluable help from Alex Berg and Hao Zhang.

I have had the great fortune of being supported throughout my graduate studies. Thank you to Intel Corporation for the IMAP Fellowship that supported my first two years as a graduate student. For the remainder of my graduate career I was supported by the DARPA E3D program (F33615-02-C-1265), ONR Grant N00014-06-1-0734, and a gift from Google.

Many friends have been by my side through the various stages of my life and education, and their humor and support have sustained me. My deepest gratitude and love to Irving Birmingham, Jonathan Karpick, Jamye Michele Jamison, Chris Moore, Shelby Hadfield, Mike Johnson, Colette Johnson, Bryan Smith, Davina Armstrong, Mark Paskin, Baq Haidri, Brandon Brown, Noah Treuhaft, Sean Rhea, Sirish Chandrasekaran, Brooke Kennedy, Elis Bradshaw, Greg Mori, Huyen Mori, Adam Kirk, and Pete MacMillin. And a big thank you to my fellow graduate students and friends who brightened every day and helped me along more than they could know: Dan Adkins, Okan Arikan, Alex Berg, Tamara Berg, Ben Bergen, Lillian Chu, Ashley

Eden, Jaety Edwards, Barbara Engelhardt, Bryan Feldman, Charless Fowlkes, Floraine Grabler, Hayley Iben, Leslie Ikemoto, Shawn Jeffery, Pushkar Joshi, David Liu, Dave Martin, Michael Maire, Deva Ramanan, Jason Sanders, Jason Waddle, Mark Whitney, and many others.

Another thanks to Mark Paskin for his thesis-in-a-box package, which takes all the pain out of the UC Berkeley thesis formatting requirements. Thank you to my early computers, the Atari 400 and Osborne 1, and to my parents for not discouraging me from writing hundreds of lines of BASIC in order to cheat on my arithmetic homework.

And finally, thank you to my family for a lifetime of love and support. It must be hard to believe I'm not a student anymore.

Dedicated to my parents, Anna and Richard Frome, and to my sister, Lesley.

Contents

Consider for example the proceedings that we call "games". I mean board-games, card-games, ball-games, Olympic games, and so on. What is common to them all?—Don't say "There *must* be something common, or they would not be called 'games' "—but *look and see* whether there is anything common to all.—for if you look at them you will not see something that is common to *all*, but similarities, relationships, and a whole series of them at that. To repeat: don't think, but look!—Look for example at board-games, with their multifarious relationships. Now pass to card-games; here you find many correspondences with the first group, but many common features drop out, and others appear. When we pass next to ball-games, much that is common is retained, but much is lost.—Are they all 'amusing'? Compare chess with noughts and crosses. Or is there always winning and losing, or competition between players? Think of patience. In ball-games there is winning and losing; but when a child throws his ball at the wall and catches it again, this feature has disappeared. Look at the parts played by skill and luck; and at the difference between skill in chess and skill in tennis. Think now of games like ring-a-ring-a-roses; here is the element of amusement, but how many other characteristic features have disappeared! And we can go through the many, many other groups of games in the same way; can see how similarities crop up and disappear.

And the result of this examination is: we see a complicated network of similarities overlapping and criss-crossing: sometimes overall similarities, sometimes similarities of detail.

– Ludwig Wittgenstein, Philosophical Investigation #66

Chapter 1

Introduction

1.1 What is Object Recognition?

The human visual system allows us to effortlessly perceive properties of the world—three-dimensional structure, boundaries, texture, colors, motion—and combine cues to make higher-level distinctions about the things that surround us. For purposes of this discussion, we define full object recognition as the task of determining the rough extent of an individual object in the visual field or in an image and identifying it as a member of a category containing other, similar objects. This thesis focuses on the challenge of performing these tasks with a computer from raw digital input, such as photographs or video.

Of course, even the concept of an "object" is a subjective human construct, debated by psychologists and philosophers, not physicists and mathematicians. It can be difficult to draw a line between what we consider a part and an object unto itself, or to determine the physical boundary between two objects or between object and non-object. (At the extreme, what exists at the atomic level are more and less dense clouds of electrical charge, hardly something that lends itself to a strict definition.)

This serves to illustrate, that even as we work with algorithms and mathematical concepts, our task is fundamentally one defined by the abilities of the human perceptual system.

Recognition is also a fuzzy concept; the meanings of "similar" and "category" are human constructs, shaped by our context and perceptual systems and studied by psychologists and psychophysicists. At best, the fuzzy, ill-defined output of the human perceptual system is the starting point for machine vision applications. Realistically, though, it is not a direct input; instead images have to be digitally collected and labeled, usually by underpaid students.

It could be argued that there exist engineering applications where the goal of a machine vision system is defined rigorously, for example, to find all the widgets on an assembly line that are faulty or correctly match faces with names. But even as algorithms show signs of out-performing humans on narrowly defined tasks, it is the human visual system that motivates the task and/or provides the feedback needed to train the system.

As for the terminology in the machine vision field, "recognition" can have different meanings depending upon the context. The task of labeling an entire image with one of 100 different generic categories is considered object recognition, but this is different from the terminology used used working with only faces; face *detection* is the task of finding the faces an in image, and face *recognition* is the grouping an image of a single face with others that belong to the same individual. So, while machine object recognition efforts are motivated and reinforced by the human visual system, fundamentally the task is defined by the data used to train the system and the direction chosen by the researcher. In this way, what the field considers object recognition is in the end dictated by the state of the art in algorithms and our ability to gather, label, and make use of visual data.

1.2 Object Recognition is Rapidly Improving

However we define it, we should be able to agree that we are getting better at it. We design and make use of elegant mathematical algorithms, but in the end machine vision is an empirical study, defined by the data we use. Without quantitatively comparing results on well-defined, benchmarked tasks, we cannot know what methods work better than others. Of course different methods will be better suited to different tasks, but for a *given task*, there are certainly some approaches that will perform better than others. Benchmarks of increasing difficulty—and competition between researchers to achieve the best results on them—have defined and driven the state of the art in general-purpose object recognition.

There are of course other advancements that helped us get to where we are today. The increase in computer speed and memory, the prevalence of computing clusters and multi-core processors, and powerful parallel computing tools have made it tractable to use more processor- and memory-intensive algorithms. The field appears to be converging upon patch-based shape features as a powerful tool for summarizing the content of images, and these have been "evolving" over the years, for example from the Gaussian-derivative jet descriptor of [Schmid and Mohr, 1996b] to the SIFT descriptor [Lowe, 1999] to the shape context [Belongie *et al.*, 2002] and geometric blur [Berg and Malik, 2001] descriptors. Probably the largest factor affecting the collection of today's data sets is the Internet; there are now large repositories of partially-labeled images, for example from Flickr or Google Image Search, that were not available just a few years ago.

While the data sets have been a primary factor in advancing the field, particularly in the last four years, every data set that has been widely used in the field has had its drawbacks. This is addressed in a recent article co-written by members of many of the top research groups in the field [J. Ponce and Zisserman, 2006]. While it is considered to be a flawed data set, the Caltech 101 data set introduced by [Fei-Fei *et al.*, 2004] has had a large, lasting impact on the field, and at the time of this writing is still actively used by many research groups[1]. Until its introduction, the recognition of varied object categories in two-dimensional images was limited to less than 10 categories, for example the faces, airplanes, cars, and motorbikes of Caltech-4 [Fergus *et al.*, 2003]. Caltech 101 was an increase by an order of magnitude with 101 different object classes (plus a background class), covering both man-made and natural, rigid and articulated objects. It is true that this data set is easy in some of the ways that earlier data sets were easy (for example, it includes the four categories from Caltech-4). In many of the images, the object takes up almost the entire image, and in many classes, the objects are in the same location and pose across images. The creators of the data set also introduced exploitable artifacts by rotating images, leaving black triangles in the corners of the images. Figure 1.1 shows a representation by Antonio Torralba of the data set, created by resizing all the images and for each category computing a pixel-wise average. It nicely illustrates the amount of regularity within each category since a few can be easily identified by their average. However, many of the categories, in particular the animal categories, remain very difficult.

Despite these regularities and artifacts, this was a very difficult data set when it was introduced. Figure 1.2 shows the progress on the data set over the last three years. The group that introduced it reported performance around 16% using fifteen examples per category, and within three years, we are almost to 65%, some of that progress coming from researchers building upon and refining other researchers' algorithms and features. The algorithm which is the main contribution of this thesis is the highest line on this graph at fifteen and twenty images per category.

[1]Information about the data set, images, and published results can be found at `http://www.vision.caltech.edu/Image_Datasets/Caltech101/Caltech101.html`

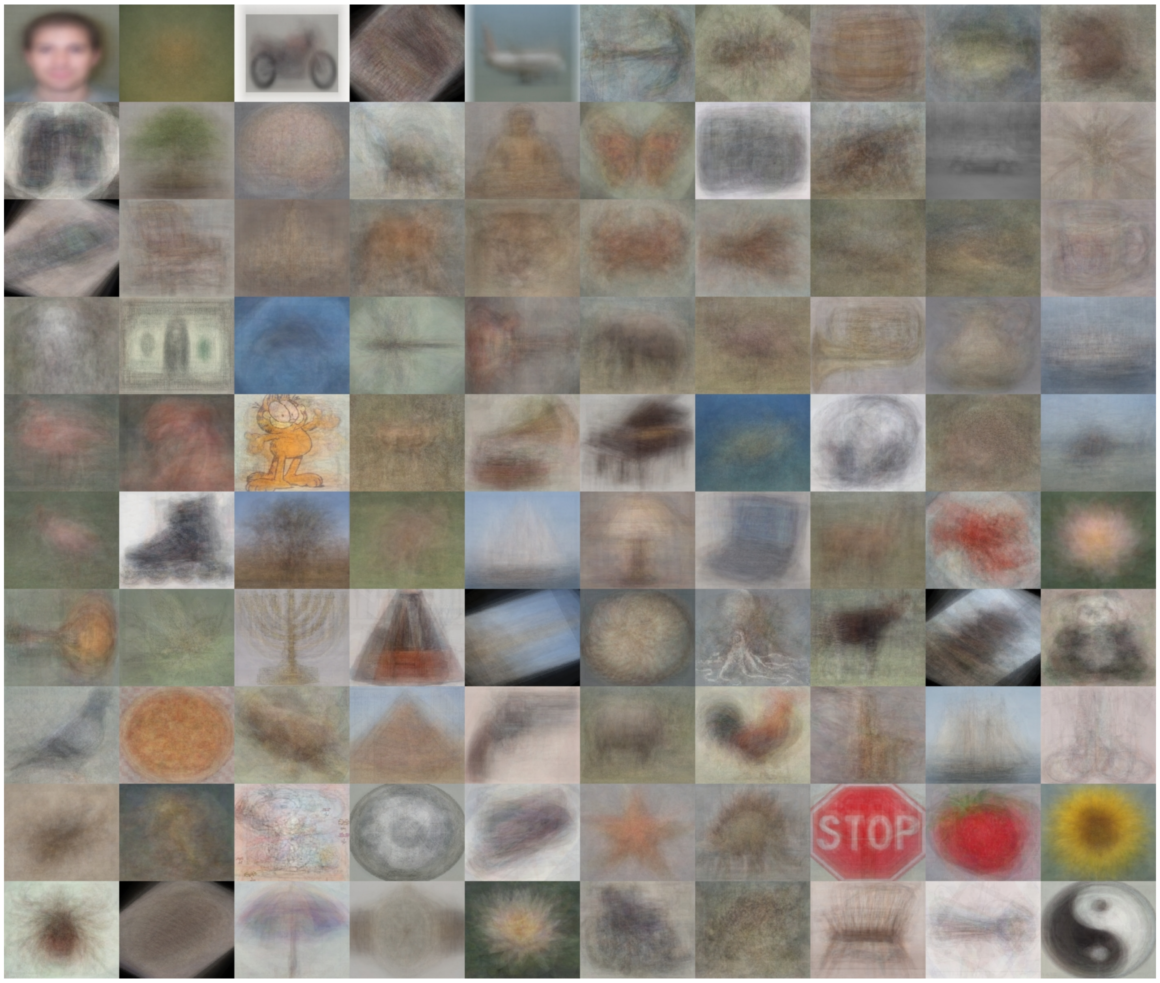


Figure 1.1: A representation of 100 of the categories of Caltech 101 (the `Faces_easy` category was removed). All the images in each category were resized, and the pixel-wise average was taken. These are the raw, unnormalized outputs. Courtesy Antonio Torralba.

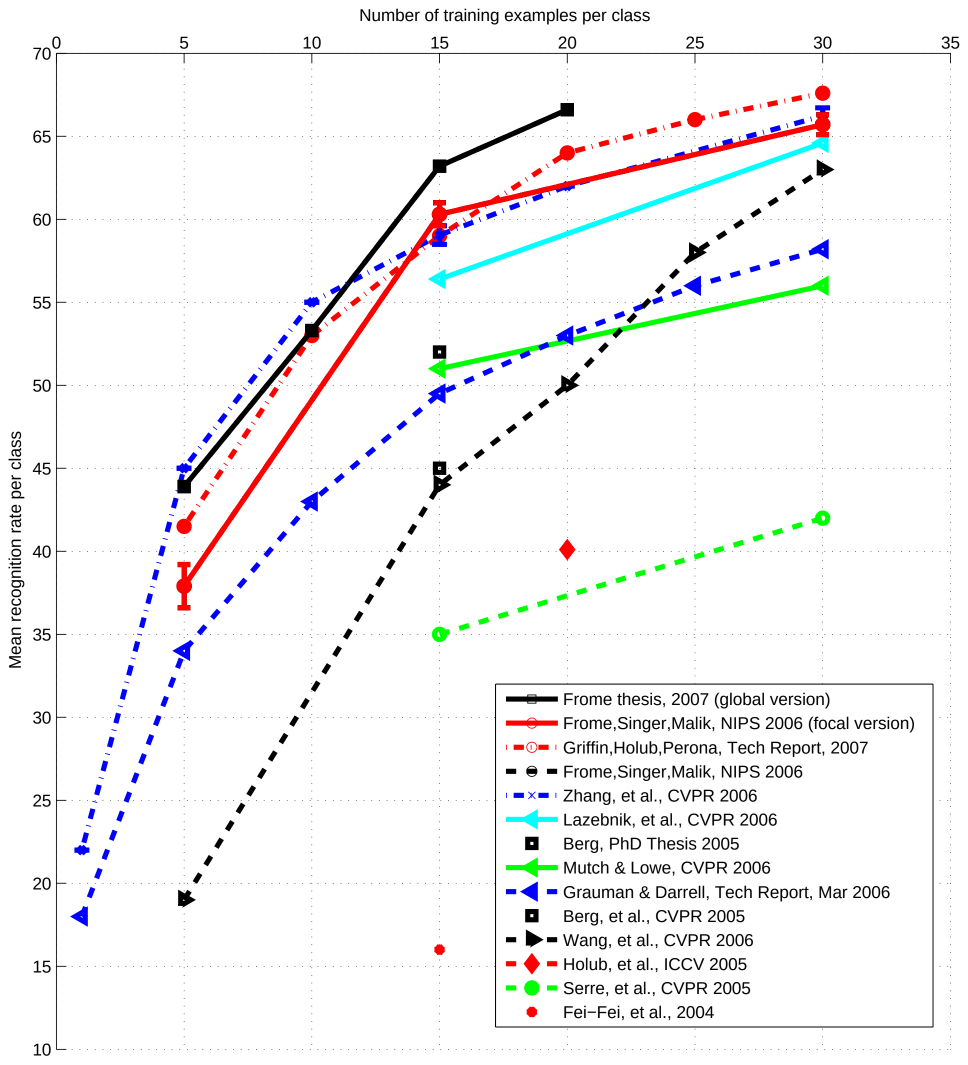


Figure 1.2: Number of images per category versus mean recognition rate on the Caltech 101 data set. Shown are results from Chapter 5 of this thesis, [Griffin *et al.*, 2007], [Zhang *et al.*, 2006], [Lazebnik *et al.*, 2006], [Mutch and Lowe, 2006], [Grauman and Darrell, 2006b], [Berg *et al.*, 2005], [Wang *et al.*, 2006], [Holub *et al.*, 2005], [Serre *et al.*, 2005], and [Fei-Fei *et al.*, 2004]. Note that the results just below ours at 20 images per category are computed differently; they do not include the `Faces_easy` category in training or testing, thus eliminating a prominent confuser.

1.3 Thesis Outline

Chapter 2 discusses our early work in the identification of different vehicles in three-dimensional laser range data. It discusses a feature we introduced for this task, and explores the effect of different matching approaches on recognition performance. It also covers the basic ideas behind semi-local feature matching for recognition and the advantages of fast nearest-neighbor algorithms, and compares the performance of three different feature types on the task.

The main contribution of this thesis is an approach to object recognition that uses the similarities between semi-local or patch-based features to learn distance functions between images. Chapter 3 frames the problem as one of learning distance functions between images, and motivates the approach with psychological studies into human categorization and examples from the Caltech 101 data set. It shows how we can learn from the similarity relationship within a triplet of images, how we can formulate a distance function from similarities between patch-based features, and introduces two variants of the approach, “focal” and “global” learning. Chapter 4 discusses the focal version, derives the optimization used, and describes in detail how it is solved. Examples of image browsing results are given for Caltech 101, as well as classification results. Chapter 5 discusses the global variant, gives classification results on Caltech 101, and shows samples of retrieval results.

Chapter 2

Features for Recognition in 2 1/2 Dimensions

2.1 Regional Descriptor Approach

Our approach to object recognition relies on the matching of feature vectors (also referred to here as *features*) which characterize a region of a two-dimensional (2D) or 3D image, where by “3D image” we mean the point cloud resulting from a laser range scan. We use the term *descriptor* to refer to the method or “template” for calculating the feature vector. There are several lines of work which develop descriptors for use in object recognition. [Schmid and Mohr, 1996a] introduced jet-based features; [Lowe, 1999] introduced the scale- and rotation-invariant feature transform (SIFT) descriptor for recognition and matching in two-dimensional intensity images; [Johnson and Hebert, 1999] describes the *spin image* descriptor for recognizing objects by shape in 3D range scans; [Belongie *et al.*, 2002] describes a histogram-based descriptor for recognizing objects in 2D images by shape, called the *shape context*, which is extended to the *generalized shape context* in [Mori and Malik, 2003]. In this chapter we discuss the 3D Shape Context descriptor, first introduced in [Frome *et al.*, 2004], which extends the two-dimensional shape context to three dimensions.

The spin image and shape context descriptors share a *regional* approach to feature calculation; the features incorporate information within a *support region* of the image centered at a chosen *basis point*. The locality of these *regional descriptors* make them robust to clutter and occlusion, while at the same time each feature contains more information than purely local descriptors due to their extended support. In some recognition approaches the features are computed at particularly salient locations in the image determined by an *interest operator*, such as in [Lowe, 1999]. In other approaches, including the cited works that make use of spin images and shape contexts, the basis points at which features are computed are chosen randomly from among edge or surface points and are not required to posses any distinguishing characteristics.

Object recognition algorithms typically work by calculating features from a query image and comparing those features to other features previously calculated from a set of *reference* images, and return a decision about which object or image from among the reference set best matches the query image. We consider *full object recognition* to be achieved when the algorithm returns the identity, location, and position of an object occurring in a query image. Our discussion in this chapter focuses on a relaxed version of the full recognition problem where the algorithm returns a *short list* of objects, at least one of which occurs *somewhere* in the image. An algorithm solving this relaxed recognition problem can be used to prune a large field of candidate objects for a more expensive algorithm which solves the full recognition problem. In a more complex system it could be used as an early stage in a cascade of object recognition algorithms which are increasingly more expensive and discriminating, similar in spirit to the cascade of classifiers made popular in the vision community by [Viola and Jones, 2004]. A pruning step or early cascade stage is effective when it reduces the total computation required for full recognition and does not reduce the recognition performance of the system. To this end, we want a short-list recognition algorithm which (1) minimizes the number of misses, that is, the fraction of queries where the short list does not include any objects present in the query image, and (2) minimizes its computational cost.

Object recognition algorithms based on features have been shown to achieve high recognition rates in the works cited above and many others, though often in a an easy or restricted recognition setting. We will demonstrate methods for speeding a simple matching algorithm while maintaining high recognition accuracy in a difficult recognition task, beginning with an approach which uses an exhaustive k-nearest-neighbor (k-NN) search to match the *query features* calculated from a query image to the *reference features* calculated from the set of reference images. Using the distances calculated between query and reference features, we generate a short list of objects which might be present in the query image.

It should be noted that the method we examine does not enforce relative geometric constraints between the basis points in the query and reference images, and that most feature-based recognition algorithms do use this additional information. For example, for reference features centered at basis points p_1 and p_2 and query features centered at basis points q_1 and q_2, if p_1 is found to be a match for q_1, p_2 a match for q_2, and we are considering rigid objects, then it should be the case that the distance in the image between p_1 and p_2 should be similar to the distance between q_1 and q_2. There are many methods for using these types of constraints, [Huttenlocher and Ullman, 1990], RANSAC, and [Berg *et al.*, 2004] to name a few. We choose not to use these constraints in order to demonstrate the power of matching feature vectors alone. A geometric-based pruning or verification method could follow the matching algorithms described in this chapter.

The drawback of an exhaustive search of stored reference features is that it is expensive, and for the method to be effective as a pruning stage, it needs to be fast. Many of the descriptors listed above are high-dimensional; in the works cited, the scale-invariant feature transform (SIFT) descriptor has 160 dimensions, the spin image has about 200, the 2D shape context has 60 (the generalized version has twice as many for the same number of bins), and the 3D shape context has almost 2000. The best algorithms for exact nearest-neighbor search in such high-dimensional spaces requires time linear in the number of reference features. In addition, the number of reference features is linear in the number of example objects the system is designed to recognize. If we aim to build systems that can recognize hundreds or thousands of example objects, then the system must be able to run in time sub-linear in the number of objects.

The goal of this chapter is to present ways to maintain the recognition accuracy of this "short-list" algorithm while reducing its computational cost. Locality-sensitive hashing (LSH) plays a key role in a final approach that is both accurate and has complexity sub-linear in the number of objects being recognized. In our experiments we will be evaluating variations on the basic matching method with the 3D shape context descriptor.

2.2 Shape Context Descriptors

We will focus on a type of descriptor called the *shape context*. In their original form, shape context features characterize shape in 2D images as histograms of edge pixels (see [Belongie *et al.*, 2001]). In [Mori and Malik, 2003] the authors use the same template as 2D shape contexts but capture more information about the shape by storing aggregate edge orientation for each bin. In [Berg and Malik, 2001], the authors developed the notion of *geometric blur* which is an analog to the 2D shape context for continuous-valued images. We extended the shape context to three dimensions in [Frome *et al.*, 2004], where it characterizes shape by histogramming the position of points in a range scan. In the rest of this section, we describe the basics of the 2D and 3D shape context descriptors in more detail, and introduce the experimental framework used in the rest of the chapter.

2.2.1 Two-dimensional Shape Contexts

To calculate a 2D shape context feature from an image, first run your favorite edge detector on the image. Next, choose a coordinate in the edge map to be a basis point, and imagine a radar-like template like the one in Figure 2.1 laid down over the image, centered at that point. The lines of this pattern divide the image into regions, each of which corresponds to one dimension of the feature vector. The value for the dimension is calculated as the number of edge pixels which fall into the region. This feature vector can be thought of as a histogram which summarizes the spatial distribution of edges in the image relative to the chosen basis point. Each region in the template corresponds to one bin in the histogram, and we use the term *bin* to refer to the region in the image as well as the dimension in the feature vector. Note that if the bins were small enough to each contain one pixel, then the histogram would be an exact description of the shape in the support region.

This template has a few advantageous properties. The bins farther from the center summarize a larger area of the image than those close to the center. The gives a foveal effect; the feature more accurately captures and weights more heavily information toward the center. To accentuate this property of shape context descriptors, we use equally spaced log-radius divisions. This causes bins to get "fuzzy" more quickly as you move from the center of the descriptor.

When comparing two shape context features, even if the shapes from which they are calculated are very similar, the following must also be similar in order to register the two features as a good match:

- orientation of the descriptor relative to the object
- scale of the object

To account for different scales, we can search over scale space, e.g., by calculating a Gaussian pyramid for our query image, calculating query features in each of the down- and up-scaled images, and finding the best match at each scale. We could sidestep the issue of orientation by assuming that objects are in a canonical orientation in the images, and orient the template the same way for all basis points. Or, to make it robust to variation, we could orient the template to the edge gradient at the basis point or include in our training set images at different orientations.

2.2.2 Three-dimensional Shape Contexts

In order to apply the same idea to range images, we extended the 2D shape context template to three dimensions. The support region for a 3D shape context is a sphere centered on the basis point p and its north pole oriented with the surface normal estimate $\mathcal{N}$ for p (Figure 2.1). The support region is divided into bins by equally spaced boundaries in the azimuth and elevation dimensions and logarithmically spaced boundaries along the radial dimension. We denote the $J+1$ radial divisions by $R = \{R_0 \dots R_J\}$, the $K+1$ elevation divisions by $\Theta = \{\Theta_0 \dots \Theta_K\}$, and the $L+1$ azimuth divisions by $\Phi = \{\Phi_0 \dots \Phi_L\}$. Each bin corresponds to one element in the $J \times K \times L$ feature vector. The first radius division R_0 is the minimum radius $r_{\min}$, and R_J is the maximum radius $r_{\max}$. The radius boundaries are calculated as

$$R_j = \exp\left\{ \ln(r_{\min}) + \frac{j}{J} \ln\left(\frac{r_{\max}}{r_{\min}}\right) \right\}. \tag{2.1}$$

Sampling logarithmically makes the descriptor more robust to distortions in shape with distance from the basis point. Bins closer to the center are smaller in all three spherical dimensions, so we use a minimum radius ($r_{\min} > 0$) to avoid being overly sensitive to small differences in shape very close to the center. The Θ and Φ divisions are evenly spaced along the 180° and 360° elevation and azimuth ranges.

Bin(j, k, l) accumulates a weighted count $w(p_i)$ for each point p_i whose spherical coordinates relative to p fall within the radius interval $[R_j, R_{j+1})$, azimuth interval $[\Phi_k, \Phi_{k+1})$ and elevation interval $[\Theta_l, \Theta_{l+1})$. The contribution to the bin count for point p_i is given by

$$w(p_i) = \frac{1}{\rho_i \sqrt[3]{V(j, k, l)}} \tag{2.2}$$

where $V(j, k, l)$ is the volume of the bin and ρ_i is the local point density around the bin. Normalizing by the bin volume compensates for the large variation in bin sizes with radius and elevation. We found empirically that using the cube root of the volume retains significant discriminative power while leaving the descriptor robust to noise which causes points to cross over bin boundaries. The local point density ρ_i is estimated as the count of points in a sphere of radius δ around p_i. This normalization accounts for variations in sampling density due to the angle of the surface or distance to the scanner.

We have a degree of freedom in the azimuth direction that we must remove in order to compare shape contexts calculated in different coordinate systems. To account for this, we choose some direction to be Φ_0 in an initial shape context, and then rotate the shape context about its north pole into L positions, such that each Φ_l division is located at the original 0° position in one of the rotations. For descriptor data sets derived from our reference scans, L rotations for each basis point are included. Since we are rotating the reference features, we do not need to rotate the query features. We could just as easily rotate the query features instead, but it should become clear why we rotate the reference features when we discuss our use of LSH later in the

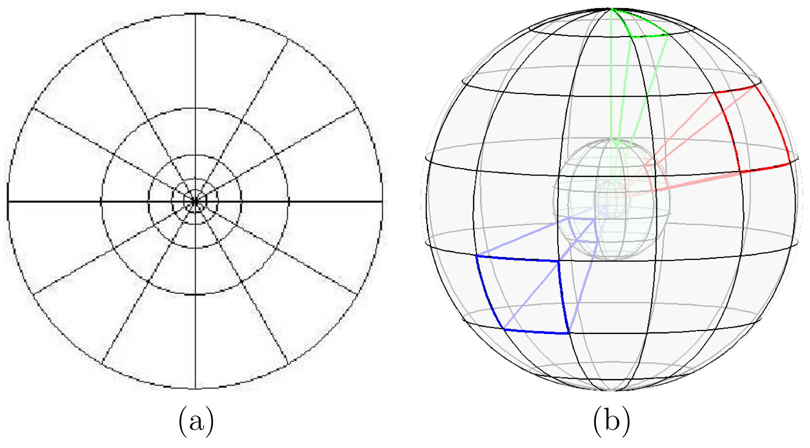


Figure 2.1: Example templates for the shape contexts: (a) for 2D, (b) for 3D. The number of divisions shown are not the same as we used in our experiments.

chapter.

In all experiments in this chapter, we use the same set of parameters in computing the 3D shape context features. The maximum radius of the spherical support region is 2.5 meters, and the minimum is 0.1 m, meaning that points closer than 0.1 m to the basis point are not counted in the descriptor. The volume is divided into fifteen equal parts in the radial dimension, eleven along the elevation, and twelve along the azimuth, giving the final features 1,980 dimensions. These values were chosen after a small amount of experimentation with a similar data set using different models.

Spin images, another descriptor used for 3D object recognition presented in [Johnson and Hebert, 1999], is very similar to the 3D shape context. It differs primarily in the shape of its support volume and its approach to the azimuth degree of freedom in the orientation: the spin image sums the counts over changes in azimuth. In Section 2.9, we directly compare the 3D shape context to the spin image descriptor.

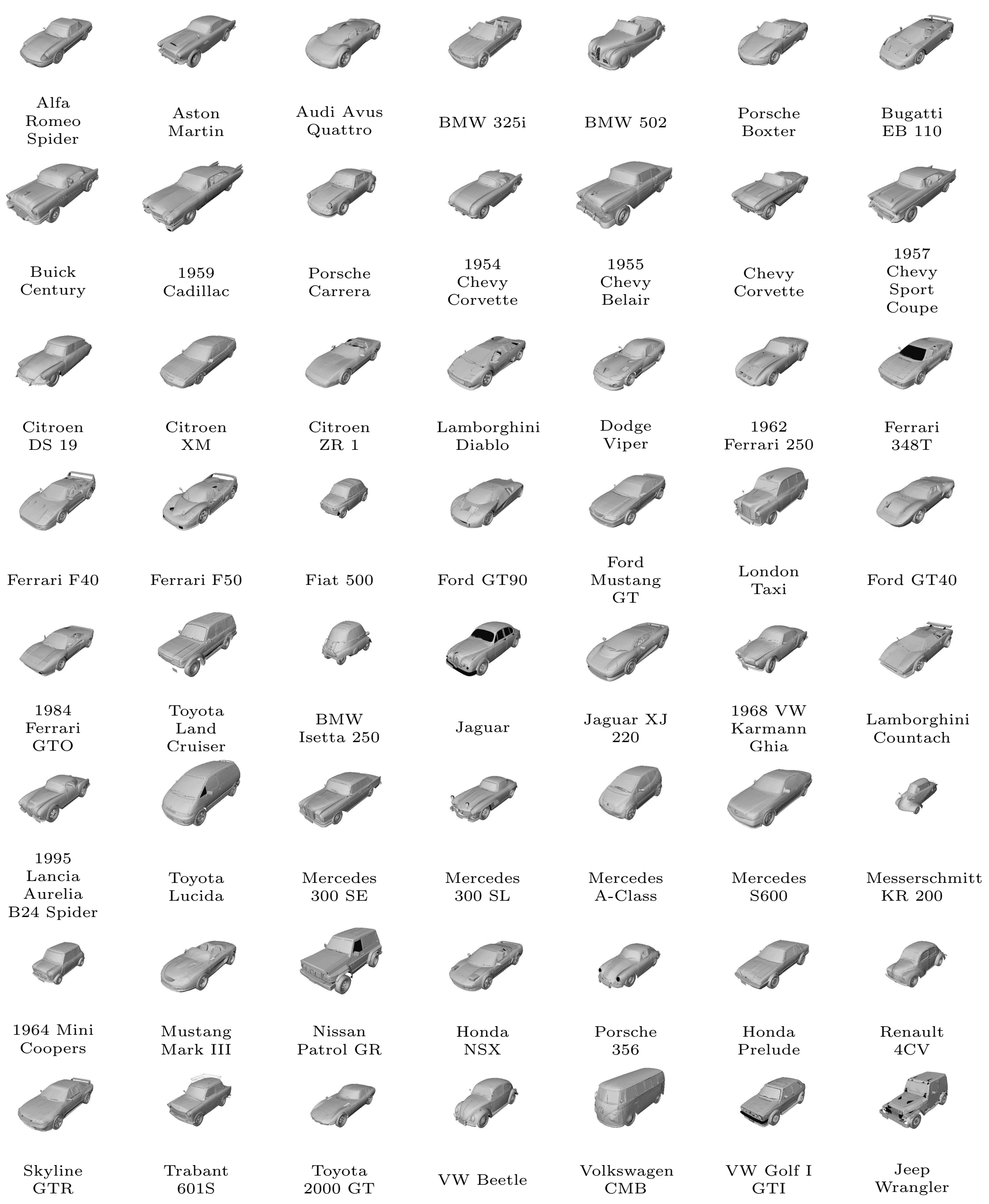


Figure 2.2: The fifty-six car models used in our experiments.

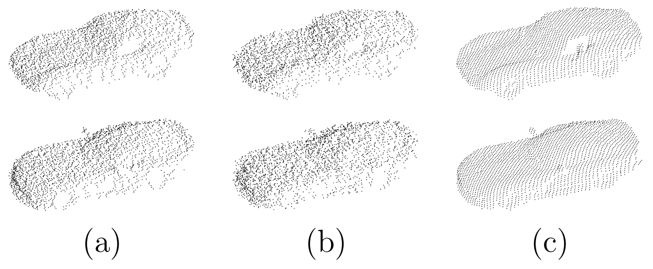


Figure 2.3: The *top row* shows scans from the 1962 Ferrari 250 model, and the *bottom* scans are from the Dodge Viper. The scans in column *(a)* are the query scans at 30 degrees elevation and 15 degrees azimuth with $\sigma = 5$ cm noise, and those in *(b)* are from the same angle but with $\sigma = 10$ cm noise. With 10 cm noise, it is difficult to differentiate the vehicles by looking at the 2D images of the point clouds. Column *(c)* shows the reference scans closest in viewing direction to the query scans (45 degrees azimuth and 45 degrees elevation).

2.2.3 Experiments with Three-dimensional Shape Contexts

In this subsection, we introduce the data set that we use throughout the chapter to evaluate recognition with 3D shape contexts. The range scans from which we calculate the features are simulated from a set of fifty-six 3D car models, and are separated into reference scans (our training set) and query scans. The full models are shown in Figure 2.2. The reference scans were generated from a viewpoint at 45 degrees elevation (measured from the horizon) and from four different azimuth positions, spaced 90 degrees apart around the car, starting from an angle halfway between the front and side views of the vehicle. The query scans were generated from a viewpoint at 30 degrees elevation and at one azimuth position 15 degrees different from the nearest reference scan. We also added Gaussian noise to the query scans along the viewing direction, with either a 5 cm or 10 cm standard deviation. This amount of noise is comparable to or greater than the noise one could expect from a quality scanner. An example of the noisy query scans next to the nearest reference scan for two of the car models is shown in Figure 2.3.

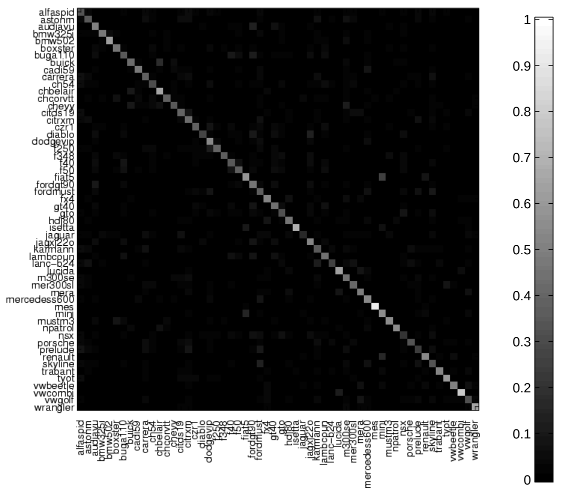
From the reference scans, we calculated normals at the points, and calculated 3D shape context features at basis points sampled uniformly over the surface, an average of 373 features per scan. For each noisy query scan, we calculated the normals, then calculated features at 300 randomly chosen basis points. Now we can describe our first experiment.

2.3 Basic Matching Experiment

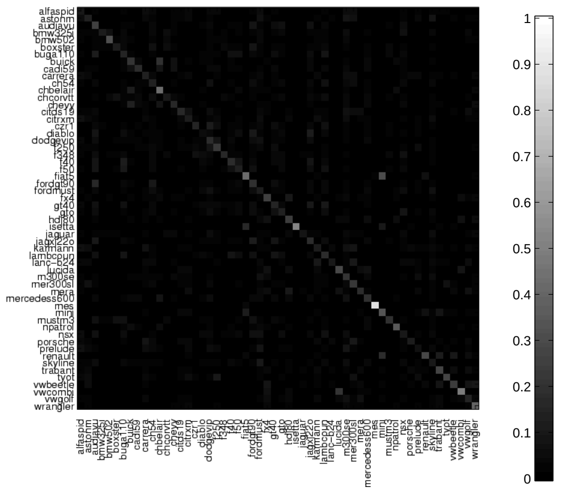
Experiment 1

Given a query scan, we want to return the best match from among the reference scans. Each of the 300 query features from the query scan casts a "vote" for one of the fifty-six car models, and the best match to the query scan as a whole is the model which received the most votes. We determine a query feature's vote by finding its nearest neighbor from among the reference features, and awarding the vote to the model that produced that reference feature. We could also give the n best matches by ordering the models by the number of votes received, and returning the top n from that list. We run this procedure for all fifty-six query scans and calculate the recognition rate as the percentage of the fifty-six query scans which were correctly identified.

The results we get are shown as confusion matrices in Figure 2.4 for the 5 cm and 10 cm queries. Each row corresponds to the results for one query scan, and each column to one car model (four reference scans). Each square is a color corresponding to the number of votes that the query gave for the model. If every query feature voted for the correct model, then the matrix would have a dark red diagonal and otherwise be dark blue. Perfect recognition is achieved when the diagonal has the largest number from each row, which is the case here for the 5 cm noise data set. In the 10 cm experiment, we got fifty-two out of fifty-six queries correct, giving a



(a)



(b)

Figure 2.4: Confusion matrices for experiment 1 for *(a)* 5 cm and *(b)* 10 cm noise queries. Each row corresponds to one query and each column to one reference model. A square in the matrix represents the percentage of votes for the column's reference models by the row's query, where each row sums to 100%. The scale at the far right maps the colors to numbers. The strong diagonal in *(a)* means that most of the votes for each 5 cm noise query went to the correct corresponding model, giving us 100% recognition in the top choice. There was more confusion in the 10 cm query, with fifty-two of the fifty-six models correctly identified in the top choice, and 100% recognition within the top four choices.

recognition rate of 92.86%. The correct model is always in the top four matches, so if we are want a short list of depth four or greater, then our recognition is 100%.

2.3.1 Complexity and Computation Time

Take

- m to be the number of reference images (assume one object per reference image),
- n_r the number of features calculated per reference image,
- n_q the number of features calculated per query image,
- d the dimensionality of the features,
- p the number of pixels or points in the query scene, and
- s the number of scales over which we need to search.

Let us first look at the cost of computing each query feature. For the 2D shape context, we need to compute edge features at all the pixels that may lie in one of the descriptors' support, and then count the number of edge pixels in each bin. This gives us a preprocessing cost of $O(p)$ and a computation cost of $O(p)$ for each query feature, for a total of $O(p) + O(p \cdot n_q)$ for the query image as a whole.

For the 3D shape context, we do not need to preprocess all the points in the scan, just the neighborhood around the basis point to get the normal at that point. We still need to look through the points in the scene to calculate the bin contents, giving a cost of $O(p \cdot n_q)$.

Once we have the query features, we need to search through the $m \cdot n_r$ reference features. If we are performing an exact nearest-neighbor search as in experiment 1, we need to calculate the distance between each of those reference features and each of the query features. The cost for that is $O(m \cdot n_r \cdot n_q \cdot d)$. If we are working with 2D shape contexts, then we may also have to search over scale, increasing the cost to $O(m \cdot n_r \cdot n_q \cdot d \cdot s)$.

For the 3D shape contexts, this gives us a total cost of $O(p \cdot n_q) + O(m \cdot n_r \cdot n_q \cdot d)$. In experiment 1, $n_q = 300$, $m = 224$, $n_r = 4476$ (average of 373 features per reference scan times the twelve rotations through the azimuth for each), and $d = 1980$ ($11 \times 12 \times 15$), so the second term sums to 5.96×10^{11} pairs of floating point numbers we need to examine in our search. On a 1.3 GHz 64-bit Itanium 2 processor, the comparison of 300 query features to the full database of reference features takes an average of 3.3 hours, using some optimization and disk blocking. The high recognition rate we have seen comes at a high computational cost.

The rest of this chapter focuses on reducing the cost of computing these matches, first by reducing n_q using the *representative descriptor method* and then by reducing n_r using LSH. The voting results for $n_q = 300$ using exact nearest neighbor provides a baseline for performance, to which we will compare our results.

2.4 Reducing Running Time with Representative Descriptors

If we densely sample features from the reference scans (i.e., choose a large n_r), then we can sparsely sample basis points at which to calculate features from query scans. This is the case for a few reasons.

- Because the features are fuzzy, they are robust to small changes due to noise, clutter, and shift in the center point location. This makes it possible to match a feature from a reference object and a feature from a query scene even if they are centered at slightly different locations on the object or are oriented slightly differently. This also affects how densely we need to sample the reference object.

- Since regional descriptors describe a large part of the scene in fuzzy terms and a small part specifically, few are needed to describe a query scene well.
- Finally, these features can be very discriminative. Even with the data set we use below where we are distinguishing between several very similar objects, the features are descriptive enough that only a few are enough to tell apart very similar shapes.

We make use of these properties via a method originally introduced in in [Mori *et al.*, 2001] as *representative shape contexts*, which speeds search of 2D shape contexts. In this and previous work, we refer to the method as the *representative descriptor* method, since it also applies to the use of other descriptors, such as spin images. Each of the few features calculated from the query scene is referred to as a *representative descriptor* or *RD*. What we refer to as the *representative descriptor method* really involves four aspects:

1. Using a reduced number of query points as centers for query features
2. A method for choosing which points to use as representative descriptors
3. A method for calculating a score between an RD and a reference object
4. A method for aggregating the scores for the RDs to give one score for the match between the query scene and the reference object

In our experiments, we try a range of values for the number of RDs and find that for simple matching tasks (e.g., low-noise queries), few are needed to achieve near-perfect performance. As the matching task becomes more difficult, the number required to get a good recognition rate increases.

We choose the basis points for the RDs uniformly at random from the 300 basis points from the query scans. This is probably the least sophisticated way to make the choice, and we do so to provide a baseline. Instead, we could use an interest operator such as those used with SIFT descriptors.

We take the score between one RD and a particular car model to be the smallest distance between the RD and a feature from one of the four reference scans for the model. To calculate the score between the query scene as a whole and the model, we sum the individual RD scores for that model. The model with the smallest summation is determined to be the best match. We have found this summation to be superior to the "voting" method where we take a maximum over the scores; the individual distances give a notion of the quality of the match, and summing makes use of that information, whereas taking a maximum discards it.

2.4.1 Experiment and Results

Experiment 2

Calculate n_q features from the query scan, which will be our RDs. Find the nearest neighbors to each of the RDs from each of the models, and calculate the scores. The model with the smallest score is the best match. Repeat for all queries and calculate the recognition rate as the percentage of query models that were correctly matched. Repeat the experiment several times with different randomly chosen sets of n_q features, and report the average recognition rate across these runs. Perform the experiment for different values of n_q.

The graphs in Figure 2.5 show the results. Note that the number of comparisons increases linearly with the number of RDs. For example, if the voting method with 300 query features required n comparisons, then using thirty RDs requires $n \times \frac{30}{300}$

comparisons. With the 5 cm queries, we achieve 100% recognition with thirty descriptors if we consider the top seven matches. If we use forty RDs, we achieve 99.9% in the top two matches and 100% in the top three. The performance on the 10 cm noise query degrades quickly with fewer RDs. Because of the noise, fewer of the original 300 query points are useful in matching, so we randomly choose more RDs in the hopes that we will get more of the distinctive query features. With the 10 cm queries, we achieve 97.8% mean recognition in the top seven results using eighty RDs. The mean recognition within the top seven with 160 RDs is 98%.

When we consider only our top match, our performance has dropped significantly with both query sets. However, we are primarily interested in getting a *short list* of candidates, and for the 5 cm queries we can reduce the number of computations required by 87% to 90% (depending on the length of our short list) by using the RD method over voting. And for almost all choices of the forty RDs, we find the correct match in the top five returned. With the 10 cm set, we can reduce our computation by 47% to 73%. Also keep in mind that these are recognition rates averaged across 100 different random selections of the RDs; for many choices of the RDs we are achieving perfect recognition.

2.5 Reducing Search Space with a Locality-Sensitive Hash

When comparing a query feature to the reference features, we could save computation by computing distances only to the reference features that are nearby. Call this the “1, 2, 3, many” philosophy: the few close ones play a large role in the recognition; the rest of the features have little meaning for the query. One way to achieve this is to use an algorithm for approximate k-NN search that returns a set of candidates

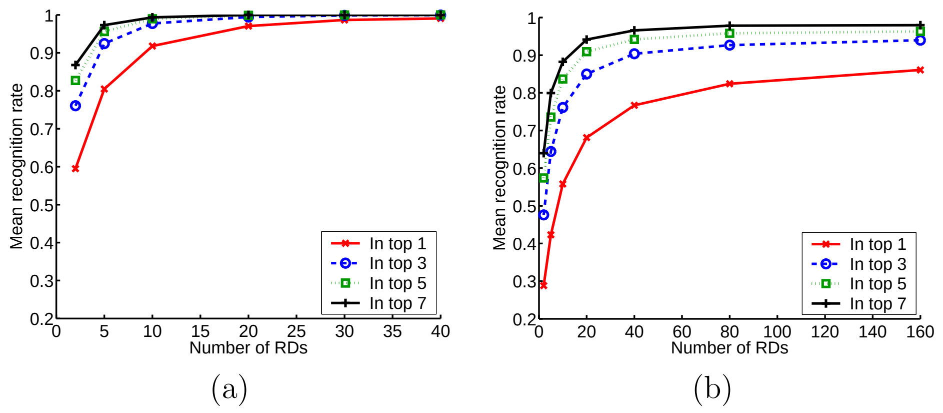


Figure 2.5: Results from experiment 2, shown as the number of RDs vs. mean recognition rate for the *(a)* 5 cm noise and *(b)* 10 cm noise queries. While our performance has dropped when considering only the top match, our recognition within a short list of matches is still very good, while we are performing a fraction of the feature comparisons. Note that the number of feature comparisons increases linearly with the number of RDs.

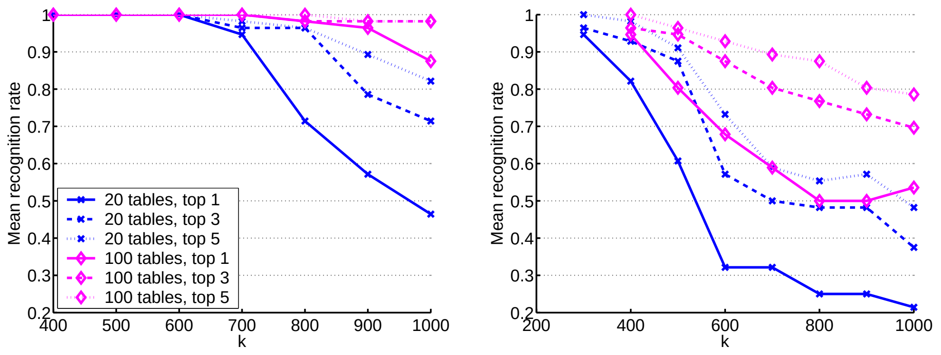
that probably lie close to the query feature. The method we will look at is Locality Sensitive Hashing (LSH), first introduced in [Indyk and Motwani, 1998].

We use a version of the simple LSH algorithm described in [Gionis *et al.*, 1999]. To create a hash, we first find the range of the data in each of the dimensions and sum them to get the total range. Then choose k values from that range. Each of those values now defines a cut in one of the dimensions, which can be visualized as a hyperplane parallel to that dimension's axis. These planes divide the feature space into hypercubes, and two features in the same hypercube hash to the same bucket in the table. We represent each hypercube by an array of integers, and refer to this array as the *first-level hash* or *locality-sensitive hash*. There are an exponential number of these hashes, so we use a standard second-level hash function on integer arrays to translate each to a single integer. This is the number of the bucket in the table, also called the *second-level hash* value. To decrease the probability that we will miss close neighbors, we create l tables, independently generating the k cuts in each. In most of our experiments in this section, we will use twenty tables. We will use the notation $b = h_i(\cdot)$ to refer to the hash function for the ith table which takes a feature vector and returns a second-level hash, or bucket, number. $T_i(b_i)$ will refer to the set of identifiers stored in bucket b_i in the ith table.

To populate the ith hash table, we calculate $b_i = h_i(f_j)$ for each feature f_j in the set of features calculated from the reference scans, and store the unique integer identifier j for the feature f_j in bucket b_i. Given a query feature q, we find matches in two stages. First, we retrieve the set of identifiers which are the union of the matches from the l tables: $\mathbb{F} = \bigcup_{i=1}^{l} T_i(h_i(q))$. Second, we retrieve from a database on disk the feature vectors for the identifiers, and calculate the distances $\text{dist}(q, f_j)$ for all features f_j where $j \in \mathbb{F}$.

The first part is the LSH query overhead, and in our experiments this takes 0.01 to 0.03 second to retrieve and sort all the identifiers from twenty tables. This is small compared to the time required in the second step, which ranges from an average of 1.12 to 2.96 seconds per query feature, depending upon the number of matches returned. Because the overhead for LSH is negligible compared to the time to do the feature comparisons, we will compare the "speed" of our queries across methods using the number of feature comparisons performed. This avoids anomalies common in timing numbers due to network congestion, disk speed, caching, and interference from other processes.

As we mentioned earlier, we are storing in the hash tables the azimuth rotations of the reference features instead of performing the rotations on the query features. If LSH returns only the features that are most similar to a query q, it will effectively select for us the rotations to which we should compare, which saves us a linear search over rotations.



5 cm noise 10 cm noise

Figure 2.6: Results for experiment 3 using the voting method with 300 query features. The graph shows the recognition rate versus the number of hash divisions (k) for 20 and 100 tables and for short lists of length one, three, and five (the legend applies to both graphs). The left and right graphs show results for the 5 cm and 10 cm noise queries, respectively. In general, as the number of hash divisions increases for a given number of tables, the performance degrades, and if the number of tables is increased, for a given value of k, performance increases. To see how the same factors affect the number of comparisons performed, see Figure 2.7. To visualize the tradeoff between the number of comparisons and recognition rate, see section 2.8.

2.5.1 LSH with Voting Method

We first examine the performance of LSH using the voting method from subsection 2.2.3 to provide a comparison with the strong results achieved using exact nearest neighbor.

Experiment 3

Given the number of hash divisions k and the number of LSH tables l, perform LSH search with 300 features per query, and tabulate the best matches using the voting scheme, as we did in experiment 1. Perform for 5 cm and 10 cm noise queries.

We created 100 tables, and ran experiments using all 100 tables as well as subsets of 20, 40, 60, and 80 tables. In Figure 2.6, we show how the recognition rate changes

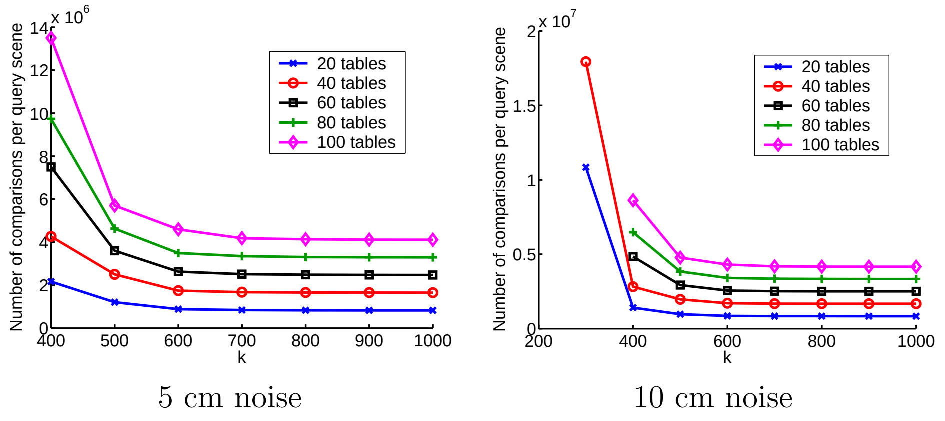
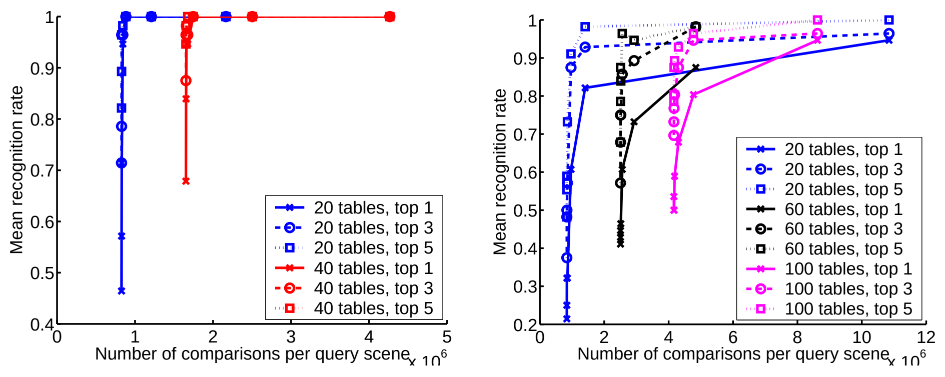


Figure 2.7: Results for experiment 3, showing the mean number of comparisons per query scene vs. the number of hash divisions (k), using 20, 40, 60, 80, or 100 tables. The left and right graphs show results for the 5 cm noise and 10 cm noise queries, respectively. The scale of the y-axis in the 10 cm graph is larger than in the 5 cm graph to accommodate the $k = 300$ results, though the number of comparisons required for each k and l combination is fewer with the 10 cm queries.

with variations in the number of hash divisions for the 5 cm and 10 cm queries. We show results for experiments with 20 and 100 tables, and show the recognition rate within the top choice, top three choices, and top five choices. In the 5 cm experiment, we maintain 100% recognition with up to 600 hash divisions when using twenty tables, and up to 800 hash divisions if we use 100 tables and consider a short list of length five. Notice that when using twenty tables, recognition degrades quickly as k increases, whereas recognition is better maintained when using 100 tables.

In the 10 cm experiments, we only achieve 100% recognition looking at the top five and using 300 or 400 hash divisions, with recognition declining quickly for larger values of k. Also notice that recognition falls with increasing k more quickly in the 10 cm experiments. As the queries become more difficult, it is less likely we will randomly generate a table with many divisions that performs well for many of our query features.

The recognition rate is only one measure of the performance. In Figure 2.7, we



5 cm noise   10 cm noise

Figure 2.8: Results for experiment 3, where we vary the value of k along each line to show the tradeoff between the number of comparisons performed and the mean recognition rate. The ideal point is in the upper-left corner where the number of comparisons is low and recognition is perfect. Exact nearest neighbor is off the graph in the upper-right corner, and would lie at $(3.0 \times 10^8, 1)$ if it were plotted.

show the mean number of comparisons per query scene vs. the number of hash divisions. Here we show results for 20, 40, 60, 80, and 100 tables. In both the 5 cm and 10 cm queries, we see a decline in the number of comparisons with increasing k, though the decline quickly becomes asymptotic. We also see a linear increase in the number of computations with a linear increase in the number of tables used.

For the 10 cm query, we tried using 300 hash divisions, but for more than forty tables, the queries were computationally too expensive. The range on the y-axis is larger in the 10 cm graph than in the 5 cm graph due to the jump at $k = 300$, but the number of computations performed for all other combinations of k and l are fewer in the 10 cm experiments. This seems to indicate that in general, the 10 cm query features lie farther away from the reference features in feature space than the 5 cm query features.

We see that as k decreases or the number of tables increases, the recognition improves, but the number of comparisons increases. To evaluate the trade off between speed and accuracy, we show in Figure 2.8 the number of comparisons vs. the recognition rate, varying k along each line. The ideal point would be in the upper-left corner, where the recognition rate is high and the number of comparisons is low. Exact nearest neighbor gives us a point at $(3.0 \times 10^8, 1)$, off the graph in the far upper-right corner. In the 5 cm graph, the leftmost point still at 100% recognition is from the experiment with 600 divisions and twenty tables. We can see that there is little to gain in increasing the number of divisions or the number of tables. The rightmost points in the 10 cm graph correspond to the experiments with 300 divisions, showing that the high recognition comes at a high computational cost. The points closest to the upper-left corner are from experiments using twenty tables and either 400 or 500 hash divisions. Unless we require perfect recognition for all queries, it makes little sense to use fewer than 400 divisions or more than twenty tables.

Lastly, while we are still achieving 100% mean recognition with $k = 600$ on the 5 cm queries using the voting method, the confusion matrix in figure 2.9 shows that we are not as confident about the matches relative to the confusion matrix for exact nearest neighbor (see Figure 2.4). The RD method depends upon having several distinguishing query features, so if we combine LSH with RD, we expect a decrease in the number of comparisons but also a further degradation in recognition performance.

2.5.2 Using RDs with LSH

Experiment 4

Perform LSH search with varying numbers of RDs, values of k, and numbers of tables. Using the model labels returned with each feature, tabulate scores as we did in experiment 2, with one exception: it is possible that LSH does not return any matches corresponding to a particular model, and in that case, we substitute for the RD model score a number larger than any of the distances as a penalty.

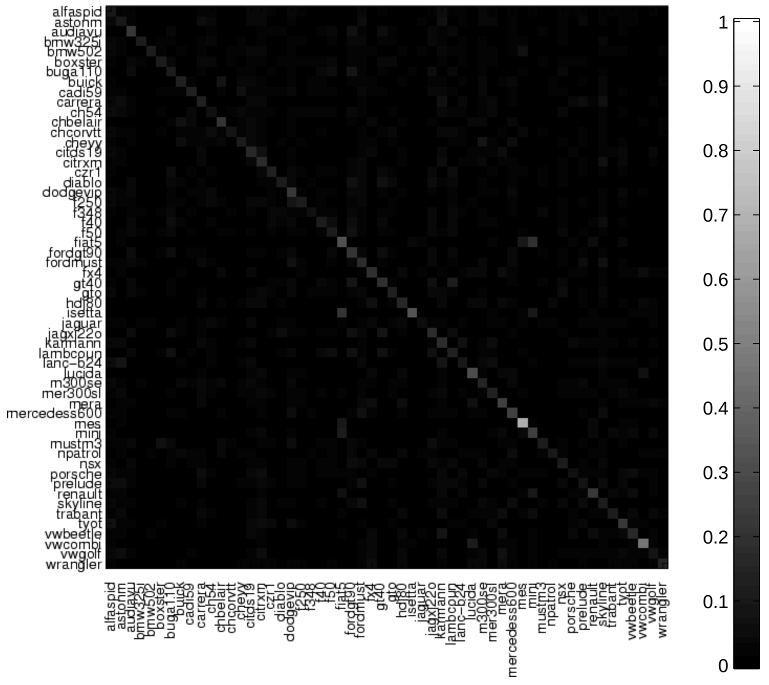
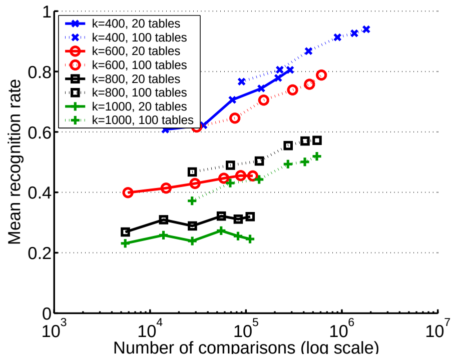
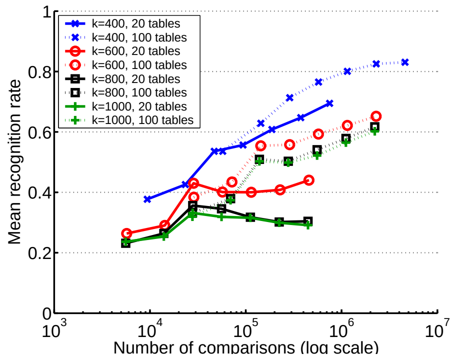


Figure 2.9: Confusion matrix showing the results for the 5 cm query for $k = 600$ from experiment 3. While we achieved 100% recognition with the top choice, comparing this matrix to the one from experiment 1 using exact nearest neighbor (see Figure 2.4) shows that we are less certain of our choices.



5 cm noise (top 5)



10 cm noise (top 7)

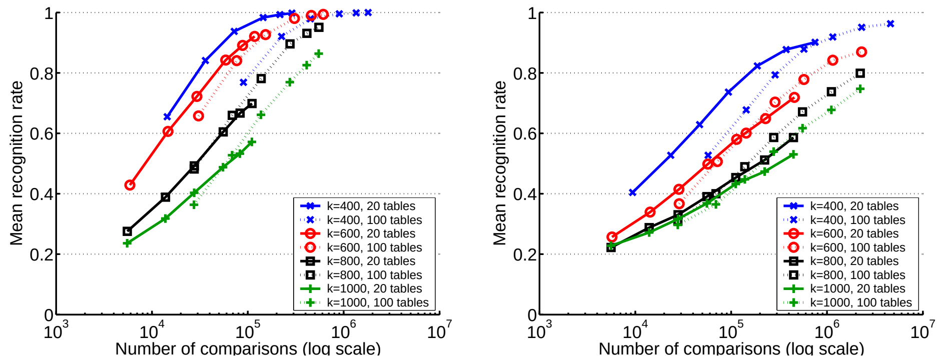
Figure 2.10: Results for experiment 4, showing number of comparisons vs. recognition rate with varying numbers of RDs along each line. We ran the experiment for different values of k and different numbers of tables. In the left graph we show the recognition within the top five results for the 5 cm queries, and in the right graph we show recognition within the top seven results for the 10 cm queries.

The tradeoff for experiment 4 between number of comparisons and mean recognition rate is shown in Figure 2.10. Along each line we varied the number of RDs, and show results for combinations of 400, 600, 800, and 1000 divisions and 20 and 100 tables. For the 5 cm experiment, we show recognition in a short list of five, and show results within the top seven for the 10 cm experiment. The mean recognition in the 5 cm experiment using 400 divisions and twenty tables reaches 80%, which is much worse than before. With $k = 600$ and twenty tables, which demonstrated a good tradeoff when we using the voting method with 300 query features, only a 45% mean recognition rate is achieved. We do see however, that increasing the number of tables has helped us; using $k = 400$ with 100 tables yields a mean recognition rate of 94%. We see similar degradation with the 10 cm experiments, achieving only 83% mean recognition withing the top seven results using 400 divisions and 100 tables.

Recognition performance is decreased when using LSH because LSH misses many of the nearest neighbors to the query points, resulting in a heavy penalty. We can improve performance by being more "forgiving" and including in the RD sum only the closest x percent of the RD model matches, hopefully discarding the large values that arise because LSH unluckily misses good matches. If we are using twenty RDs and we are summing the top 50%, then for a given model, we would search for the model's closest reference features to each of the twenty RDs, and include in the sum only the ten of those which are closest.

Experiment 5

We perform LSH search with varying numbers of RDs, values of k, and numbers of tables. We tally the RD scores by including in the sum the distances from only the best 50% of the RD model matches.



5 cm noise (top five) 10 cm noise (top seven)

Figure 2.11: Results from experiment 5, where we use the RD method but sum only the top half of the RD scores. The graphs show the number of comparisons vs. the mean recognition rate, with the number of RDs varying along each line. In the left graph we show the recognition within the top 5 results for the 5 cm queries, and in the right graph we show recognition with the top 7 results for the 10 cm queries. Note the logarithmic scale along the x-axis.

The results for experiment 5 in Figure 2.11 show that this method improved performance significantly within the top five results for 5 cm and top seven for 10 cm. In the 5 cm experiments, our sweet spot appears to be forty RDs, 400 divisions, and twenty tables with a mean recognition rate of 99.8% within the top five matches (and 95.3% with the top match; 99.4% within the top three, not shown). In the 10 cm experiments we reach 96% mean recognition with 160 RDs, 400 divisions, and 100 tables within the top seven matches (and 93.6% in the top five, not shown). We reach 90% mean recognition with 160 RDs, 400 divisions, and twenty tables within the top seven matches, which requires less than one-sixth the number of comparisons as with the same settings except with 100 tables.

The key to further improving performance lies primarily with getting better results from our approximate nearest-neighbor algorithm. In the next section, we examine the quality of the LSH results relative to exact nearest neighbor, and use this to motivate the need for algorithms that provide better nearest-neighbor performance.

2.6 Nearest-Neighbor Performance of LSH

In this section, we look at the performance of LSH as an approximate nearest-neighbor algorithm, independent of any recognition procedure. In most work on approximate nearest-neighbor algorithms, the performance is measured using the *effective distance error* or a similar measure [Liu *et al.*, 2004; Gionis *et al.*, 1999; Arya *et al.*, 1998], defined for the nth nearest neighbor as

$$E = \frac{1}{Q} \sum_{q \in Q} \left(\frac{d_{alg,n}}{d^*_n} - 1 \right), \tag{2.3}$$

where Q is the set of query features, d^*_n is the distance from the query q to the nth true nearest neighbor, and $d_{alg,n}$ is the distance from q to the nth best feature returned from the approximate algorithm. The effective distance error with increasing rank depth n is shown for the 5 cm and 10 cm queries in the first row of Figure 2.12. Each line of the graphs represents one LSH query with a different number of hash divisions

(k).

The effective distance error does not capture whether an approximate nearest-neighbor algorithm is returning the *correct* nearest neighbors, only how close it gets to them. In a recognition setting, the *identity* of the features returned is of primary interest, so we suggest a better measure would be *error by rank*. If we want the n nearest neighbors, the error by rank is the percentage of the true n nearest neighbors that were missing in the list of n returned by the approximate algorithm. The graphs in the second row of Figure 2.12 show the error by rank with increasing n for the 5 cm and 10 cm queries.

In the first column of Figure 2.12 we see that for the 5 cm query, the effective distance error reaches a maximum at 40% for 800, 900, and 1000 hash divisions, but the same LSH results show almost 100% error by rank, meaning that almost never are any of the correct nearest neighbors returned. The second column of the figure shows results for the 10 cm queries. Notice that, relative to the 5 cm queries, the ceiling on the effective distance error is actually lower; the 900 and 1000 hash division LSH queries level off around 0.32, and all queries except LSH with 400 and 500 hash divisions are actually performing better by this measure than in the 5 cm query. However, we know from our recognition results that this should not be the case, that recognition results for the 10 cm queries were worse than the 5 cm queries for the same LSH settings. Indeed, we can see in the error-by-rank graph that the 10 cm queries are performing much worse than the 5 cm queries for all LSH settings.

As an aside, the lines on these graphs are monotonically increasing, which does not have to be the case in general. If an approximate nearest-neighbor algorithm misses the first nearest neighbor, but then correctly finds every nearest neighbor of deeper rank, than the error by rank would decrease with increasing rank depth, from 100% to 50% to 33%, etc. It is also true that the effective distance error need not increase with increasing rank depth. It is a feature of LSH that we get fewer correct results as we look further from the query, which means that we cannot expect to increase our recognition performance by considering a greater number of nearest neighbors.

In Figure 2.13, we show the tradeoff for different numbers of tables and hash divisions (l and k). Each line corresponds to a fixed number of divisions, and we vary the number of tables along the line, with the largest number of tables at the rightmost point on each line. As expected, with a greater number of tables we see better performance but we also perform more comparisons.

In general, recognition performance should increase as error by rank decreases, though to what degree will depend upon the recognition algorithm and data set. Next we introduce a variant of LSH which will find the same or more of the nearest neighbors as LSH, but at a computational cost between LSH and exact nearest neighbor.

2.7 Associative LSH

In order to improve the error-by-rank and recognition performance, we introduce a variation which we will refer to as *associative LSH*. This algorithm begins with the results returned from LSH, and then uses the LSH tables to further explore the neighborhood around the query feature.

Consider the situation in Figure 2.14 where we have a query q and the closest point to it, p^*, where for all tables i, $h_i(q) \neq h_i(p^*)$. It may be the case that there exists a point p_0 such that for two different tables i and j, $h_i(q) = h_i(p_0)$ and $h_j(p_0) = h_j(p^*)$. This suggests that we could use p_0 to find p^*.

First, a word about the data structures necessary. We will need the l LSH hash tables. To speed the algorithm we will also use precomputed l reverse hashes $b_i = R_i(j)$, which take an integer feature identifier and return the bucket in the ith table in which it is stored. Note that this is the reverse of the $T_i(b_i)$ function. Note that these reverse hashes are not necessary since we could retrieve the feature f_j from disk and calculate $h_i(f_i)$.

Results will be written to a structure $\mathcal{R}$ that for each match found so far stores the integer feature identifier j and the distance to the query, $\text{dist}(q, f_j)$, sorted by distance. This is the same structure we used for results when performing LSH queries. We will keep lists of the numbers of the buckets we have visited, one for each of the tables. Call the ith of these lists $\mathcal{B}_i$. We will also have a set of integer identifiers $\mathcal{A}$ which is initially empty.

The algorithm takes as input a rank depth r and a query feature q and outputs the results structure $\mathcal{R}$. Notice that the first three steps below are identical to the original LSH algorithm as described earlier, with the exception of the use of $\mathcal{B}_i$ for record-keeping.

1. For all i, calculate $b_i = h_i(q)$. Add b_i to $\mathcal{B}_i$ so that we do not visit the bucket b_i in the ith table again.
2. Calculate $\mathbb{F} = \bigcup_{i=1}^{l} T_i(b_i)$.
3. For all $j \in \mathbb{F}$, calculate $\text{dist}(q, f_j)$ and add to the results list R.
4. Find a feature identifier that is within the top r results in $\mathcal{R}$ and that is not in the set $\mathcal{A}$, call it a. If such a feature does not exist, then terminate.
5. Add a to the set $\mathcal{A}$. This, with the check above, ensures that we do not iterate using this feature again.
6. For all i, find $b_i = R_i(a)$, the bucket in which a is stored in the ith table.
7. For all i where $b_i \notin \mathcal{B}_i$ (i.e., we have not already looked in bucket b_i in table i), retrieve $\mathbb{F} = \bigcup_i T_i(b_i)$, the identifiers in the buckets in which a resides.

8. For all i, add b_i to $\mathcal{B}_i$.

9. For each identifier $j \in \mathbb{F}$ that is not already in $\mathcal{R}$, calculate $\text{dist}(q, f_j)$ and store the result in $\mathcal{R}$.

10. Go to step 4.

This algorithm requires only one parameter, r, that LSH does not require. In our experiments, we did not tune r, setting it only to two. Setting it higher would result in more comparisons and perhaps better results. The data structures for $R_i(\cdot)$ are l arrays, each with an element for each reference feature stored in the LSH tables. This roughly doubles the amount of memory required to hold the LSH tables, though it does not need to be stored on disk as it can quickly be generated when the tables are loaded from disk. Note that any variation on LSH that randomly generates the hash divisions can be used with this method as well.

The running time of the algorithm is dependent upon the number of associative iterations performed and the number of features retrieved on each iteration. The additional bookkeeping required for associative LSH over regular LSH adds a negligible amount of overhead. Step 4 requires a $O(r)$ search through the results list and comparison with the hashed set $\mathcal{A}$, but r will be set to a small constant (two in our experiments). Steps 8 and 9 require additional bookkeeping using the structure $\mathcal{B}_i$, but the complexity in both cases is $O(l)$ if we make $\mathcal{B}_i$ a hashed set.

In Figure 2.15 we show the tradeoff between the number of comparisons performed and the error by rank for our associative LSH queries. We see a drop in the error by rank over regular LSH, especially when comparing results using the same number of hash divisions, but we see a corresponding increase in the number of comparisons.

In Figure 2.15 we show the tradeoff between comparisons and error by rank using associative LSH. Comparing to the results for LSH in figure 2.13 we see that we achieve a better tradeoff. For example, in the 5 cm experiments using 400 divisions, associative LSH achieves a slightly lower error and about the same number of comparisons using twenty tables as LSH does using eighty tables. Similarly, using 600 divisions, associative LSH achieves 65% error in 5×10^6 comparisons using twenty tables, whereas LSH reaches only 72% error in the same number of comparisons using 100 tables. From these results we can see that our search using associative LSH is more focused; we are finding a comparable number of nearest neighbors with associative LSH but with fewer comparisons. In the 10 cm experiments, this effect is more dramatic as associative LSH is able to achieve much lower error rates with a comparable number of comparisons.

Another important difference is that associative LSH is much less sensitive to the choices of k and the number of tables. With LSH, error changes dramatically with a change in the number of tables, and we see a quick degradation with an increase in the number of divisions.

2.8 Summary of 3D Shape Context Experiments

In this chapter, we have performed an analysis of methods for performing object recognition on a particular data set, with a focus on the tradeoff between the speed of computation and the recognition performance of the methods. We made use of LSH for improving the speed of our queries, and demonstrated ways in which it could be made more robust.

In Figure 2.16 we display as a scatterplot results from the different methods discussed earlier in the chapter on the 5 cm query data set. For each method, we show points for all the variations of number of RDs, number of hash divisions, and number of tables. In general, results for associative LSH using voting lie between LSH and exact nearest neighbor using voting, with the same true for all three methods using

RDs. Looking at the left graph showing results using the top choice, the best associative LSH results using RDs is close both in recognition and speed to LSH results using voting. If we can accept finding the match in the top three results returned, all the methods presented can get us to 100% recognition, with LSH with RDs achieving the lowest number of comparisons by a small margin over LSH with voting and associative LSH with RDs. We note again that the range of k using in the associative LSH experiments is much larger than in the LSH experiments, showing that we can achieve similar performance with less precise tuning of the parameters.

In Figure 2.17 we give a scatterplot of the results for the various 10 cm noise experiments. Again we see that the associative LSH results lie between LSH and exact nearest neighbor, though as we see in the first plot, LSH using 300 divisions and the voting method shows a slightly higher recognition rate and lower comparisons than associative LSH. In general, however, associative LSH yields a higher recognition rate than LSH, though by performing more comparisons. We also note that when using the voting method, the results for associative LSH are more tightly packed than the LSH results, despite using a wider range of parameters for associative LSH in the experiments. This indicates that associative LSH can yield similar results on this data set with less tuning of the parameters.

In conclusion, we have found that LSH is an effective method for speeding nearest-neighbor search in a difficult object recognition task, but at the cost of some recognition performance. We have touched upon the connection between the reduction in recognition performance and the performance of LSH as a nearest-neighbor algorithm, and have presented a variation, associative LSH, which gives an improvement in nearest-neighbor performance on our data set. This increase in nearest-neighbor performance translates only roughly into recognition performance, showing small gains in recognition performance on this data set for an additional computational cost.

2.9 Comparative Experiments

In this section we summarize experimentation from [Frome *et al.*, 2004], in which we systematically compared the performance of the 3D shape context, harmonic shape context, and spin images in recognizing similar objects in scenes with noise or clutter.

2.9.1 Harmonic Shape Context

To compute harmonic shape contexts, we begin with the histogram described above for 3D shape contexts, but we use the bin values as samples to calculate a spherical harmonic transformation for the shells and discard the original histogram. The descriptor is a vector of the amplitudes of the transformation, which are rotationally invariant in the azimuth direction, thus removing the degree of freedom.

Any real function $f(\theta, \phi)$ can be expressed as a sum of complex spherical harmonic basis functions Y_l^m.

$$f(\theta, \phi) = \sum_{l=0}^{\infty} \sum_{m=-l}^{m=l} A_l^m Y_l^m(\theta, \phi) \tag{2.4}$$

A key property of this harmonic transformation is that a rotation in the azimuthal direction results in a phase shift in the frequency domain, and hence amplitudes of the harmonic coefficients $\|A_l^m\|$ are invariant to rotations in the azimuth direction. We translate a 3D shape context into a harmonic shape context by defining a function $f_j(\theta, \phi)$ based on the bins of the 3D shape context in a single spherical shell $R_j \leq R < R_{j+1}$ as:

$$f_j(\theta, \phi) = SC(j, k, l), \theta_k < \theta \leq \theta_{k+1},\ \phi_l < \phi \leq \phi_{l+1}. \tag{2.5}$$

As in [Kazhdan *et al.*, 2003], we choose a bandwidth b and store only b lowest-frequency components of the harmonic representation in our descriptor, which is given by $HSC(l,m,k) = \|A^m_{l,k}\|,\ l,m = 0\ldots b, r = 0\ldots K$. For any real function, $\|A^m_l\| = \|A^{-m}_l\|$, so we drop the coefficients A^m_l for $m < 0$. The dimensionality of the resulting harmonic shape context is $K \cdot b(b+1)/2$. Note that the number of azimuth and elevation divisions do not affect the dimensionality of the descriptor.

Harmonic shape contexts are related to the rotation-invariant shape descriptors SH(f) described in [Kazhdan *et al.*, 2003]. One difference between those and the harmonic shape contexts is that one SH(f) descriptor is used to describe the global shape of a single object. Also, the shape descriptor SH(f) is a vector of length b whose components are the energies of the function f in the b lowest frequencies: $\mathrm{SH}_l(f) = \|\sum_{m=-l}^{l} A^m_l Y^m_l\|$. In contrast, harmonic shape contexts retain the amplitudes of the individual frequency components, and, as a result, are more descriptive.

2.9.2 Spin Images

We compared the performance of both of these shape context-based descriptors to spin images [Johnson and Hebert, 1999]. Spin-images are well-known 3D shape descriptors that have proven useful for object recognition [Johnson and Hebert, 1999], classification [Ruiz-Correa *et al.*, 2003], and modeling [Huber and Hebert, 2003]. Although spin-images were originally defined for surfaces, the adaptation to point clouds is straightforward. The support region of a spin image at a basis point p is a cylinder of radius $r_{\max}$ and height h centered on p with its axis aligned with the surface normal at p. The support region is divided linearly into J segments radially and K segments vertically, forming a set of $J \times K$ rings. The spin-image for a basis point p is computed by counting the points that fall within each ring, forming a 2D histogram. As with the other descriptors, the contribution of each point q_i is weighted by the inverse of that point's density estimate (ρ_i); however, the bins are not weighted by volume. Summing within each ring eliminates the degree of freedom along the azimuth, making spin-images rotationally invariant. We treat a spin-image as a $J \times K$ dimensional feature vector.

2.9.3 Experiments with 5cm noise

This subsection and following two subsections describe the experiments comparing 3D shape contexts, harmonic shape contexts, and spin images. The parameters for the 3D shape contexts are the same as in the previous sections, and Table 2.9.3 shows the parameters used for the other two types of descriptor. The parameters were chosen based on extensive experimentation on other sets of 3D models not used in these experiments. However, some parameters (specifically K and $r_{\min}$) were fine-tuned using descriptors in 20 randomly selected models from our 56 vehicle database. The basis points used for training were independent from those used in testing. The relative scale of the support regions was chosen to make the volume encompassed comparable across descriptors. The noisy data sets used in these experiments are also the same as those used for the previous experiments in this chapter.

In this set of experiments, our query data was a set of 56 scans, each containing one of the car models. We added Gaussian noise to the query scans along the scan viewing direction with a standard deviation of 5 cm (Fig. 2.3). The window for computing normals was a cube 55 cm on a side. Fig. 2.18 shows the mean recognition rate versus number of RDs. All of the descriptors perform roughly equally, achieving close to 100% average recognition with 40 RDs.

2.9.4 Experiments with 10cm noise

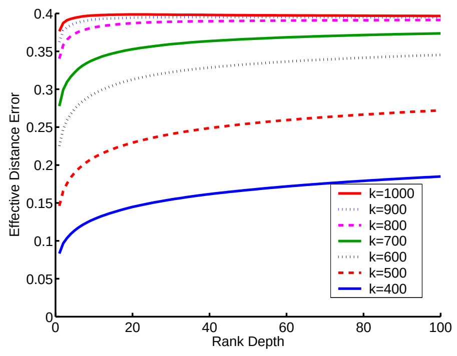
We performed two experiments with the standard deviation increased to 10 cm (see Fig. 2.3). In the first experiment, our window size for computing normals was the same as in the 5 cm experiments. The results in Fig. 2.9.4 show a significant decrease in performance by all three descriptors, especially spin images. To test how much the normals contributed to the decrease in recognition, we performed a second experiment with a normal estimation window size of 105 cm, giving us normals more robust to noise. The spin images showed the most improvement, indicating their performance is more sensitive to the quality of the normals.

2.9.5 Experiments with clutter

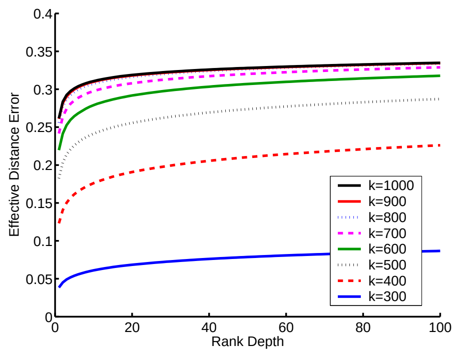
To test the ability of the descriptors to handle a query scene containing substantial clutter, we created scenes by placing each of the vehicle models in the clutter scene shown in Figure 2.20. We generated scans of each scene from a 30° declination and two different azimuth angles ($\phi = 150°$ and $\phi = 300°$), which we will call views #1 and #2, shown in Figure 2.21. We assume that the approximate location of the target model is given in the form of a box-shaped volume of interest (VOI). The VOI could be determined automatically by a generic object saliency algorithm, but for the controlled experiments in this paper, we manually specified the VOI to be a 2 m $\times$ 4 m $\times$ 6 m volume that contains the vehicle as well as some clutter, including the ground plane (Figure 2.22). Basis points for the descriptors were chosen from within this VOI, but for a given basis point, all the scene points within the descriptor's support region were used, including those outside of the VOI.

We ran separate experiments for views 1 and 2, using 80 RDs for each run. When calculating the representative descriptor cost for a given scene-model pair, we included in the sum only the 40 smallest distances between RDs and the reference descriptors for a given model. This acts as a form of outlier rejection, filtering out many of the basis points not located on the vehicle. We chose 40 because approximately half of the basis points in each VOI fell on a vehicle. The results are shown in Fig. 2.23.

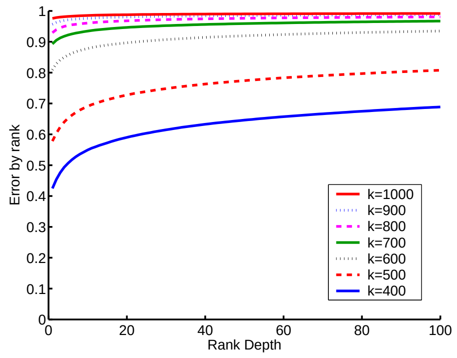
The shape context performance is impressive given that this is a result of doing naïve point-to-point matching without taking geometric constraints into account. Points on the ground plane were routinely confused for some of the car models which geometric constraints could rule out. A benefit of the 3D shape context over the other two descriptors is that a point-to-point match gives a candidate orientation of the model in the scene which can be used to verify other point matches.



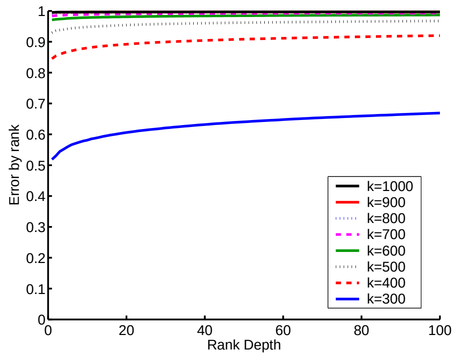
(a)



(c)

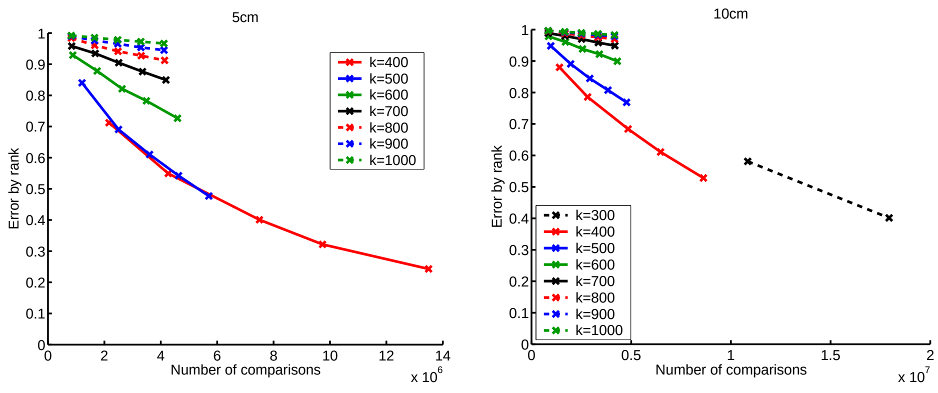


(b)
5 cm noise



(d)
10 cm noise

Figure 2.12: LSH performance, relative to exact nearest neighbor. The graphs in the first column show the performance on the 5 cm queries, using effective distance error in (a) and error by rank in (b). The second column shows results for the 10 cm query, with (c) showing effective distance error and (d) showing error by rank. All results are for twenty tables.



5 cm noise 10 cm noise

Figure 2.13: Nearest-neighbor performance of LSH, shown as the tradeoff between the number of comparisons and the error-by-rank for the 5 cm and 10 cm query sets. The lower-right corner of the graph is the ideal result, where the number of comparisons and the error by rank are low. The number of tables used is varied from 20 to 100 along each line. With 400 divisions, we drive down the error by rank, but also dramatically increase the number of comparisons required.

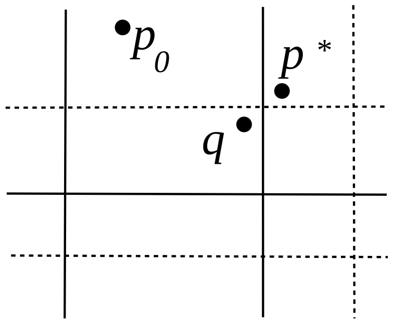


Figure 2.14: A 2D LSH example showing the space divided into bins by axis-parallel lines. The solid lines represent the divisions from one hash table, and the dashed lines represent divisions from another. Note that although p^* is the nearest neighbor to q, they do not occupy the same bin in either of the tables. It is the case, however, that p^* can be reached from q: q and p_0 are binmates in the solid-line table and p_0 and p^* are binmates in the dashed-line table.

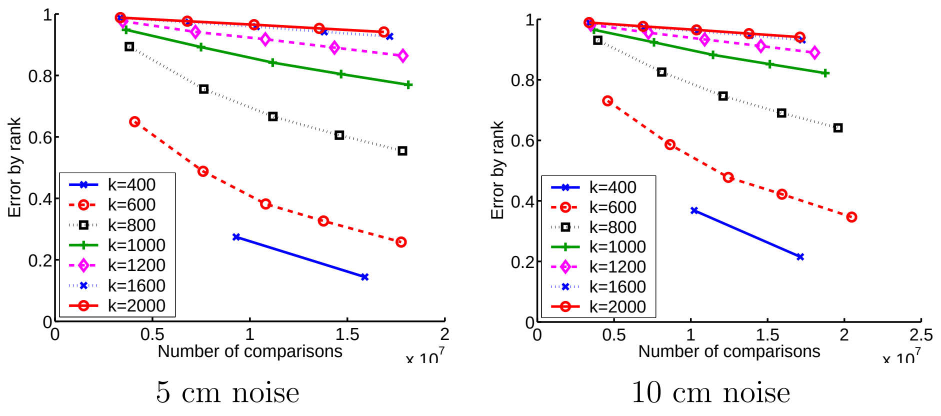


Figure 2.15: Nearest-neighbor performance of associative LSH, shown as the tradeoff between the number of comparisons and the error by rank for the 5 cm and 10 cm query sets. Compare these graphs to those in figure 2.13 showing nearest-neighbor performance of LSH. The number of tables used is varied from 20 to 100 along each line.

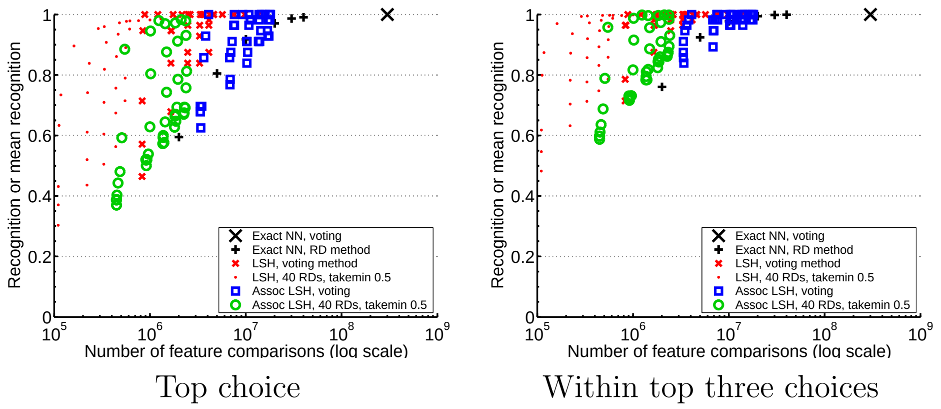
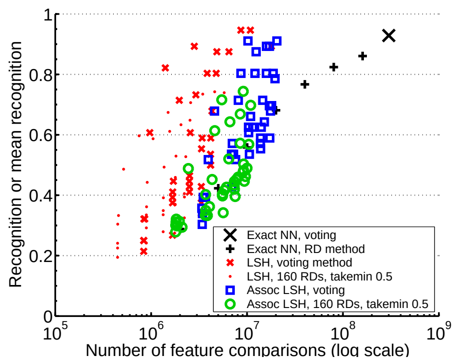
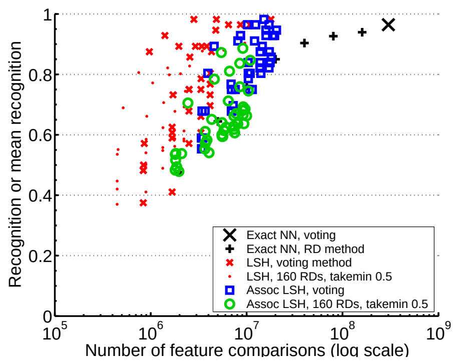


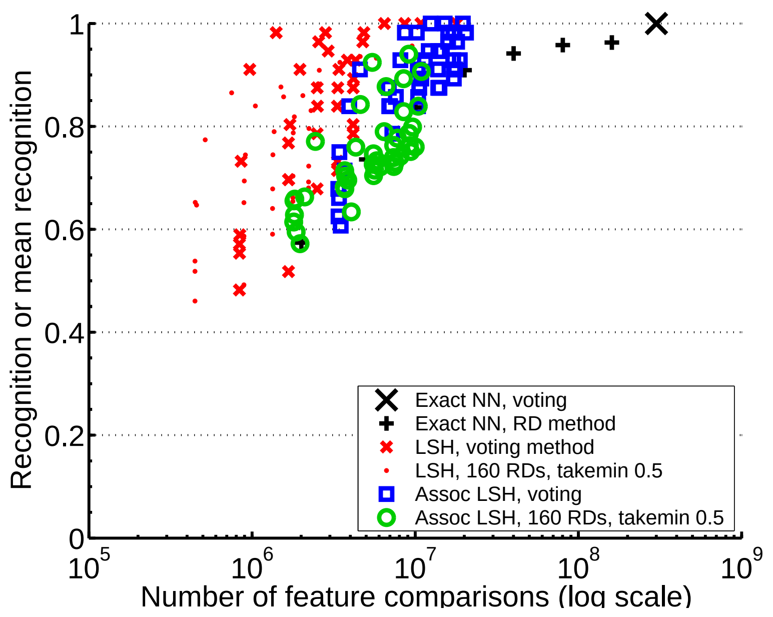
Figure 2.16: Summary of results for various methods on the 5 cm noise data set. For each method, we show points for all the variations of number of RDs, number of hash divisions, and number of tables discussed earlier in the chapter.



Top choice



Top three choices



Top five choices

Figure 2.17: Summary of results for various methods on the 10 cm noise data set, showing results for the top choice, top three, and top five choices. For each method, we show points for all the variations of number of RDs, number of hash divisions, and number of tables discussed earlier in the chapter.

	SC	HSC	SI
max radius ($r_{\max}$)	2.5	2.5	2.5
min radius ($r_{\min}$)	0.1	0.1	-
height (h)	-	-	2.5
radial divisions (J)	15	15	15
elev./ht. divisions (K)	11	11	15
azimuth divisions (L)	12	12	-
bandwidth (b)	-	16	-
dimensions	1980	2040	225
density radius (δ)	0.2	0.2	0.2

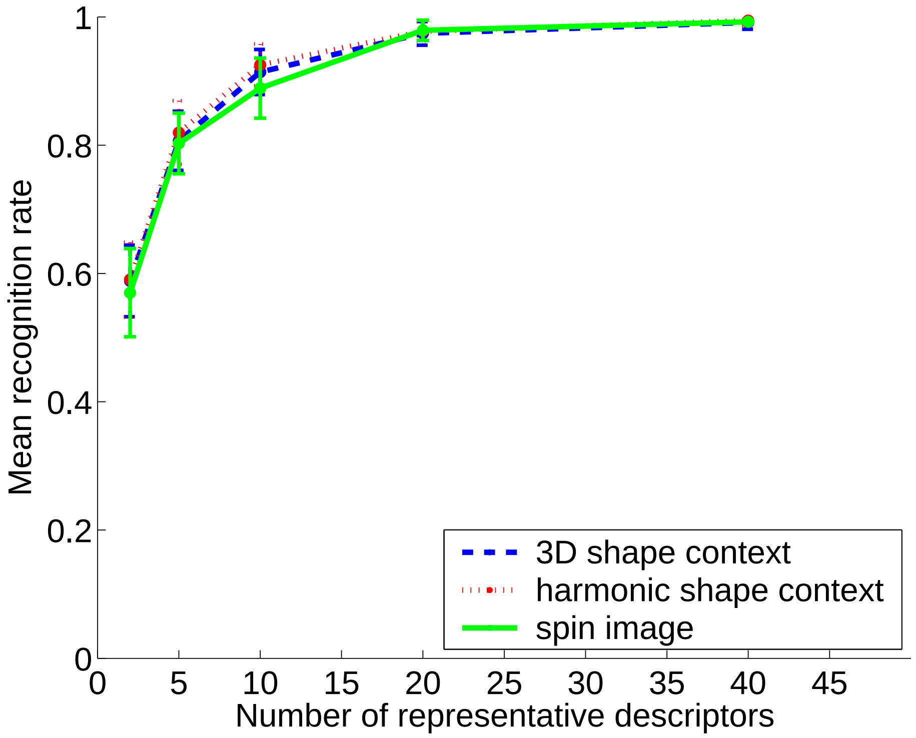
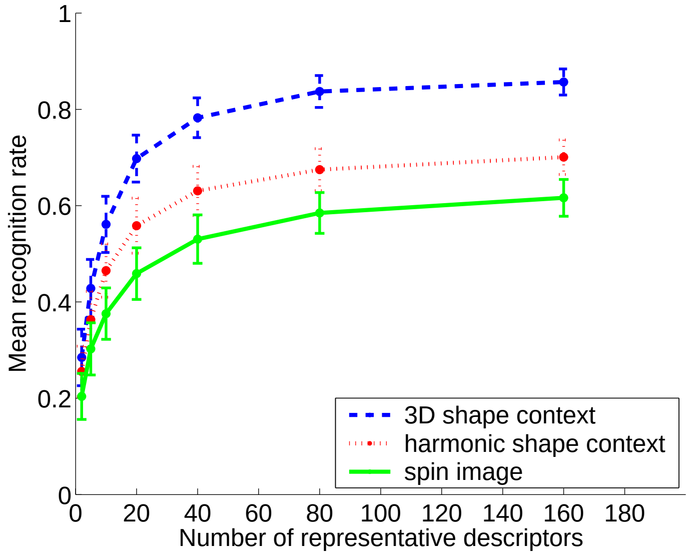
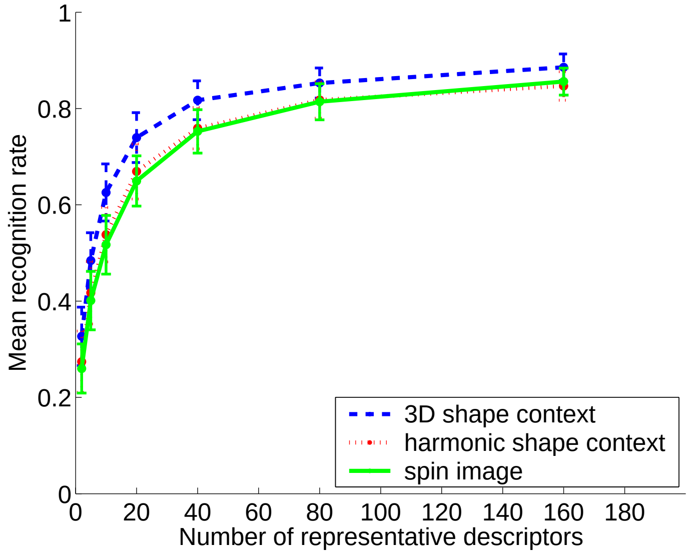


Figure 2.18: Results for the 5cm noise experiment. All three methods performed roughly equally. From 300 basis points sampled evenly from the surface, we chose varying numbers of RDs, and recorded the mean recognition rate. The error bars show one standard deviation.



(a)



(b)

Figure 2.19: Results for 10 cm noise experiments. In experiment (a) we used a window for the normals that was a cube 55 cm on a side, whereas in (b) the size was increased to a cube 105 cm on a side. The error bars show one standard deviation from the mean. From this experiment, we see that shape contexts degrade less as we add noise and in particular are less sensitive to the quality of the normals than spin images. All three methods would benefit from tuning their parameters to the higher noise case, but this would entail a recalculation of the reference set. In general, a method that is more robust to changes in query conditions is preferable.

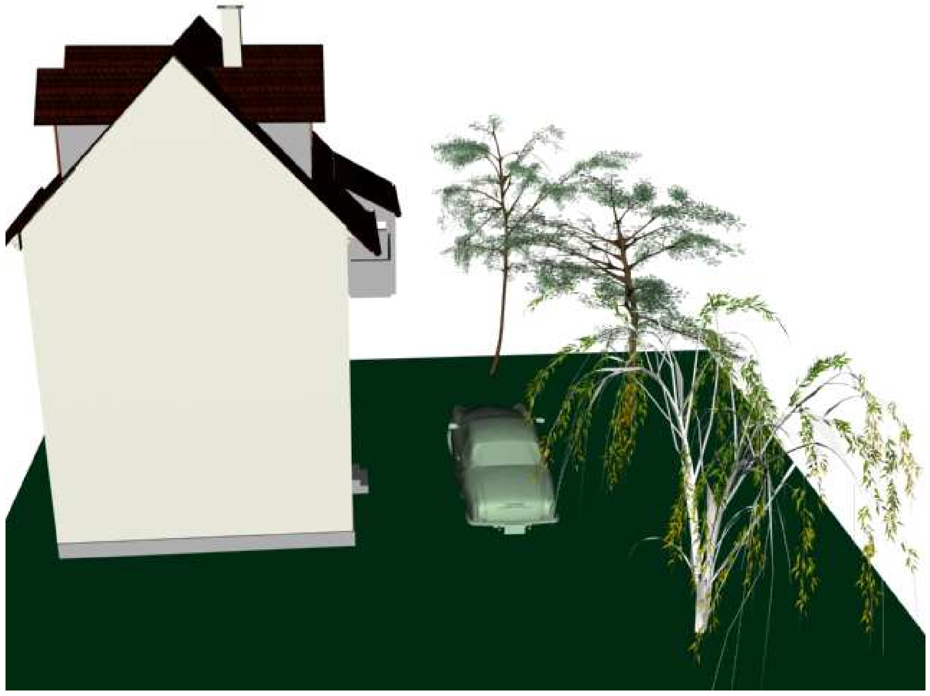
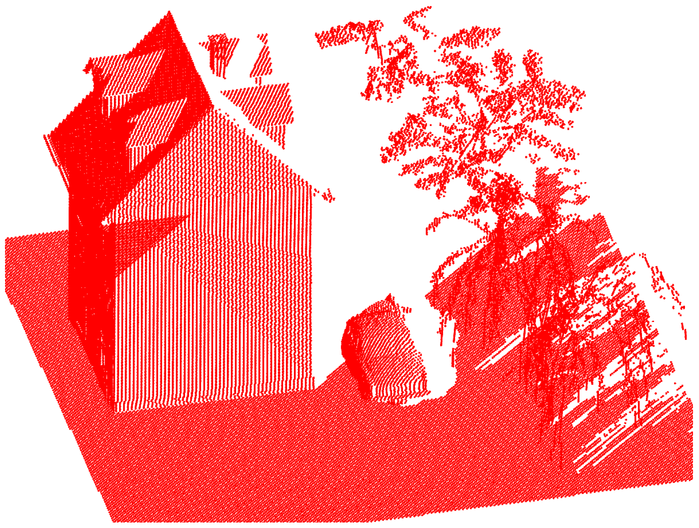


Figure 2.20: An example of a cluttered scene containing trees, a house, the ground, and a vehicle to be recognized.


(a)

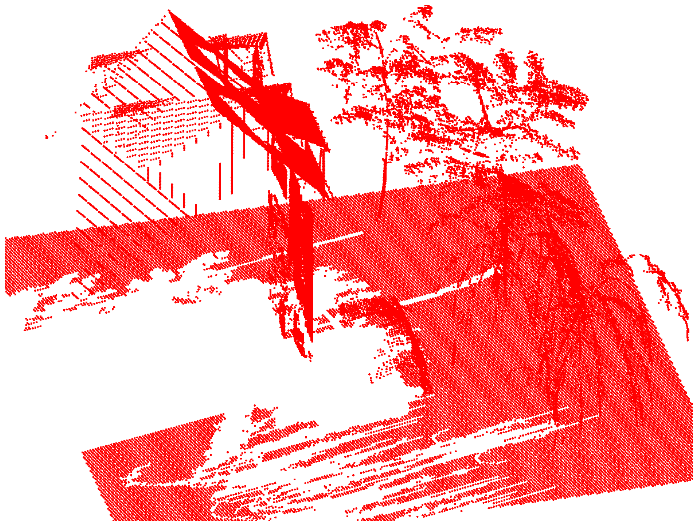

(b)

Figure 2.21: Point clouds generated from a two scan simulations of the scene in Figure 2.20, from different angles. The top and bottom scans are referred to in the text as views #1 and #2, respectively. The scanner in view 1 was located on the other side of the building from the car, causing the hood of the car to be mostly occluded by the building's shadow. In view 2, the scanner was on the other side of the trees, so the branches occlude large parts of the vehicle.

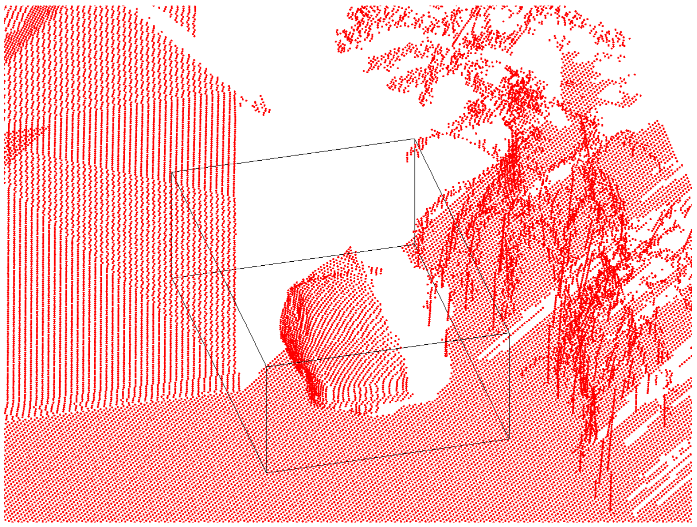
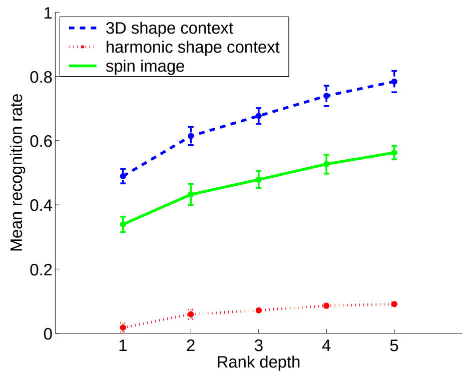
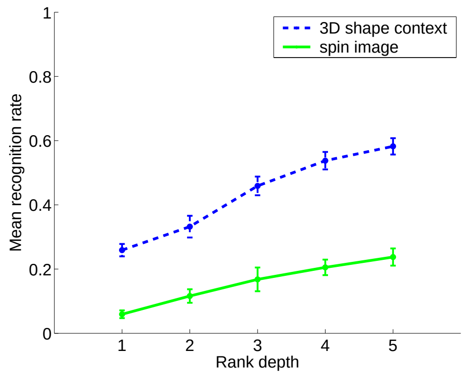


Figure 2.22: A close-up of the VOI in view #1, shown in Figure 2.21. There were about 100 basis points in the VOI in each query scene, and from those we randomly chose 80 representative descriptors for each run.



(a)



(b)

Figure 2.23: Cluttered scene results. In both, we included in the cost the 40 smallest distances out of those calculated for 80 RDs. The graphs show recognition rate versus rank depth with error bars one standard deviation from the mean. We calculated the recognition rate based on the k best choices, where k is our *rank depth* (as opposed to considering only the best choice for each query scene). We computed the mean recognition rate as described before, but counted a match to a query scene as "correct" if the correct model was within the top k matches. Graph (a) shows the results for view #1 and (b) for view #2. Using the 3D shape context we identifying on average 78% of the 56 models correctly using the top 5 choices for each scene, but only 49% of the models if we look at only the top choice for each. Spin images did not perform as well; considering the top 5 matches, spin images achieved a mean recognition rate of 56% and only 34% if only the top choice is considered. Harmonic shape contexts do particularly bad, achieving recognition slightly above chance. They chose the largest vehicles as matches to almost all the queries.

Chapter 3

Learning To Compare

I can think of no better expression to characterize these similarities than "family resemblances"; for the various resemblances between members of a family: build, features, colour of eyes, gait, temperament, etc., etc. overlap and criss-cross in the same way.—And I shall say: 'games' form a family. [...]

–Ludwig Wittgenstein, Philosophical Investigation #67

[...] The kinship is that of two pictures, one of which consists of colour patches with vague contours, and the other of patches similarly shaped and distributed, but with clear contours. The kinship is just as undeniable as the difference.

–Ludwig Wittgenstein, Philosophical Investigation # 76

3.1 Introduction

In Chapter 2, we used the Representative Descriptor (RD) method to compute distances between pairs of images. It works by sampling patches and summing patch-to-image distances, effectively treating all patches equally. In this chapter, we build upon this type of approach by introducing a way to learn which patches are more important for determining similarity and present the primary contribution of this thesis: a method that uses labeled training data to learn distance functions between images, and uses those distance functions for image retrieval and classification. In particular, we learn distance functions that are specific to individual images, and are defined in terms of similarity between their parts.

The intuition behind this work comes from three main observations:

1. the visual features of an image are not equally important in determining its similarity to some prototype;
2. which features are more salient depend on the category, or even the image, being considered; and
3. the quality of the similarity structure found within and between categories affects the performance of retrieval and classification;

Section 3.2 describes the common structure of many current algorithms for image classification, identifies distance functions as playing a key role, and gives background into recent work in designing and learning distance functions for classification. The third observation above underlies much of the work in designing and learning distance functions for classification.

Section 3.3 switches gears to talk about the nature of human categorization and the concept of "family resemblances" as an organizing principle behind human category formation. We discuss one early psychological study into category formation and structure which supports the first observation above [Rosch and Mervis, 1975].

In Section 3.4 we bring these ideas together by giving an intuition as to why it makes sense to learn distance functions in a way that makes use of family resemblances. In the remaining sections we lay out our approach to learning local distance functions from triplets of images, which is guided by the second observation above. This approach is the main contribution of this thesis.

3.2 The Role of Distance Functions

Figure 3.1 shows a flow chart that represents a crude overview of the most discriminative machine vision algorithms for classification. The learning algorithms are often based on the distances between pairs of images, such as those on the far left. From the images, sets of features are computed (e.g., raw patches, SIFT, or geometric blur). These sets of features are given to a distance (or similarity) function that returns a real number. This value, along with the values for many other pairs of images in the training set, are the input to the learning algorithm. At test time, a query image is presented to the algorithm, which compares it to some number of training examples, and using that information with the learned model, returns a classification result (a class label, e.g., "dalmatian"). Additionally, the algorithm may also be able to return retrieval results, which is typically a list of training images, ordered by their similarity to the query image.

The features computed from the images characterize image patches by fixed length vectors, which can be compared using standard metrics such as L_1 or L_2. After the feature computation step, each image is represented as a set of these feature vectors. These sets of features are input to a distance or similarity computation. One possible approach to computing an image-to-image distance is to attempt to solve the correspondence problem between the image patches by taking into account both the distances between feature vectors and the geometric arrangement of their patches (*e.g.* [Berg *et al.*, 2005]). However, this is expensive, so approaches have made use of relaxations of varying degrees. Close to the other extreme is the *bag-of-features* approach, named in reference to the bag-of-words representation common

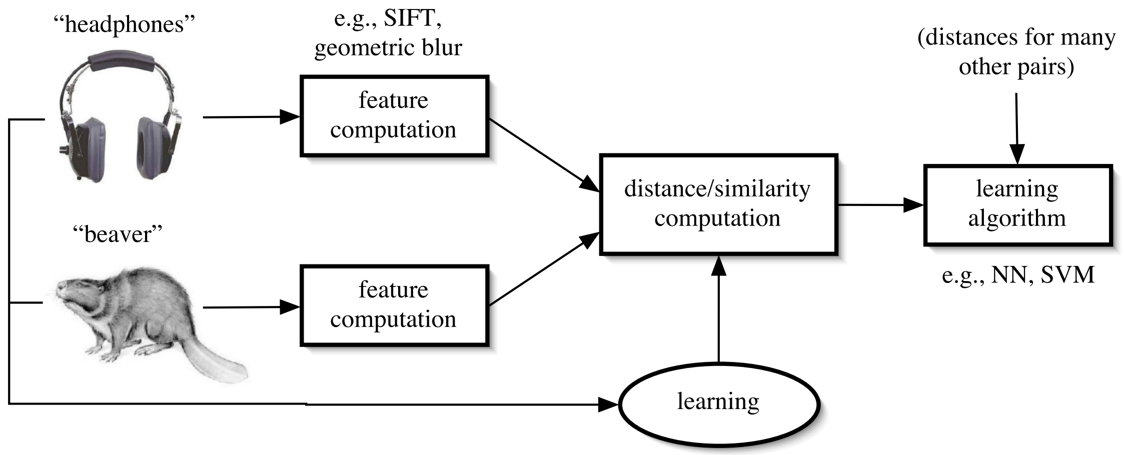


Figure 3.1: This flow chart represents a crude overview of most discriminative machine vision algorithms. The information given to the learning algorithm is usually based on pairs of images, such as the two images on the left. From each image, a set of features is computed. In the case of semi-local features, patches are first selected from the images and then feature vectors, such as SIFT or geometric blur features, are computed from the patches. This gives us two sets of patches, one for each image. From these two sets of feature vectors, a distance or similarity score is computed. That score, along with the scores from many other pairs of images, are then given to the discriminative learning algorithm. (In the case of a simple nearest neighbor algorithm, only distances between test and training images are computed and the "learning" consists only of storing the sets of features computed from the training images.) In this work, we want to involve learning earlier in the process, at the stage where the distance computation is performed (the ellipse in the diagram). We use triplets of training images to learn the distance functions, and use those functions at test time with a nearest neighbor classifier.

in Natural Language Processing. In that setting, information about the position of the patches is discarded, features are vector quantized, and an image is represented by a binary or count vector that records which clusters are represented by features present in the image. In [Mori *et al.*, 2001], an empirical comparison showed a loss of recognition performance in going from a representative descriptor to a bag-of-features representation. Many recent approaches use the feature vectors and the absolute positions of their patches, or only the feature vectors without any positional information. These approximations work well in practice, and as in Chapter 2, we work with only the distances between the feature vectors.

In the work to date in machine vision, it appears that both the choice of features and distance function are vital to the performance of the algorithm. As mentioned in Chapter 1, both SIFT and geometric blur have been successfully applied to difficult image classification and object recognition tasks, demonstrating their ability to both generalize and discriminate, so *a priori* we decide to use features of this type. This leaves open the type of distance function to use.

In the machine vision community, some recent classification work has focused on designing good similarity functions between sets of fixed-length vectors, such as patch-based features. In particular, [Grauman and Darrell, 2006b], [Lazebnik *et al.*, 2006], and [Grauman and Darrell, 2006a] introduce the Pyramid Match Kernel, Spatial Pyramid, and Vocabulary Guided (VG) Pyramid Match Kernel, respectively, which are all Mercer kernels that approximate the correspondence between two sets of features, in time linear in the number of features. They all choose a space in which to perform the matching, and divide that space hierarchically, such that the bottom-most level has very fine divisions and the uppermost level contains all the features in the sets. Starting with the bottom-most level, they compute a histogram for each of the two sets as the number of features that fall into each bucket. The amount of overlap between these histograms gives gives a similarity score for that level. As they move up the levels, they remove the features that were matched previously and continue to compute similarity values for each level. The final similarity values is a weighted combination of the similarity values for the levels of the pyramid. This is proven in [Grauman and Darrell, 2006b] to define a Mercer kernel which makes it useful for algorithms such as Support Vector Machines (SVMs).

The original Pyramid Match Kernel draws axis-parallel boundaries, similar to those used in locality-sensitive hashing, and builds the pyramid by halving the number of divisions at each level. The VG Pyramid Match Kernel addresses quantization error by choosing the divisions in a data-driven manner; they use hierarchical K-means with Euclidean distances between feature vectors to build the pyramid such that boundaries are unlikely to pass through clusters of points. Both of these approaches use the full feature vectors and ignore the position of the patches in the 2D image. The Spatial Pyramid uses the same underlying pyramid algorithm, but represents a different set of design choices. They vector quantize the features, thus discarding the detailed full feature vectors, and build the pyramid over the 2D image space. At each level, a histogram is formed for each set, for each feature cluster. Thus, they compute a distance function that uses the absolute position of features in the pyramid matching, while making use of approximations of the feature vectors.

Of these three, only the VG Pyramid Match Kernel makes use of the training data prior to training the classifier (SVM); it makes use of the feature vectors as independent samples, without regard to their co-occurrence in the images or the image class labels. It is fair to say that it does not select "visual features" in the sense that machine vision "features" describe coherent patches of an image. Instead it modifies the distances between sets of features by acting on the dimensions of the feature vectors, such as the elements of a SIFT or geometric blur feature vector. These most often correspond to edge responses in a particular part of every patch, but not coherent visual features of the image.

3.2.1 Learning Distance Functions

In this subsection we focus on methods that learn distance functions or metrics in a more data-driven manner, and there is a large body of work within the machine learning community from which we can draw.

One family of algorithms learns metrics in an unsupervised manner in order to embed data in a low-dimensional space. This family includes the classical technique multi-dimensional scaling (MDS), and more recently, Locally-Linear Embedding (LLE) [Roweis and Saul, 2000] and Isomap [Tenenbaum *et al.*, 2000]. LLE and Isomap work in a "local" manner in that they try to capture the relationships between pairs or neighborhoods of points, but in a way that globally preserves the overall structure of the data.

For each data point, LLE finds a weight vector that reconstructs that point from its K neighbors. Then, all the data points are simultaneously mapped to a lower-dimensional manifold that minimizes the reconstruction error according to those weights. The first step is a least-squares problem, while the second is an eigenvector problem.

Isomap also works by examining neighbors, but in a very different way. First, the K nearest neighbors for each point are identified. Then, it estimates the geodesic distance between every pair of points, defined as the shortest path between two points, passing only through the neighbors found in the first step. In this way, it respects any non-linear structure in the original high-dimensional manifold. Last, classical MDS is applied to the geodesic distances to give a lower-dimensional embedding, again an eigenvector problem.

These algorithms were designed to embed data for the purposes of of dimensionality reduction and data visualization. While some attempts have been made to extend them to out-of-sample tasks [Bengio *et al.*, 2004], they are still unsupervised in that they do not make use of training labels in learning their metrics.

In recent years, a body of work has focused on the supervised learning of metrics that then can be used with classification algorithms, such as nearest-neighbor. For example [Xing *et al.*, 2002; Schultz and Joachims, 2003; Shalev-Shwartz *et al.*, 2004; Weinberger *et al.*, 2005; Globerson and Roweis, 2005] all aim to learn a distance metric from labeled training data. In [Schultz and Joachims, 2003], the training data is structured in terms of more and less similar images, while the others use examples labeled as in- and out-of-class. They all assume that each labeled exemplar (image in our case) is represented by a single fixed-length feature vector, and the metric they learn between two vectors $\mathbf{x}$ and $\mathbf{x}'$ is the Mahalanobis distance $(\mathbf{x} - \mathbf{x}')^\top A(\mathbf{x} - \mathbf{x}')$, where the positive semidefinite matrix A is learned by the algorithm. This metric can be thought of as a warped Euclidean distance; each input point $\mathbf{x}$ is replaced by $A^{\frac{1}{2}}\mathbf{x}$ and the new distance is the Euclidean distance between the new points. If A is a diagonal matrix, then it simply rescales the axes of the feature space independently.

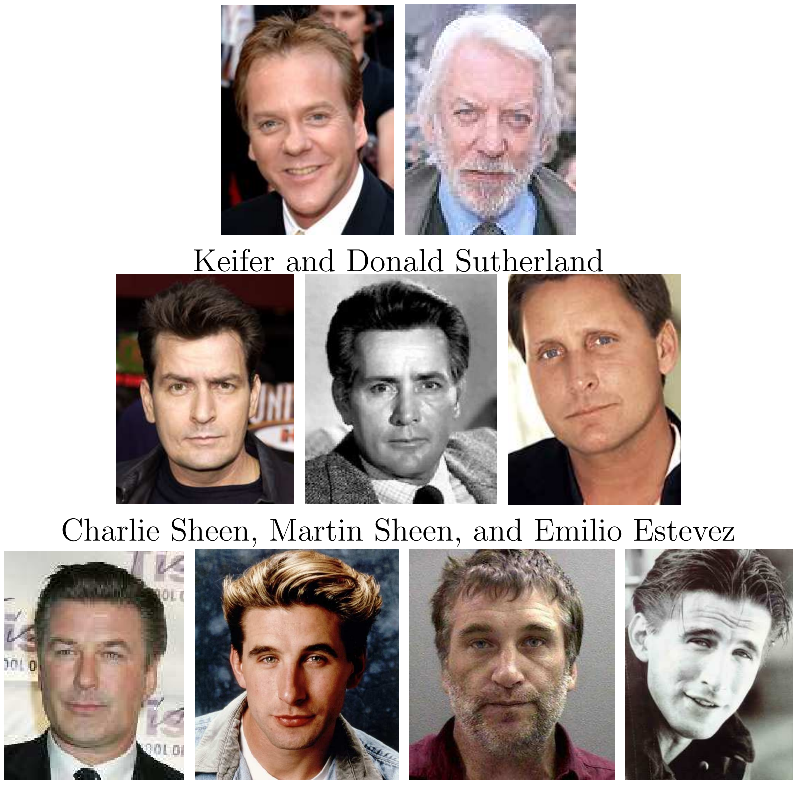
While the goal of all the above-mentioned metric learning algorithms is to learn the weight matrix, they differ in how they formulate their optimizations. For example, the goal of [Globerson and Roweis, 2005] is to collapse points in the same class while spreading apart points that are in different classes. They take a probabilistic interpretation which minimizes a KL divergence term between a probability parameterized by the weight matrix and an "ideal" probability based on the complete separation of the classes. The large-margin algorithm in [Shalev-Shwartz *et al.*, 2004] classifies examples as similar or dissimilar to examples according to some threshold, and seeks a weight matrix that minimizes errors. The approach of [Schultz and Joachims, 2003] resembles a soft-margin SVM in that the relative parameterized distances between points are constraints and the objective is a trade-off between a loss term summing slack variables and a term that regularizes the weight matrix. Our learning formulation, detailed in Chapter 4 is most similar to this approach.

These algorithms do not directly apply to the setting where each image is represented by a set of feature vectors, but they can be adapted by using a bag-of-features representation mentioned earlier in the chapter. First a sample of the feature vectors from several images is clustered, and the features in the training data are replaced with the identity of the cluster to which they are assigned. Then, an image's set of features can then be represented as a binary vector indicating the presence or absence of the different clusters, or alternatively, as a count of the presence of different clusters. In combining this representation with a Mahalanobis metric learning algorithm, we would likely find that some clusters of visual features are more important than others. However, there is evidence that recognition performance can suffer when going from a set of original features to a bag-of-features representation [Mori *et al.*, 2001].

3.3 Category Structure: Family Resemblances

The classical view of categories, from the time of Aristotle, is that they have well-defined boundaries, have sufficient and necessary conditions for membership, and are an all-or-nothing phenomenon. In his 1953 work Philosophical Investigations, Wittgenstein introduced the idea of "family resemblances" to describe the way in which members of natural categories relate to one another [Wittgenstein, 2001]. In investigations 66 and 67, quoted in the front of this thesis and at the start of this chapter, he sets up the argument and introduces this terminology. He primarily uses the example of games, demonstrating that it is not possible to find one distinguishing characteristic that is true for all games.

Taking his analogy literally, consider the images in Figure 3.2. Here we have members from three prominent families of actors, and while the family members look similar to one another, there are plenty of ways in which they also look different.



Keifer and Donald Sutherland

Charlie Sheen, Martin Sheen, and Emilio Estevez

Alec, Billy, Daniel, and William Baldwin

Figure 3.2: A literal example of Wittgenstein's *family resemblances*. While the family members look similar to one another, there are plenty of ways in which they also look different. Furthermore, where there are three or more members, there are features that some share but not others. Take for example Charlie Sheen, Emelio Estevez, and their father Martin Sheen. All three could be said to share the same mouth, but Charlie and Martin share the same eyes and brow line, where as Emelio does not. Charlie's and Emelio's noses are very similar and appear different than their father's. It is still clear, however, that they are all related. This serves as a useful analogy for the internal structure of categories.

Furthermore, where there are three or more members, there are features that some share but not others. Take for example Charlie Sheen, Emelio Estevez, and their father Martin Sheen. All three could be said to share the same mouth, but Charlie and Martin share the same eyes and brow line, where as Emelio does not. Charlie's and Emelio's noses are very similar and appear different than their father's. It is still clear, however, that they are all related.

A couple decades after Wittgenstein's Investigations, there was a growing body of work in psychology that explored the nature of human categorization, both of natural and artificially-constructed categories. Eleanor Rosch was one researcher at the forefront of this research. In a 1981 retrospective paper, Mervis and Rosch list six critical issues in investigating and modeling human categorization, in order from most to least concrete, most of which were tackled by her previous research [Mervis and Rosch, 1981]. Quoting:

1. Arbitrariness of categories. Are there any *a priori* reasons for dividing objects into categories, or is this division initially arbitrary?

2. Equivalence of category members. Are all category members equally representative of the category, as has often been assumed?

3. Determinacy of category membership and representation. Are categories specified by necessary and sufficient conditions for membership? Are boundaries of categories well defined?

4. The nature of abstraction. How much abstraction is required—that is, do we need only memory for individual exemplars to account for categorization? Or, at the other extreme, are higher-order abstractions of general knowledge, beyond the individual categories, necessary?

5. Decomposability of categories into elements. Does a reasonable explanation of objects consist in their decomposition into elementary qualities?

6. The nature of attributes. What are the characteristics of these "attributes" into which categories are to be decomposed?

These questions underly modeling choices still made in the field of machine vision, in particular the questions raised in the second through fifth items.

In the rest of this section, we focus on items two and three and the conclusions drawn from the studies in [Rosch and Mervis, 1975]. There is ample evidence that some category members are more prototypical than others. This has been shown to be true for basic perceptual categories such as color [Kay and McDaniel, 1978], and for natural and artificially-constructed categories [Rosch and Mervis, 1975]. [Rosch and Mervis, 1975] was the first work to go a step further and investigate the internal structure of categories in terms of family resemblances. Their basic hypothesis was that members of a class are prototypical to the extent that they share characteristics or features with other members of the class, and that, conversely, there will be little family resemblance between prototypical items and members of other classes. The results of their experiments on human subjects using natural and artificial categories support these claims.

For the natural superordinate categories `furniture`, `vehicle`, `fruit`, `weapon`, `vegetable`, and `clothing`, there are seldom features that all members share which other category members do not possess, and they found a gradient of representativeness from the central, prototypical members of the category to the exemplars at the boundary. They note that because superordinate categories share few if any features, "such categories may consist almost entirely of items related to each other by means of family resemblances of overlapping attributes".

In the same study, they addressed the third item in the above list by showing that the boundaries between classes are not well-defined, and that exemplars at the boundary are those that share attributes with other categories. As part of their analysis, they apply multidimensional scaling (MDS) to analyze the structure of categories based on distances derived from the sharing of features. They find at the center of the embedded space those examples subjects reported as most prototypical.

[Rosch and Mervis, 1975] also included experiments using artificially-constructed categories made of strings of letters and numbers to test category structure where no prior concept knowledge is available. They trained subjects with examples from two categories and then tested their ability to categorize novel strings, and found that examples that had a greater family resemblance to other members of their category were learned and identified more rapidly and were judged to be more prototypical members.

We wish to turn the structural model of family resemblances from [Rosch and Mervis, 1975] into a processing one by using the "family resemblances" between our training examples to help us classify novel images. Since we are only given the examples and their class labels, we must learn the resemblances from the training examples. Note that, like the experiments using artificial categories, our algorithm has no prior knowledge of the categories, and like the human subjects, we expect an algorithm based upon prototypes and family resemblances to suffer when given bad examples.

3.4 Extending Family Resemblances to Local Distance Functions

The main contribution of this thesis is a method for learning distance functions that capture family resemblances, and for using those functions to perform image retrieval and classification. In this section we provide an intuition for the claim at the beginning of the chapter that

> which visual features are more salient depend on the category, or even the image, being considered.

The experiments in the rest of this thesis will make use of the Caltech 101 object recognition dataset introduced in the first chapter. While many of these categories would be considered *basic-* or *entry-level* categories, we argue that, given the strictly visual nature of the task, and the variations in appearance, pose, and clutter, these categories more often act as superordinate classes in terms of their family resemblance structure. In particular, using state-of-the-art approaches, it is unlikely that, from the pixels, we can extract features that would occur in *all* examples of a category but not in any other category. Take for example the `chair` category, from which 56 examples are shown in Figure 3.3. While we probably could not extract a visual feature common to all, there are certainly relationships between these examples that would connect many of them. Furthermore, there is a good deal of overlap between the classes in the data set. Also in Caltech 101 is the category `windsor_chair` (see Figure 3.4), and it is very unlikely that we could find features consistent throughout the `chair` category that do not occur in the `windsor_chair` category.

This effect is more exaggerated for articulated objects, for example the 39 animal categories in the dataset. Consider Figure 3.5 which shows examples from the `cougar_face` category. If we consider the objects irrespective of representation, cougar faces do not vary much in their appearance, as the top set of images demonstrates. However, even for an object with consistent shape and color, the representation or appearance in an image can change greatly. Though all three of the bottom images belong to the same category, there are few visual features that could be extracted from the pixels that they would share; the features that determine similarity

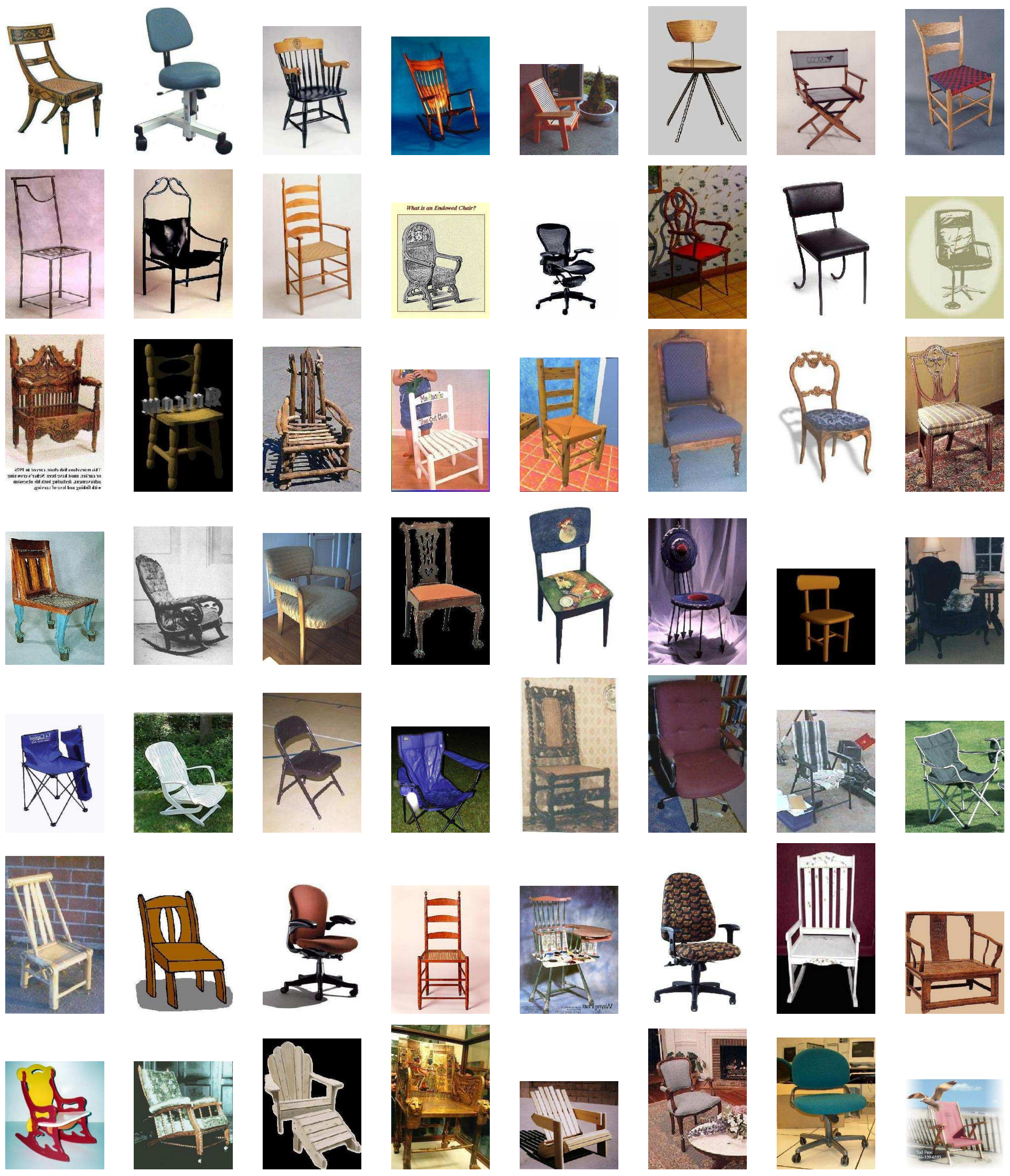


Figure 3.3: Members of the Caltech 101 `chairs` category.

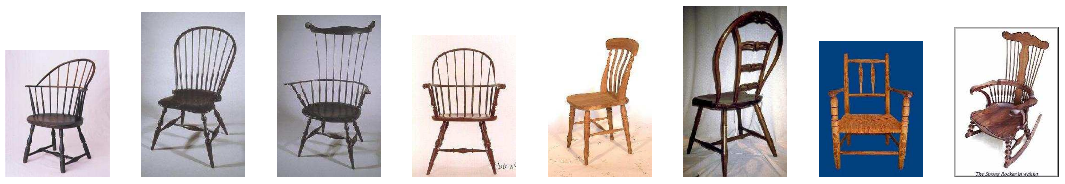


Figure 3.4: Members of the Caltech 101 `windsor_chair` category. Notice the overlap in appearance with the examples from the `chair` category in Figure 3.3. It would be difficult to find visual features that are present only in one of these categories but not the other, making the task closer to the superordinate setting described in [Rosch and Mervis, 1975].

between the left of the three and its nearest prototype are different than the features that would make the center or right images close to their nearest prototype. This clearly demonstrates that the important features can change on an image-by-image basis. If images from the bottom row are given as exemplars to a nearest-neighbor algorithm, and we wish to maximally leverage them, we would want our algorithm to pay most attention to those features that they have in common with other, more prototypical training images. But much like the human subjects in the experiments with artificial categories, our algorithm would likely be handicapped if given bad examples like the bottom three images.

Choosing which features are important for categorization on an image-by-image basis may also give us a way to make better use of cluttered examples. For example, consider the images in Figure 3.6. The first belongs to the category `seahorse`. While human facial features are very important to finding faces, we want to ignore the face for the purposes of using this as an example of the `seahorse` category. The second image is an example from the `buddha` category. It may be that in many cases strong vertical lines are very salient features, but in this case, we want to ignore the fence in front of the object of interest.

Returning to the distance functions discussed in Section 3.2, the Mahalanobis metric learning approaches give a "global" deformation in that all points in the space

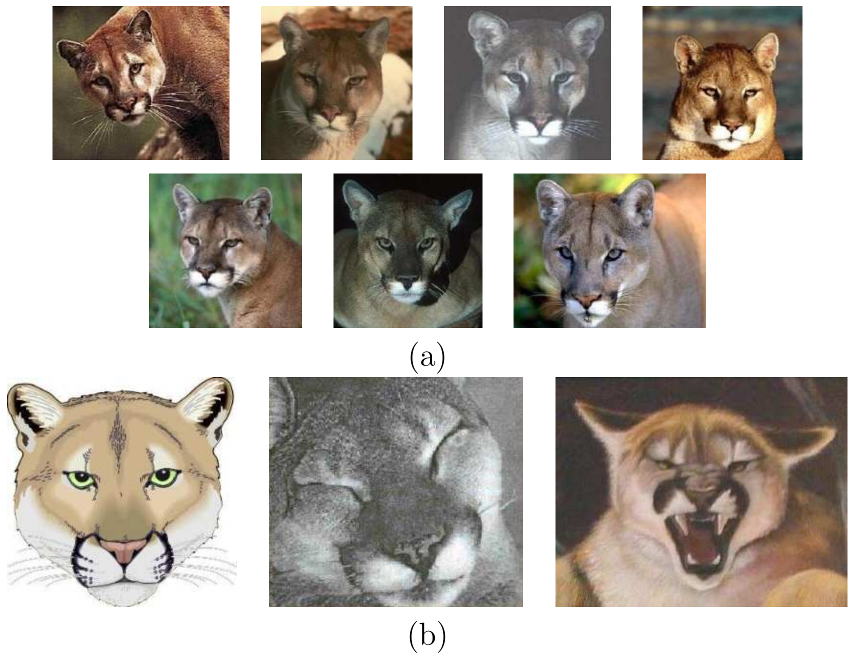


Figure 3.5: These exemplars are all drawn from the `cougar_face` category of the Caltech 101 dataset, but we can see a great deal of variation. The image on the left is a clear, color image of a cougar face. As with most `cougar_face` exemplars, the locations and appearances of the eyes and ears are a strong signal for class membership, as well as the color pattern of the face. Now consider the grayscale center image, where the appearance of the eyes has changed, the ears are no longer visible, and hue is useless. For this image, the markings around the mouth and the texture of the fur become a better signal. The image on the right shows the ears, eyes, and mouth, but due to articulation, the appearance of all have changed again, perhaps representing a common visual sub-category. If we were to limit ourselves to learning one model of relative importance across these features for all images, or even for each category, it could reduce our ability to determine similarity to these exemplars.

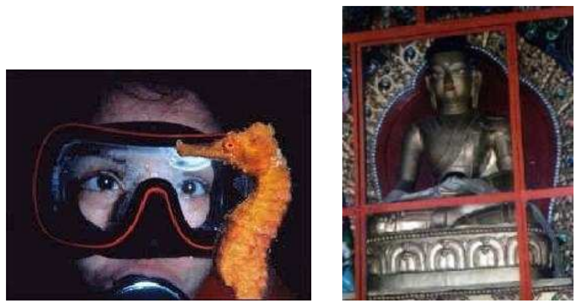


Figure 3.6: Examples of cluttered training images where the clutter would seem useful, if we didn't know better. These demonstrate that it may be useful to determine the saliency of visual features on an image-by-image basis.

are acted upon in the same way by the matrix A. For example, if we were to use a bag-of-features representation, the relative importance of the of features would be the same regardless of an image's category or the combination of features present in an image. In LLE and Isomap, however, the transformations are "local" in that different points may be acted upon differently, depending upon their neighbors. We wish to incorporate the local nature of these embedding techniques with the supervised, out-of-sample setting of the metric learning algorithms. Furthermore, we would like to do this in a way that makes use of sets of features without turning to a bag-of-feature representation. We show in the rest of this thesis that one way to achieve this is to learn a distance function for each image.[1] This allows us to choose the features that are most salient in determining an image's similarity to its neighbors. In essence, we try to learn the "family resemblances" between members of a category. We do this by learning from triplets of images.

[1]It has been suggested that learning a distance function for every image in a supervised setting is doomed to overfit, but this work demonstrates that it does not overfit, but instead outperforms state-of-the-art algorithms on a difficult classification data set.

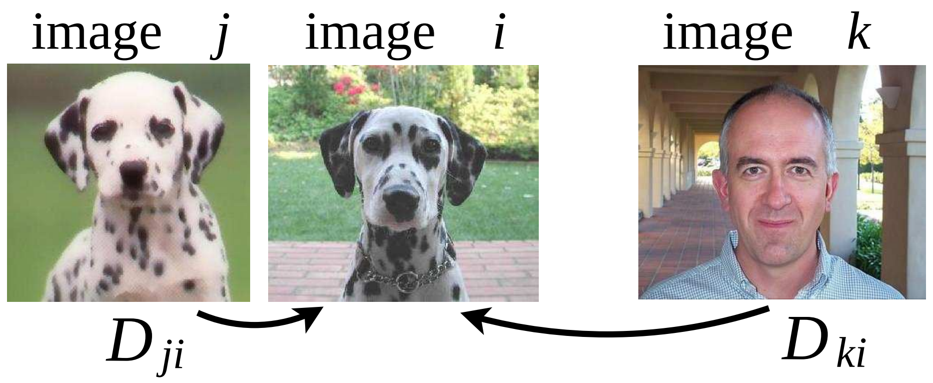


Figure 3.7: Three images from the Caltech101 data set, two from the `dalmatian` category, one from the `Faces` category. We want to learn distance functions between pairs of images such that the distance from j to i (D_{ji}) is smaller than from k to i (D_{ki}). Triplets like this one form the basis of our learning algorithm.

3.5 Learning From Image Triplets

Consider the triplet of images in Figure 3.7. Images i and j are from the Caltech 101 `dalmatian` category, while image k is from the `faces` category. Given triplets of this sort, it is clear that image i should be considered more similar to image j than it is to image k. Our goal is to find distances between images such that relationships of this type hold, for example, that the distance $D_{ji} < D_{ki}$. Say that we had such distances. Then, if we take training image i, we would be able to rank order all the training images in order of similarity to image i, and we would have a "perfect" retrieval result, in that all the in-class images would be ranked above all the out-of-class images.

Of course, if all our images are from the training set, they all come with labels, so we don't actually need the distance functions at all; we could simply rank images using their labels, a trivial task. But now consider, in place of image i, we have a novel query image and either we do not know its label or using its label would be cheating. To perform the same sort of retrieval we now need the image-to-image distances. We can learn the distance functions using the available training images and their labels.

In an ideal situation, we would have distance functions that generalize perfectly, meaning that we would still be able to rank those training images that match the query's unknown label above those that do not. This would be nice, but it is highly unlikely that we could find such functions, especially given the difficulty of the visual categorization task. However, we can learn distance functions that do perform well, as measured by the error rate on standard classification benchmarks such as Caltech101.

Given the success of patch-based features, it makes sense to learn distance functions that are based on distances between patch-based features. To approach this problem, we will learn asymmetric image-to-image distance functions, such as D_{ji} and D_{ki}, as a weighted linear combination of feature-to-image distances, which are based on distances between patch-based shape features such as SIFT [Lowe, 1999] or geometric blur [Berg and Malik, 2001].

3.5.1 Feature-to-Image Distances

Consider the images in the top row of Figure 3.8, where we want to compute the asymmetric distance D_{ij}. This distance will come from a combination of feature-to-image distances, each of which is a distance between a single feature vector from image i and the set of features in image j, similar to the minimum distances summed to give representative descriptor scores in Chapter 2. For now we will treat these features in a generic way, and any kind of patch-based feature would work in this setting. In the second row of Figure 3.8, we have focused on one such feature in image i, call it the mth of the M patches in image i. Denote the feature vector for this patch as $\mathbf{f}_{i,m}$. In image j we show in white all the patches for which we computed a feature. As in the representative descriptor method in the last chapter, we use a feature-to-feature distance metric such as L_2 norm, L_1 norm, or inverse correlation[2]

[2]Note that this does not need to be a metric, but the commonly-used feature-to-feature distances typically are.

to find the best match to $\mathbf{f}_{i,m}$ from among the set of feature vectors $\mathcal{F}_j$. Denote the vector in image j which is the best match $\mathbf{f}_j^*$. Let $d_{ij,m}$ be the distance between $\mathbf{f}_{i,m}$ and $\mathbf{f}_j^*$ using our feature-to-feature distance metric. Note that we have M of these distances, one for each of the M patch features we extracted from image i. If we let $\mathcal{F}_j$ be the set of features in image j, we can express the feature-to-image distance from the mth patch in image i to image j formally as

$$d_{ij,m} = \min_{\mathbf{f}_j \in \mathcal{F}_j} \|\mathbf{f}_{i,m} - \mathbf{f}_j\|^2 \tag{3.1}$$

where $\mathbf{f}_j^* = \min_{\mathbf{f}_j \in \mathcal{F}_j}$. Note that this feature-to-image distance is not a metric as it does not necessarily obey symmetry or the triangle inequality. This may actually be a benefit as a perceptual model; there is evidence from psychology that similarity functions are not symmetric due to the internal structure of categories [Rosch, 1978].

Given the choice of patch feature $\mathbf{f}_{i,m}$, we can compute the distance between it and any other image in this way. These feature-to-image distances give us a natural primitive upon which we can build our image-to-image distances, and from previous work using patch-based features for recognition and classification, we know that these distances between patches do generalize to unseen images. Furthermore, since the primitive we are using is based on individual patch features, it is possible to put a greater weight on the patch features that are more salient for a particular image.

3.5.2 Parameterized Image-to-Image Distances

We could simply sum these feature-to-image distances to get an image-to-image distance, or weight them each by $\frac{1}{M}$, and this would give us the basic "representative descriptor" score used in Chapter 2, which is also the distance function used to create the kernel matrices in [Zhang *et al.*, 2006]. However, we want to learn which *parts* of an image are more important in determining its similarity to other images, so we

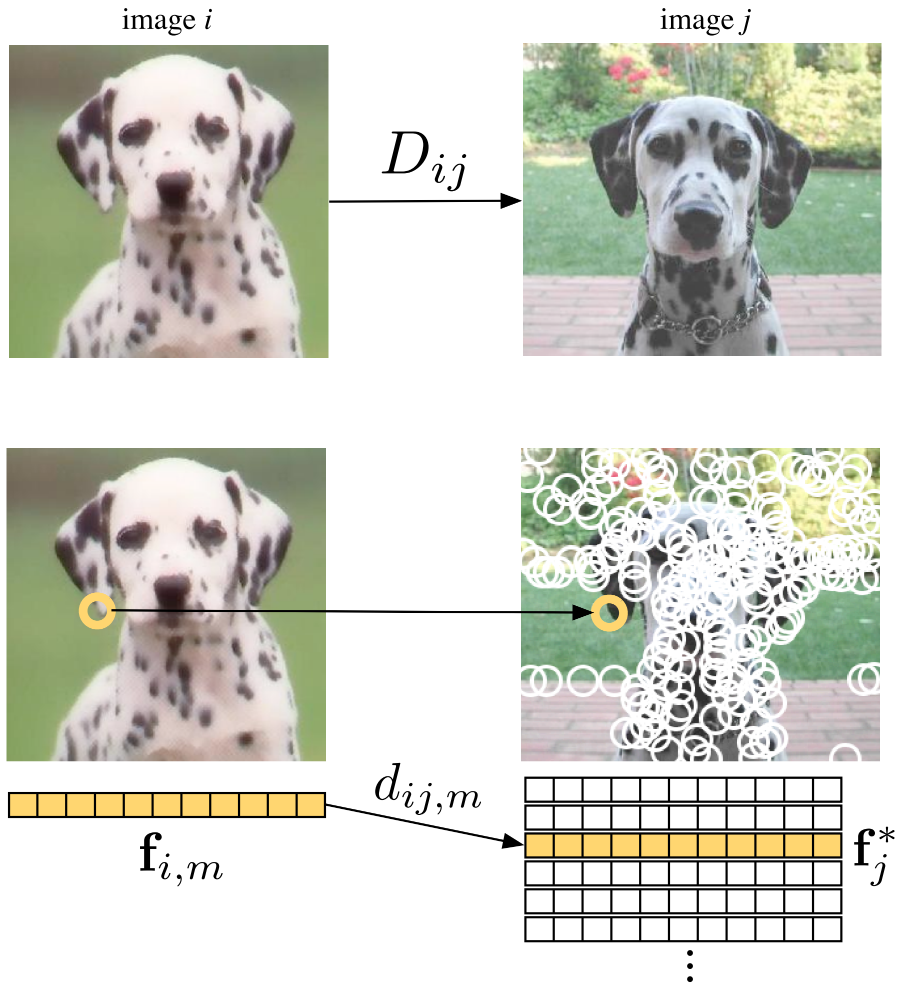


Figure 3.8: The computation of a feature-to-image distance. We wish to find the image-to-image distance from image i to image j, which will be composed of feature-to-image distances, each computed between a single feature of image i and all features image j. The best match between feature vector $\mathbf{f}_{i,m}$ and the features of image j is found using some distance metric (e.g. L_2), call it $\mathbf{f}_j^*$. Then the mth feature-to-image distance $d_{ij,m}$ is the L_2 distance between $\mathbf{f}_{i,m}$ and $\mathbf{f}_j^*$.

will assign weights to the patches, with the intuition that weights should be high for "relevant" features and low or zero for "irrelevant" or confounding features. Given a set of weights, the image-to-image distance is simply a linearly weighted combination of the M feature-to-image distances:

$$D_{ij} = \sum_{m=1}^{M} w_{i,m} d_{ij,m} \tag{3.2}$$

or, if we concatenate the M feature-to-image distances into a vector $\mathbf{d}_{ij}$, and concatenate the M weights into a vector $\mathbf{w}_i$, we can rewrite this as the dot product

$$D_{ij} = \langle \mathbf{w}_i \cdot \mathbf{d}_{ij} \rangle \tag{3.3}$$

We assume that in most settings, we are not given the vector of weights for an image, so the goal of the algorithm is to learn these weights for every training image from triplets of images, such as the triplet in Figure 3.7. To give a preview of the output of the algorithm, Figure 3.9 provides a visualization of the weights that our algorithm learned for three training images.

The weights we learn should give us image-to-image distance functions that ultimately provide good orderings over previously-unseen images. Because the image-to-image distances are asymmetric, we have two choices as to how we can use triplets of images to learn these weights. Which version we use affects the output of the algorithm and how the image-to-image distance functions can be used.

3.5.3 Focal vs. Global Learning

The weights are learned from the relationships among triplets of images, which are provided by the labeled training data. The image-to-image distances are asymmetric, so for a given triplet, we have two choices as to how we want to compare images to

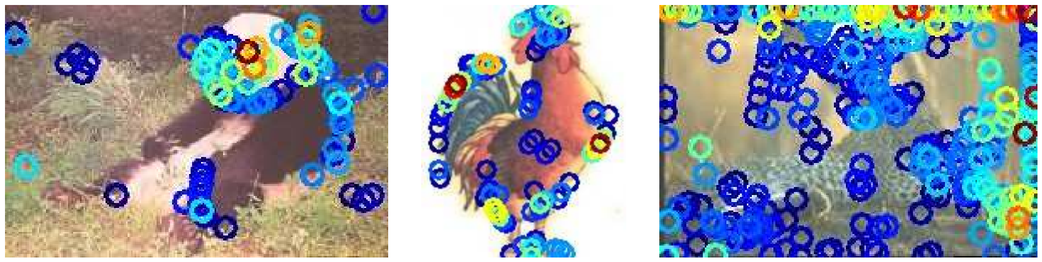


Figure 3.9: Visualizations of the weights learned by our algorithm for one type of shape feature (geometric blur with 42-pixel radius) for three images from our training set. Each circle is centered at the center point of the feature's patch and the color of the circle indicates the relative value of the weight. The colors are on Matlab's jet scale, where dark red is the highest weight (most salient feature) and dark blue is the lowest non-zero weight. Weights that were assigned a zero weight are not shown. For (a), the algorithm learned zero weights for all but 83 of the 400 small geometric blur features computed for the image, and learned that the most important feature is the patch around the eye of the panda. For (b) it learned that the most useful features are on the breast and tail of the rooster, and assigned zero weights to all but 79 of the roughly 400 small geometric blur features. For (c) it learned that the best features aren't on the leopard at all, but are along the right edge and in the upper-right corner of the image. The `Leopards` images were drawn from the Corel image set and they have a thin black border around the image that algorithms can exploit, making it a surprisingly easy category.

one another, shown in Figure 3.10.

In the first, the arrows showing the direction of the distance functions point away from the center image. In this version, the center image plays the role of a *focal* image, with respect to which the other images are ordered. For each triplet, we ideally want to maintain the relationship

$$D_{ij} < D_{ik} \tag{3.4}$$

which can be expanded as was shown in the last section to

$$\langle \mathbf{w}_i \cdot \mathbf{d}_{ij} \rangle < \langle \mathbf{w}_i \cdot \mathbf{d}_{ik} \rangle \tag{3.5}$$

Note that in this version, every triplet involves only weights for the center image, which allows us to break the problem up into a separate learning problem for every training image.

In the second, the arrows point *toward* the center image, so we are learning weights for images j and k. In this version, the center image acts as a *reference* image, in that images j and k use it to calibrate their weights to one another, and the relationship is

$$D_{ji} < D_{ki} \tag{3.6}$$

which can expanded to

$$\langle \mathbf{w}_j \cdot \mathbf{d}_{ji} \rangle < \langle \mathbf{w}_k \cdot \mathbf{d}_{ki} \rangle \tag{3.7}$$

In this version, every triplet involves the weights for two images. For one such reference image from the training set, we could make triplets of every possible pairing of positive and negative training images, and if we train with all these triplets, then the weights for all images are tied together by the triplets. In this way, this second version gives us a globally-consistent set of weights, and we refer to it as *globally learned.*

Chapter 4 explores the focally-learned version more fully, and Chapter 5 covers

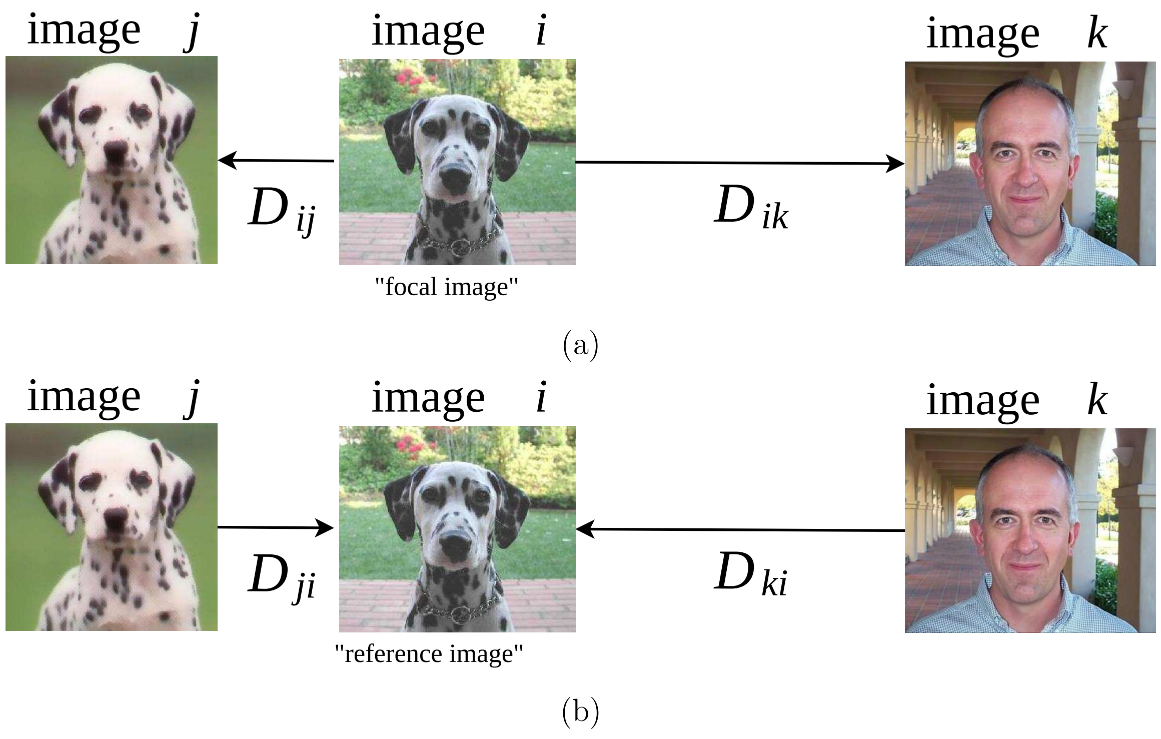


Figure 3.10: This is an illustration of the difference between the “focal” and “global” learning for local distance functions. The top row shows the configuration for focal learning, where the center image plays the role of the “focal image”, and the distance functions learned are with respect to the focal image. The bottom row shows global learning, where the center image is now a “reference image” used to calibrate the distance functions learned for the other two images.

the globally-learned version. The algorithms for the two versions are very similar; the triplets are turned into soft margin constraints and a regularization term is added to aid in generalization, turning the learning problem into a convex optimization problem, similar in final form to that in [Schultz and Joachims, 2003].

3.6 Combining Feature Types

This formulation naturally extends to allow combinations of different patch-based feature types as well as different feature-to-image distance functions. Let image i be the image for which we are learning weights. If we have several sets of features, we can compute the feature-to-image distances separately for each feature, and simply concatenate them, in the same way that we described concatenating the distances into a vector in Section 3.5.2. The only caveat being that, as is common in the application of many machine learning algorithms, it is a good idea to scale the data to be within comparable ranges so as to avoid numerical issues and make use of heuristics used to speed up running time. For example, [Platt, 1998] suggests scaling data to have unit variance. Our setting is different in that we are working with distances instead of points in space, but we can still compute the variance of *distances* in our training set, and normalize the distances we compute at test time to maintain unit variance.

In the experiments in the subsequent chapters, we will make use of geometric blur [Berg *et al.*, 2005] features at two different scales and a crude patch color feature. The computation of the geometric blur feature is shown and described in Figure 3.11. The larger of the geometric blur features has a patch radius of 70 pixels, and the smaller a patch radius of 42 pixels. Both use four oriented channels and 51 sample points, for a total of 204 dimensions. For the experiments in this thesis, we only compare “large” geometric blur features to “large” features, and “small” to “small”, so while we make use of features at different granularities, the purpose is not to make us invariant or

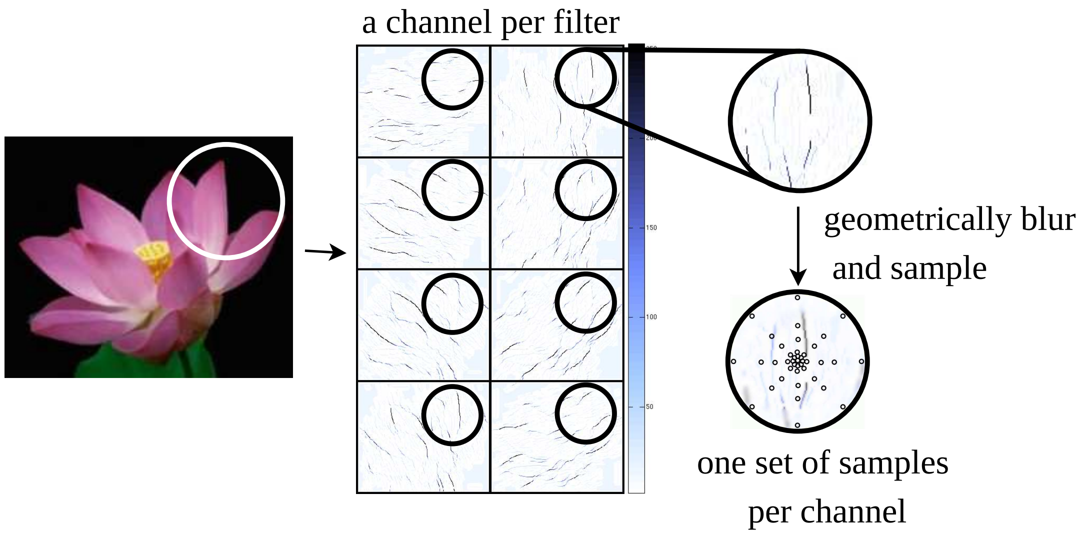


Figure 3.11: Computation of geometric blur features. A set of sparse signals is derived from the image, using filters, oriented energy, or a more sophisticated boundary detector such as Pb[Martin *et al.*, 2004]. If we have a set of eight signals, then for a given patch in the original image, we have eight signal patches. Each signal patch is blurred geometrically, meaning that the standard deviation of the Gaussian filter is increased with increasing distance from the center of the patch. The blurred patch is sampled at a set of points, and the final feature vector is the concatenation of the samples for all eight signals. In our experiments, we use a geometric blur feature computed from four orientations of Pb.

more robust in comparisons across changes of scale. We also did not tune the set of parameters used to compute the geometric blur features—we used a set of features that was previously computed for use in [Zhang *et al.*, 2006]. While many months of Alex Berg's experience have gone into the parameters used to compute geometric blur features, those parameters were in no way chosen to complement this approach.

Our color features are histograms of eight-pixel radius patches also centered at edge pixels in the image. Any "pixels" in a patch off the edge of the image are counted in a "undefined" bin, and we convert the HSV coordinates of the remaining points to a Cartesian space where the z direction is value and (x, y) is the Cartesian projection of the hue/saturation dimensions. We divide the (x, y) space into an 11×11 grid, and make three divisions in the z direction. These were the only parameters that we tested with the color features, choosing not to tune the features to the Caltech 101 dataset.

The geometric blur features are "sphered" so that the feature vectors have unit L_2 norm, and we use the minimum L_2 distance as our feature-to-image distance. In practice, the distribution of distances between geometric blur features from the training set gives a variance close to one. The color features were normalized to have a unit L_1 norm (so the bin contents represent a percentage of pixels), but still the distances between color features and between geometric blur features have very different distributions. For example, the distribution of geometric blur distances for a given image will often fit a Gaussian very well, whereas the distributions of distances between the color features tend to be multimodal and spikey. To effectively combine those features, we must post-process the feature-to-image distances. We simply adjust the color feature distances to have the same variance as the geometric blur features.

We are able to achieve very good performance without using information about the location of patches in the images, whereas many of the best results make use of the absolute position of the patches [Lazebnik *et al.*, 2006; Zhang *et al.*, 2006;

Mutch and Lowe, 2006]. [Zhang *et al.*, 2006] incorporates absolute position of the patches by combining the distance between the feature vectors with the distance between the image patch centers with a single trade-off parameter λ, and uses the same parameter for all images and all classes. Clearly incorporating position in this way would run counter to the spirit of this work. It would be more consistent to incorporate feature positions into the feature-to-image distances. For example, for feature $\mathbf{f}_{i,m}$ in image i, instead of finding the best match to *any* feature in image j, we could find the best match only among those features located in an area of image j which corresponds to the location of the mth patch in image i. This is related to the approaches taken in [Mutch and Lowe, 2006] and [Lazebnik *et al.*, 2006], and is one possible avenue for extending this work.

3.7 Previous Work In Multiway Classification

There is a relationship between the approach discussed above and other approaches to multiway classification, and in this section we discuss a few select approaches and one previous application to digit recognition that is closest to our approach.

One common approach to multiway classification is to learn several pairwise decision boundaries and somehow combine their outputs at test time to assign a single class to a test exemplar. This is popular in part due to the prevalence of simple, proven two-way discriminative classification algorithms, including SVM, logistic classification, and latent discriminative analysis (LDA). One simple strategy for combining the classifiers is to train the complete set of $\frac{N(N-1)}{2}$ pairwise classifiers, simply assign the label of the class that is chosen most often by the pairwise classifiers for a test image [Friedman, 1996]. In [Dietterich and Bakiri, 1995], pairwise classifiers are trained for every possible pair of categories, and a test image is classified using error-correcting output codes (ECOC) based on the outputs of all the pairwise classifiers. [Hastie and Tibshirani, 1997] presents an approach for turning a set of pairwise conditional class probabilities for each category into a single posterior probability for each class. That problem is underconstrained, so they pose it as an optimization where they minimize the Kullback-Liebler (KL) divergence between the conditional and desired posterior probabilities. [Platt *et al.*, 2000] introduces an algorithm called DDAG that arranges the $\frac{N(N-1)}{2}$ pairwise classifiers into a directed acyclic graph with a single root node. At the test time, one path of the graph is followed, dependent upon the output of the classifiers until it reaches the bottom where a final class decision is made. They call the use of this algorithm with SVMs the DAG-SVM. These pairwise approaches are somewhere between learning at the exemplar level and at the global level in that for different class comparisons, they learn different set of parameters. The downside is that the learning machinery is separated from the decision process by an ad hoc approach or additional layer of learning. Our "focal" learning approach detailed in the next chapter, where we learn a separate classifier for every training image, is prone to the same pitfalls. The multiclass approach of [Crammer and Singer, 2001] is a more recent approach for training a single multi-way SVM which integrates the decision process fully with the training algorithm.

[Zhang and Malik, 2003] presents an approach similar to ours for hand-written digit recognition in that they also use sets of patch-based features in a way that fits well with the psychological ideas described in this chapter. For each category, they choose a set of K exemplars, by performing a simple K-medoid clustering. These can be thought of as the most "canonical" examples in the training set. For each exemplar, they compute a set of shape context descriptors, and for every pairing with another training image, they compute a distance vector of a fixed length. If M shape contexts are computed, the distance vector to one training image is of length $2M+1$. All but one of these are computed in a manner similar to that in Equation 3.1: the feature-to-image minimum χ^2 distance from M exemplar shape contexts to the other set of shape context gives M of the distances, and M are the feature-to-image minimum L_2 distances using the raw intensity patches from which the shape contexts were extracted. The last distance is the bending energy required to match the two shapes introduced in [Belongie *et al.*, 2002]. These distance vectors are used to train logistic classifiers for every exemplar pairing. If there are N classes and K exemplars per class, $K^2N(N-1)$ two-way logistic classifiers are trained, where the target value is 1 for in-class examples and -1 for out-of-class. This results in $K(N-1)$ sets of weights for every exemplar. At test time, the shape contexts are computed for the query digit, distance vectors are computed from every exemplar to the query, and a value in $[-1,1]$ is computed for each exemplar using the weights learned by the logistic. A class choice is made using ECOC as in [Dietterich and Bakiri, 1995].

The approach in [Zhang and Malik, 2003] bears many similarities to our approach, as it was outlined earlier in this chapter. In particular, they turn sets of features into fixed-length feature vectors by taking the distances between shapes, and they learn weights for the elements of these vectors, and at test time, linearly combine the distances. However, our approach differs in many of the details. We structure our learning algorithm differently so that we learn a number of weights that is linear in the number of exemplars. Because of this we do not need to choose canonical exemplars in order to reduce the complexity, a choice which is better-suited to the more complicated structure of object categorization in natural images. Lastly, we learn weights using a large-margin approach and make class decisions using nearest neighbor. In the next two chapters, we detail how we apply these design choices in the two settings described in Section 3.5.3.

Chapter 4

Focal Learning of Local Distance Functions

4.1 Introduction

In the previous chapter, we discussed the role of distance functions and introduced two different formulations for learning distance functions from triplets, described as "focal" and "global" learning (illustrated in Figure 3.10). In this chapter, we focus on focal learning, where each learning problem involves learning a set of weights for only one image, which we call the "focal image", and the learned distance function can be used to rank images only with respect to the focal images. Sections 4.2 and 4.3 show how, starting from this triplet arrangement, we can formulate a convex optimization problem for each training image in order to learn the distance functions. Sections 4.4 and 4.5 discuss the mechanics of formulating and solving the optimization. Section 4.6 discusses leveraging the distance functions for retrieval and classification.

4.2 Focal Images

In the top row of Figure 4.1, we repeat the triplet configuration for "focal" learning introduced in the last chapter. The distance functions are asymmetric, and the arrows point away from the central image, indicating that the distance function is computed *from* image i *to* images j and k. Repeating the equations from the last chapter, each triplet gives us a constraint of the form

$$D_{ij} < D_{ik} \tag{4.1}$$

and by expressing each image-to-image distance as a weighted sum of patch-to-image distances, we can expand this to

$$\begin{aligned} \langle \mathbf{w}_i \cdot \mathbf{d}_{ik} \rangle &> \langle \mathbf{w}_i \cdot \mathbf{d}_{ij} \rangle \\ \langle \mathbf{w}_i \cdot (\mathbf{d}_{ik} - \mathbf{d}_{ij}) \rangle &> 0. \end{aligned}$$

The key thing of note here is that this triplet constraint only involves one set of weights, the weights for the "focal" image i. In the rest of Figure 4.1, we number the four images shown, and show them in different roles in the triplets. A constraint involves the weight vector for an image if and only if that image is in the central position in the triplet. This means, for example, that the triplets that affect the weight vector for image 1 are completely disjoint from those that act on the weight vector for image 2, as is shown by the grouping brackets on the right side of the figure. Since the goal is to learn the weight vectors, we can formulate two entirely independent learning problems for images 1 and 2, and if we have 1,000 training images, we formulate 1,000 completely independent learning problems. Once we have learned the weights for the focal image, we can use the parameterized distance function to order *any set of images* relative to the focal image.

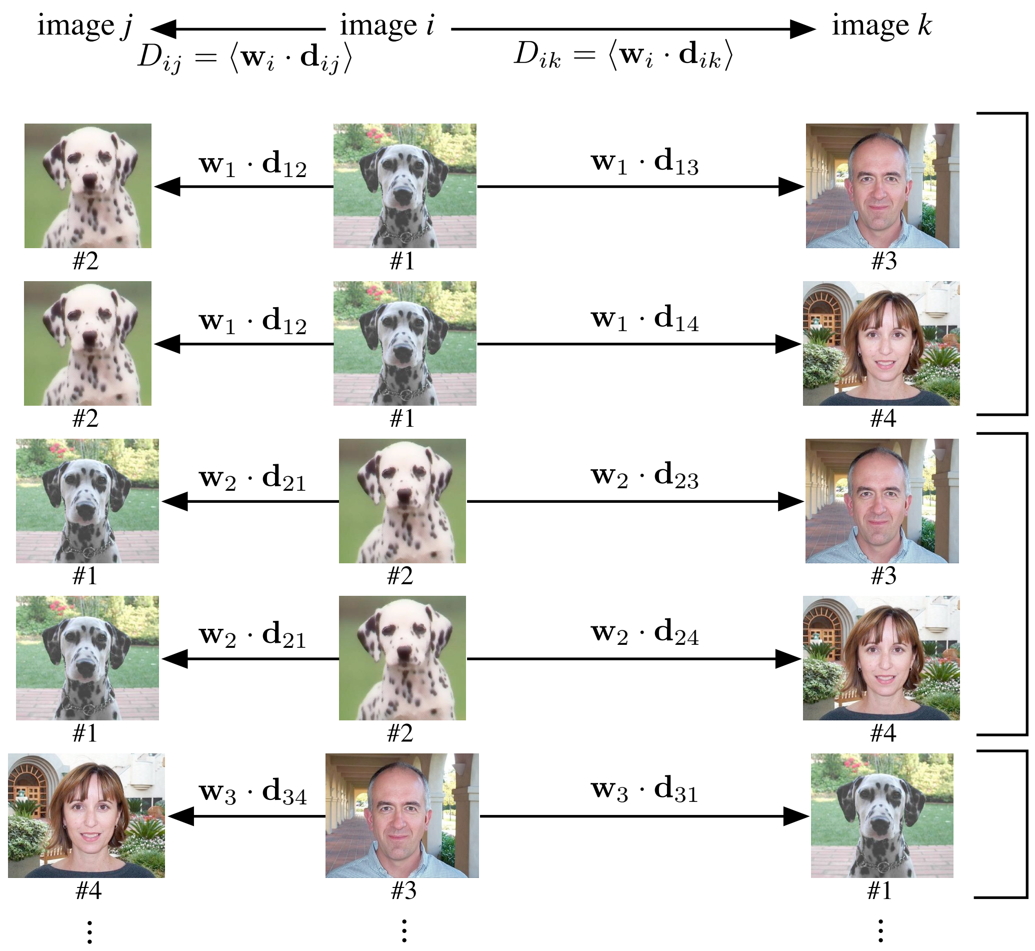


image j
image i
image k
$D_{ij} = \langle \mathbf{w}_i \cdot \mathbf{d}_{ij} \rangle$
$D_{ik} = \langle \mathbf{w}_i \cdot \mathbf{d}_{ik} \rangle$
$\mathbf{w}_1 \cdot \mathbf{d}_{12}$
$\mathbf{w}_1 \cdot \mathbf{d}_{13}$
#2
#1
#3
$\mathbf{w}_1 \cdot \mathbf{d}_{12}$
$\mathbf{w}_1 \cdot \mathbf{d}_{14}$
#2
#1
#4
$\mathbf{w}_2 \cdot \mathbf{d}_{21}$
$\mathbf{w}_2 \cdot \mathbf{d}_{23}$
#1
#2
#3
$\mathbf{w}_2 \cdot \mathbf{d}_{21}$
$\mathbf{w}_2 \cdot \mathbf{d}_{24}$
#1
#2
#4
$\mathbf{w}_3 \cdot \mathbf{d}_{34}$
$\mathbf{w}_3 \cdot \mathbf{d}_{31}$
#4
#3
#1

Figure 4.1: A subset of possible triplets for four training images. Each triplet constraint involves only the focal image's weights, thus the constraints can be grouped into independent sets by focal image.

4.3 Large-Margin Formulation

In this section, we will address the learning problem for one focal image, and formulate the optimization that we use to learn the weights for that focal image. We begin with an optimization using the hard triplet constraints, but then relax those constraints, resulting in a form similar to the soft-margin support vector machine [Platt, 1998]. Throughout the rest of the chapter, image i will be our focal image.

We begin with the basic elements of such an optimization. We could start by using the triplet constraints from Equation 4.2 as hard constraints in our optimization (we will relax this later). Second, we want to enforce non-negativity on the weights we are learning. This is necessary primarily because the quantities we are linearly combining are distances between patches and images, and these distances are always non-negative. We want our final image-to-image distances to have the same property of non-negativity, so our weights must also be non-negative. We choose our patch-to-image distances such that they should be zero where an image is compared with itself, which means that the image-to-image distance is also zero in that case. If we allow negative weights, and thus negative image-to-image distances, we enter a strange case where an image can be closer to a focal image than the focal image is with itself. Lastly, we cannot think of a case in our setting where negative weights would be useful that is not already covered by allowing zero weights. If a patch in the focal image is irrelevant or confounding, it is not clear why allowing a negative weight is better than simply giving the patch a zero weight. In fact, it seems logical not to penalize an image for having the same confounding or irrelevant information as the focal image (which is what would happen with a negative weight).

The triplet constraints and non-negativity become the constraints of our optimization problem, and in our objective function we minimize the norm of the weight vector, which allows us to find an unique solution and helps to prevent overfitting:

$$\begin{aligned} \arg\min_{\mathbf{w}_i} \quad & \tfrac{1}{2}\left\|\mathbf{w}_i\right\|^2 \\ \text{s.t.}: \quad & \forall j,k: \ \langle \mathbf{w}_i \cdot (\mathbf{d}_{ik} - \mathbf{d}_{ij}) \rangle > 0 \\ & \forall m: w_{i,m} \geq 0 \end{aligned} \tag{4.2}$$

Note the constraints are only for all j and k because i is held fixed as the focal image.

This optimization has the nice property of convexity, meaning that there is one unique answer, the global minimum. We have a choice of the norm we wish to minimize, and throughout this work, we use the L_2 norm. We chose the L_2 norm because it is more robust to noise, which is certainly a problem in image classification or recognition settings. However, we ideally would like our weight vector to be sparse, and the L_2 norm does not normally provide a sparse solution, though empirically we do see some sparsity as a result of the constraint on $\mathbf{w}$. If we were to use the L_1 norm, we may be able to get a more sparse solution, depending upon the specifics of the solver. It would be an interesting avenue of exploration to empirically test the trade-offs between the L_1 and L_2 norms.

In an idealistic, noise-free setting, it may be possible to find weight vectors such that the constraints above can be satisfied. However, machine vision problems are often far from the idealistic, noise-free setting, and if any of the triplet constraints cannot be satisfied, then the above optimization problem does not have a solution. We relax the problem by constructing an empirical loss function that allows for noise in our data, and minimizing that in our objective function. First, we turn our constraints into *large-margin* constraints,

$$\langle \mathbf{w}_i \cdot (\mathbf{d}_{ik} - \mathbf{d}_{ij}) \rangle \geq 1.$$

and then penalize linearly for each violation. Let $[z]_+$ denote the function $\max\{0, z\}$. The hinge loss for one triplet constraint is thus

$$[1 - \langle \mathbf{w}_i \cdot (\mathbf{d}_{ik} - \mathbf{d}_{ij}) \rangle]_+.$$

We wish to find a weight vector which minimizes the cumulative hinge loss across all triplets.

As in the hard-constraint version, we include the norm of $\mathbf{w}_i$ in the objective function, though here it is traded off against the sum of the hinge losses:

$$\begin{aligned} \arg\min_{\mathbf{w}_i} \quad & \tfrac{1}{2} \|\mathbf{w}_i\|^2 + C \textstyle\sum_{jk} [1 - \langle \mathbf{w}_i \cdot (\mathbf{d}_{ik} - \mathbf{d}_{ij}) \rangle]_+ \\ \text{s.t.}: \quad & \forall m: \; w_{i,m} \geq 0 \end{aligned} \tag{4.3}$$

The scalar C in the objective is a trade-off parameter between the regularization term and the empirical loss which must be set prior to training. The larger C is, the greater the emphasis on obtaining a small empirical error. Here we use one value of C for all triplets, but it would also be possible to use a different C for each triplet. When doing local learning, we choose C using cross-validation.

This optimization can be rearranged to an equivalent, more convenient form by introducing a slack variable ξ_{ijk} for each triplet:

$$\begin{aligned} \arg\min_{\mathbf{w}_i, \boldsymbol{\xi}_i} \quad & \tfrac{1}{2} \|\mathbf{w}_i\|^2 + C \textstyle\sum_{jk} \xi_{ijk} \\ \text{s.t.}: \quad & \forall j,k: \; \langle \mathbf{w}_i \cdot (\mathbf{d}_{ik} - \mathbf{d}_{ij}) \rangle \geq 1 - \xi_{ijk} \\ & \forall m: \; w_{i,m} \geq 0 \\ & \forall j,k: \; \xi_{ijk} \geq 0 \end{aligned} \tag{4.4}$$

This form is similar to the soft-margin support vector machine [Platt, 1998], but has some key differences. First, we have a non-negativity constraint on $\mathbf{w}_i$, which is not a part of the standard SVM formulation. Second, because our data is organized as triplets which implicitly capture similarity or class labels, we do not have the explicit class labels y_i that are part of the standard two-class SVM formulation. Last, we do not have a bias term, also because of the relative nature of our input data. This last difference allows us to greatly simplify the solver, as we will show in Section 4.4.

4.3.1 Relationship to Schultz and Joachims, 2003

The form of our optimization is close to that introduced in [Schultz and Joachims, 2003]. The goal of their work is to learn weights for a global metric, as was discussed in Section 3.2.1. Their optimization acts on fixed-length feature vectors $\mathbf{x}_i$ of length M, and learns an $M \times M$ weight matrix W:

$$\begin{aligned}
\underset{W,\boldsymbol{\xi}_i}{\arg\min} \quad & \tfrac{1}{2}\|AWA^T\|_F^2 + C\textstyle\sum_{ijk}\xi_{ijk} \\
\text{s.t.}: \quad & \forall i,j,k: \ (\mathbf{x}_i - \mathbf{x}_k)^T AWA^T(\mathbf{x}_i - \mathbf{x}_k) - (\mathbf{x}_i - \mathbf{x}_j)^T AWA^T(\mathbf{x}_i - \mathbf{x}_j) \geq 1 - \xi_{ijk} \\
& \forall m: \ W_{mm} \geq 0 \\
& \forall i,j,k: \ \xi_{ijk} \geq 0
\end{aligned} \tag{4.5}$$

where A is a user-defined matrix that encodes prior knowledge about the relative importance of the feature dimensions. Note that the diagonal of W is required to be non-negative, and if we set $A = I$, the identity matrix, then we can replace the matrix W with the weight vector $\mathbf{w}$ from its diagonal. Regardless of the value of A, they show that the constraints are equivalent to the those of the form

$$\langle \mathbf{w} \cdot (\boldsymbol{\Delta}_{ik} - \boldsymbol{\Delta}_{ij}) \rangle \geq 1 - \xi_{ijk} \tag{4.6}$$

if they let $\boldsymbol{\Delta}_{ij} = (A^T\mathbf{x}_i - A^T\mathbf{x}_k). * (A^T\mathbf{x}_i - A^T\mathbf{x}_k)$, where we use Matlab notation $.*$ to denote element-wise multiplication. Replacing the $\boldsymbol{\Delta}_{ij}$ variables in the constraints with our feature-to-image distances $\mathbf{d}_{ij}$, and setting $A = I$, we get the form of our optimization in Equation 4.4.

While the form is very close, the assumptions and setting are different. Their goal is to learn a metric function, so their distances must also be metrics, and they arrive at their formulation by assuming the distance functions $\mathbf{\Delta}_{ij}$ are warped L_2 distances between fixed-length feature vectors. In our setting, we do not require our final distances to be metrics, so we do not need our feature-to-image distances to be metrics (see Section 3.5.1).

4.4 Solving the Optimization

We wrote a custom solver for the optimization in Equation 4.4. Standard support vector machine packages do not handle the non-negativity constraint on the weight vectors, but just as important, our formulation allows us to use a faster, more straightforward solver. We also could have used a standard convex optimization package, but given the special structure of the constraints, a custom solver can be faster and more memory-efficient. We chose to solve the optimization using a dual method, and were able to find $\mathbf{w}_i$ for one focal image with 2,000 triplet constraints in about one to two seconds. In the rest of this section we give the derivation for the dual, the updates for the solver that we use, and implementation details.

4.4.1 Derivation of the Dual

First, let us simplify our notation. The subscript i refers to the focal image, and is the same throughout the optimization, so let us remove i from the optimization.

Then let $\mathbf{x}_{jk} = \mathbf{d}_{ik} - \mathbf{d}_{ij}$, such that the optimization from Equation 4.4 becomes

$$\begin{aligned} \underset{\mathbf{w},\boldsymbol{\xi}}{\arg\min} \quad & \tfrac{1}{2}\left\|\mathbf{w}\right\|^2 + C\textstyle\sum_{jk}\xi_{jk} \\ \text{s.t.}: \quad & \forall j,k: \ \langle \mathbf{w}\cdot\mathbf{x}_{jk}\rangle \geq 1-\xi_{jk} \\ & \forall m: \ w_m \geq 0 \\ & \forall j,k: \ \xi_{jk} \geq 0 \end{aligned} \tag{4.7}$$

We will use α_{jk} to denote the Lagrange multiplier for the triplet constraint involving images j and k, $\boldsymbol{\mu}$ to denote the set of Lagrange multipliers that enforce non-negativity on the elements of $\mathbf{w}$, and λ_{jk} to denote the multiplier for the non-negativity constraint on the slack variables ξ_{jk}. The Lagrangian therefore is:

$$\begin{aligned} \mathcal{L}(\mathbf{w},\boldsymbol{\xi},\boldsymbol{\lambda},\boldsymbol{\alpha},\boldsymbol{\mu}) = \quad & \tfrac{1}{2}||\mathbf{w}||^2 + C\textstyle\sum_{jk}\xi_{jk} - \sum_{jk}\alpha_{jk}\left(\langle\mathbf{w}\cdot\mathbf{x}_{jk}\rangle - 1 + \xi_{jk}\right) \\ & - \textstyle\sum_{jk}\lambda_{jk}\xi_{jk} - \langle\boldsymbol{\mu}\cdot\mathbf{w}\rangle \end{aligned} \tag{4.8}$$

Let us gather our dual variables to get a more convenient form

$$\mathcal{L}(\mathbf{w},\boldsymbol{\xi},\boldsymbol{\lambda},\boldsymbol{\alpha},\boldsymbol{\mu}) = \frac{1}{2}||\mathbf{w}||^2 - \sum_{jk}\alpha_{jk}\left(\langle\mathbf{w}\cdot\mathbf{x}_{jk}\rangle - 1\right) - \langle\boldsymbol{\mu}\cdot\mathbf{w}\rangle + \sum_{jk}\xi_{jk}(C-\alpha_{jk}-\lambda_{jk}) \tag{4.9}$$

Now we need to find the minimum of the Lagrangian with respect to our primal variables, $\mathbf{w}$ and $\boldsymbol{\xi}$. Each ξ_{jk} value can be maximized independently, and we can see that, since the Lagrangian is linear in ξ_{jk}, it attains a finite value only when the slope with respect to ξ_{jk} is zero. Thus, either ξ_{jk} or $C-\alpha_{jk}-\lambda_{jk}$ must be zero. This removes the ξ variables from the Lagrangian and gives us the constraint

$$C-\alpha_{jk}-\lambda_{jk} \geq 0 \tag{4.10}$$

which, combined with the constraint $\lambda_{jk} \geq 0$ gives us $C-\alpha_{jk} \geq \lambda_{jk} \geq 0$. Combining this with the positivity constraint on the dual variable α, we get

$$0 \leq \alpha_{jk} \leq C. \tag{4.11}$$

This is the same box constraint as in the soft-margin SVM [Platt, 1998]. While it will not be necessary to know ξ_{jk} to get the weight vector, we do need it if we wish to compute the primal, which is useful for determining convergence. According to the primal constraints, $\xi_{jk} \geq 1 - \langle \mathbf{w} \cdot \mathbf{x}_{jk} \rangle$ and $\xi_{jk} \geq 0$. Since increasing ξ_{jk} increases the primal objective, we want ξ_{jk} to be as small as possible, while remaining positive. This gives us

$$\xi^*_{jk} = \max\{0, 1 - \langle \mathbf{w} \cdot \mathbf{x}_{jk} \rangle\} \tag{4.12}$$

Now we find the optimal value for $\mathbf{w}$ by taking the derivative of the remaining terms with respect to $\mathbf{w}$ and setting to zero:

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial \mathbf{w}} = & \quad \mathbf{w} - \sum_{jk} \alpha_{jk} \mathbf{x}_{jk} - \boldsymbol{\mu} \overset{\text{set}}{=} 0 \\ \mathbf{w} = & \quad \sum_{jk} \alpha_{jk} \mathbf{x}_{jk} + \boldsymbol{\mu} \quad . \end{aligned} \tag{4.13}$$

Substituting this equality back into the Lagrangian, we get the dual

$$\begin{aligned}\theta(\boldsymbol{\alpha}, \boldsymbol{\mu}) &= \frac{1}{2}\left\|\sum_{jk}\alpha_{jk}\mathbf{x}_{jk} + \boldsymbol{\mu}\right\|^2 - \sum_{jk}\alpha_{jk}\left[\left\langle\left(\sum_{j}\alpha_j\mathbf{x}_j + \boldsymbol{\mu}\right)\cdot\mathbf{x}_{jk}\right\rangle - 1\right] \\ &- \left\langle\boldsymbol{\mu}\cdot\left(\sum_{jk}\alpha_{jk}\mathbf{x}_{jk} + \boldsymbol{\mu})\right)\right\rangle \\ &= \frac{1}{2}\left\|\sum_{jk}\alpha_{jk}\mathbf{x}_{jk} + \boldsymbol{\mu}\right\|^2 - \left\|\sum_{jk}\alpha_{jk}\mathbf{x}_{jk}\right\|^2 - 2\left\langle\boldsymbol{\mu}\cdot\sum_{jk}\alpha_{jk}\mathbf{x}_{jk}\right\rangle + \sum_{jk}\alpha_{jk} \\ &- \|\boldsymbol{\mu}\|^2 \\ &= -\frac{1}{2}\left\|\sum_{jk}\alpha_{jk}\mathbf{x}_{jk} + \boldsymbol{\mu}\right\|^2 + \sum_{jk}\alpha_{jk}\end{aligned} \tag{4.14}$$

The dual optimization that we need to solve is

$$\begin{aligned}\underset{\boldsymbol{\alpha},\boldsymbol{\mu}}{\arg\max} \quad & -\frac{1}{2}\left\|\sum_{jk}\alpha_{jk}\mathbf{x}_{jk} + \boldsymbol{\mu}\right\|^2 + \sum_{jk}\alpha_{jk} \\ \text{s.t.}: \quad & \forall j,k:\ 0 \le \alpha_{jk} \le C \\ & \forall m:\ \mu_m \ge 0\end{aligned} \tag{4.15}$$

4.4.2 Solving the Dual

Because our primal does not include a bias term, our dual optimization does not include a term that jointly constrains the α_{jk} dual variables. The soft-margin SVM does include a term that constrains the sum of the dual variables, which is why chunking algorithms that work with sets of the dual variables, such as Sequential Minimal Optimization (SMO), are used to solve the optimization. Instead, we can look at each dual variable independently in a very fast update, allowing us to use an approach similar to the row action method described in [Censor and Zenios, 1998]. We derive these updates by taking the derivative of the dual objective with respect to our dual variables, $\boldsymbol{\mu}$ and the set of α variables.

Taking the derivative with respect to one variable, call it α_{ab},

$$\begin{aligned}
\frac{\partial\theta}{\partial\alpha_{ab}} &= \left\langle \left(-\sum_{jk} \alpha_{jk}\mathbf{x}_{jk} - \boldsymbol{\mu} \right) \cdot \mathbf{x}_{ab} \right\rangle + 1 \\
&= -\sum_{jk} \alpha_{jk} \langle \mathbf{x}_{jk} \cdot \mathbf{x}_{ab} \rangle - \langle \boldsymbol{\mu} \cdot \mathbf{x}_{ab} \rangle + 1 \\
&= -\sum_{jk \neq ab} \alpha_{jk} \langle \mathbf{x}_{jk} \cdot \mathbf{x}_{ab} \rangle - \alpha_{ab} \|\mathbf{x}_{ab}\|^2 - \langle \boldsymbol{\mu} \cdot \mathbf{x}_{ab} \rangle + 1 \overset{\text{set}}{=} 0 \\
\alpha_{ab} &\leftarrow \frac{1 - \sum_{jk \neq ab} \alpha_{jk} \langle \mathbf{x}_{jk} \cdot \mathbf{x}_{ab} \rangle - \langle \boldsymbol{\mu} \cdot \mathbf{x}_{ab} \rangle}{\|\mathbf{x}_{ab}\|^2} \quad .
\end{aligned} \tag{4.16}$$

After the update, we can simply clip the value to the feasible region, $0 \leq \alpha_{ab} \leq C$, since, due to convexity, if the optimum is outside the feasible region, the best answer within the feasible region is at the boundary. This update can be performed sequentially for each α variable.

Similarly, we can derive the update to the $\boldsymbol{\mu}$ variable:

$$\begin{aligned}
\frac{\partial\theta}{\partial\boldsymbol{\mu}} = -\sum_{jk} \alpha_{jk}\mathbf{x}_{jk} - \boldsymbol{\mu} \overset{\text{set}}{=} 0 \\
\boldsymbol{\mu} \leftarrow \max\left\{ \mathbf{0}, -\textstyle\sum_{jk} \alpha_{jk}\mathbf{x}_{jk} \right\},
\end{aligned} \tag{4.17}$$

where the max in the second line is an element-wise max. This is necessary because the values of $\boldsymbol{\mu}$ are constrained to be positive, and as with α, if outside the feasible region, it finds its optimum at the boundary. Note that the non-zero entries of $\boldsymbol{\mu}$ are simply the negative values of $\sum_{jk} \alpha_{jk}\mathbf{x}_{jk}$, multiplied by -1, which means that when added to $\sum_{jk} \alpha_{jk}\mathbf{x}_{jk}$, $\boldsymbol{\mu}$ simply zeroes out the negative entries. Remember that $\mathbf{w} = \sum_{jk} \alpha_{jk}\mathbf{x}_{jk} + \boldsymbol{\mu}$, so the update to $\boldsymbol{\mu}$ is just really just a clipping of the negative entries of $\mathbf{w}$ to zero, which makes sense since $\boldsymbol{\mu}$ was the dual variable enforcing positivity on $\mathbf{w}$.

4.4.3 KKT Conditions

The optimum of a convex nonlinear optimization is found when the Karush-Kuhn-Tucker (KKT) conditions are satisfied for every constraint, and that the dual variable α for a constraint does not to be updated if the constraint passes the KKT conditions. For our optimization problem, the KKT conditions are:

$$\begin{aligned} \alpha_{jk} = 0 &\Rightarrow \langle \mathbf{w} \cdot \mathbf{x}_{jk} \rangle \geq 1 \\ 0 < \alpha_{jk} < C &\Rightarrow \langle \mathbf{w} \cdot \mathbf{x}_{jk} \rangle = 1 \\ \alpha_{jk} = C &\Rightarrow \langle \mathbf{w} \cdot \mathbf{x}_{jk} \rangle \leq 1 \end{aligned} \tag{4.18}$$

These are very similar to the KKT conditions for the SVM, as described in [Platt, 1998]. We test the conditions within some tolerance to account for numerical issues and also to speed the solver. For example, if $0.999 < \langle \mathbf{w} \cdot \mathbf{x}_{jk} \rangle < 1.001$, then we consider the KKT conditions to be satisfied, regardless of the value of α_{jk}. This test plays a dual role: (1) we always test the KKT conditions before performing an update, skipping the update if it passes, and (2) once all constraints pass the KKT conditions within tolerance, we stop the optimization.

4.4.4 Iterations

We now have a set of updates, one for each α variable, and one for the $\boldsymbol{\mu}$ variable, and they can be applied in any order. In our implementation, we alternate between updating one α variable and updating the $\boldsymbol{\mu}$ variable (which is equivalent to recomputing $\mathbf{w}$ and clipping the values to be non-negative).

In structuring our iterations over the constraints, we use a variant of the heuristics in [Platt, 1998]. A constraint is *bound* when $\alpha_{jk} = 0$ or $\alpha_{jk} = C$. The iterations are divided at the top level into *epochs*. Each epoch begins with all constraints marked as unbound, and ends when little no progress is being made with the remaining unbound constraints. We keep a list of the "active constraints" in the current epoch, which are those that have not yet become bound since the beginning of the epoch. Once a constraint is bound, it is removed from the active set and is not examined again until the beginning of a new epoch. This heuristic is based on the observation that, once a variable becomes bound, it is less likely to change, thus time should be spent updating other variables.

When beginning a new epoch, we randomly reorder the constraints and iterate through them repeatedly in that order, performing updates when they fail the KKT conditions. We refer to each of these passes through the unbound constraints as a *sweep*.

There are many possible heuristics we could use to decide when "little or no progress" is being made in order to end an epoch. We use a combination of two tests. If we iterate through all unbound constraints, and none change, this means all unbound constraints pass the KKT conditions, and we end the epoch. Or, if the dual objective is changing less than some threshold, we end the epoch. When this second condition is met, we decrease the threshold for the next epoch, making it more difficult to meet.

We end the optimization the first time we begin a new epoch, and on the first sweep, no updates are performed (i.e. the KKT conditions are met for all constraints).

4.4.5 Bookkeeping

How we perform updates and track the state of the optimization can affect the amount of computation needed to complete the optimization. In this section, we explain the variables that we track in this implementation, and give pseudocode for updating one constraint. Let N be the number of constraints for the optimization and D be the dimensionality of the $\mathbf{x}_{jk}$ vectors.

To perform the α updates in Equation 4.16, we need the value for $\|\mathbf{x}_{jk}\|^2$ for every triplet. We compute these before we begin our epochs since they are based only on the training data and will not change. There are N of these and they are stored in the array `normxsq`.

We create an array `alphas` of length N for our α_{jk} variables, and set the entries to zero. We create a variable `sumalpha` that will maintain $\sum_{jk} \alpha_{jk}$, and initialize it to zero. We initialize an array `alphax` of length D to zeros. This array will maintain $\sum_{jk} \alpha_{jk}\mathbf{x}_{jk}$, which is used in place of $\mathbf{w}$ and $\boldsymbol{\mu}$ in updates. In the code below, when an array is not indexed using brackets, we are performing a vector computation. We will use [`alphax`]$_+$ to indicate that we are only using the positive elements of the `alphax` array. We have a matrix `x` of the data vectors, and we use `x[i]` to denote the data vector for one triplet. The temporary variables `oldalpha` and `newalpha` are scalars.

For purposes of pseudocode, we index each constraint by a single index, i, which replaces the two indexes we were using previously, e.g. jk. We will use $\cdot$ to denote the dot product between two vectors. `C` and `precision` are user-defined parameters. The update for the ith constraint:

```
1. testval ← [alphax]$_+$ · x[i]

2. IF (alpha[i] == 0 AND testval $\geq$ 1 - precision) OR
   (alpha[i] > 0 AND alpha[i] < C AND abs(testval - 1) $\leq$ precision) OR
   (alpha[i] == C AND testval $\leq$ 1 + precision),
   ERASE i, next i, GOTO step 1.  ELSE GOTO step 3.

3. oldalpha = alpha[i]

4. other_alphax[d] ← alphax - oldalpha $\times$ x[i]

5. newalpha ← (1 - (other_alphax + [−alphax]$_+$) · x) / normxsq[i]
```

```
6. IF newalpha $<$ 0, newalpha $\leftarrow$ 0
   IF newalpha $>$ C, newalpha $\leftarrow$ C
7. alphax $\leftarrow$ other_alphax + newalpha $\times$ $\mathbf{x}$
8. sumalpha $\leftarrow$ sumalpha - oldalpha + newalpha
9. alpha[i] $\leftarrow$ newalpha
```

4.5 Selecting Triplets

In creating the set of triplets from class-labeled training data, we could make the exhaustive set of triplets for each focal image by pairing every in-class image with every out-of-class image. This is excessive and can adversely affect the results.

The use of the hinge loss in the optimization gives it the property that, once a constraint is satisfied, it does not matter by how much, and the algorithm only focuses on constraints that are unsatisfied or are exactly satisfied. Those constraints are referred to as "at the margin". If we have a triplet that gives us a constraint that is very easy to satisfy, it adds little to nothing to the final result. These easy triplets can occur when the negative example is so different from the focal image that any reasonable weight vector would satisfy a triplet involving it and most other positive example. While these extra triplets do not adversely affect the outcome, they do add to the memory and computational overhead, making it advantageous to remove them.

While easy negative examples are merely extraneous, hard positive examples are potentially hazardous. A hard positive example is one that is so different from the focal image in its features that, when paired with some negative example, it is not possible to satisfy the constraints and maintain positivity of the weight vector. When using an exhaustive set of triplets in one run on an image in the `scissors` category of Caltech 101, we observed that the only solution the algorithm could find was to set all weights to zero, essentially "collapsing" the space.

Thus, we wish to select a subset of triplets that removes the easy negative examples and the hard positive examples. This is equivalent to saying that we only want to start out with training examples that are similar to the focal image in some way. For clarity, we refer to all the images available for training as the *training set* and the set of images used to train with respect to a given focal image as its *learning set*. We keep in our learning set those images that are similar to the focal image according to at least one feature-to-image distance measure. For each of the M feature-to-image patch distance measures, we find the top K closest images. If that group contains both in- and out-of-class images, then we make triplets out of the full bipartite match. If all K images are in-class, then we find the closest out-of-class image according to that distance measure and make K triplets with one out-of-class image and the K similar images. We do the converse if all K images are out of class. In our experiments, we used $K = 5$, and we have not yet performed experiments to determine the effect of the choice of K. The final set of triplets for the focal image is the union of the triplets chosen by the M measures. On average, we used 2,210 triplets per focal image, and mean training time was 1-2 seconds (not including the time to compute the features, feature-to-image distances, or choose the triplets). While we have to solve N of these learning problems, each can be run completely independently, so that for a training set of 1,515 images, we can complete this optimization on a cluster of 50 1GHz computers in about one minute.

4.6 Caltech101 Experiments

We test our approach on the Caltech101 dataset [Fei-Fei *et al.*, 2004], discussed in the first chapter. This dataset has artifacts that make a few classes easy, but many are quite difficult, and due to the important challenges it poses for scalable object recognition, for the last three years it has been one of the *de facto* standard benchmarks for two-dimensional multi-class object recognition. The dataset contains images from 101 different categories (ignoring the background class), with the number of images per category ranging from 31 to 800, with a median of about 50 images.

In all results we show in this chapter and the next, the features described in Section 3.6 are used. The images are first resized to speed feature computation. The aspect ratio is maintained, but all images are scaled down to be around 200×300. We computed features for each of these images as described in Section 3.6. We used up to 400 of each type of feature (two sizes of geometric blur and one color), for a maximum total of 1,200 features per image. For images with few edge points, we computed fewer features so that the features were not overly redundant. After computing feature-to-image distances, we rescale the distances for each focal image and feature to have a standard deviation of 0.1.

After the large-margin training, we have a set of weights, a distance function, and an image ranking for every training image. We can get a sense of what is learned by looking at some of the focal rankings. Figure 4.2 and 4.3 show hand-picked rankings for two images. Figure 4.4 shows shorter rank lists for a selection of focal images chosen uniformly at random.

4.7 Direct Applications: Searching By Prototype

The focal ranking could naturally be the basis for a browsing application over a set of images which are sparsely labeled with relative similarity information. Consider a ranking of images with respect to one focal image, as in Figure 4.2. The user may see this and decide they want more sunflower images. Not all the images shown are focal (training) images, but the eighth image in the top row is, and the user could choose

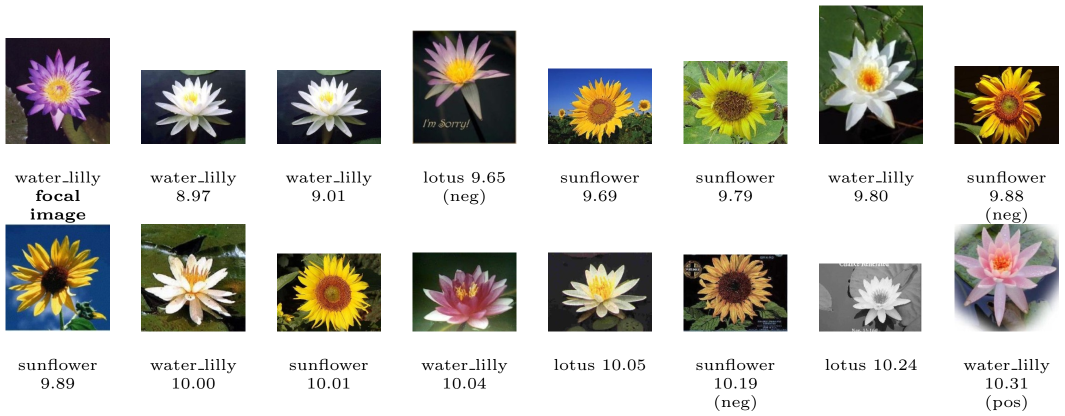


Figure 4.2: The first 15 images from a ranking induced for the focal image in the upper-left corner, trained with 15 images/category. Each image is shown with its raw distance distance, and only those marked with (pos) or (neg) were in the learning set for this focal image. Note that the `lotus` image is marked as a negative example even though it looks very similar to the focal image. One of the artifacts of the Caltech 101 data set is the redundancy in the categories.

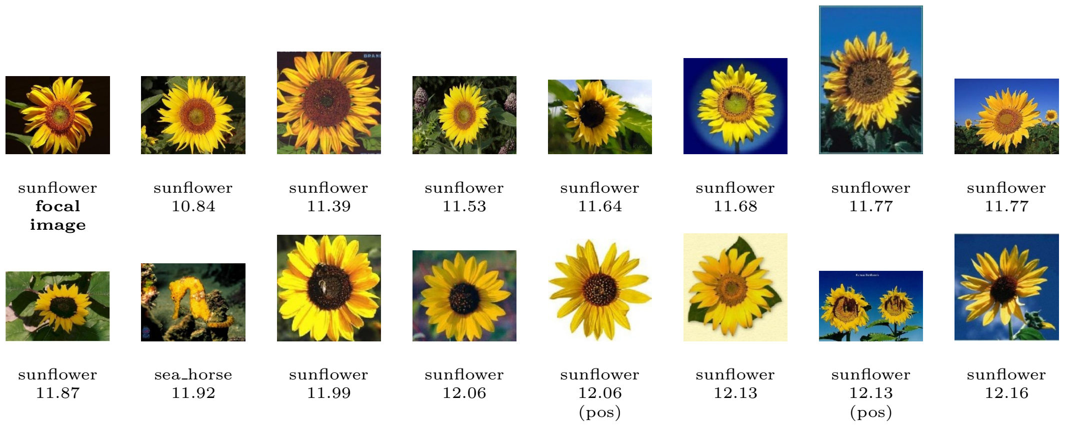


Figure 4.3: Focal ranking for the image in the upper-left corner, which was selected from the results in Figure 4.2. Each image is shown with its raw distance distance, and only those marked with (pos) or (neg) were in the learning set for this focal image.

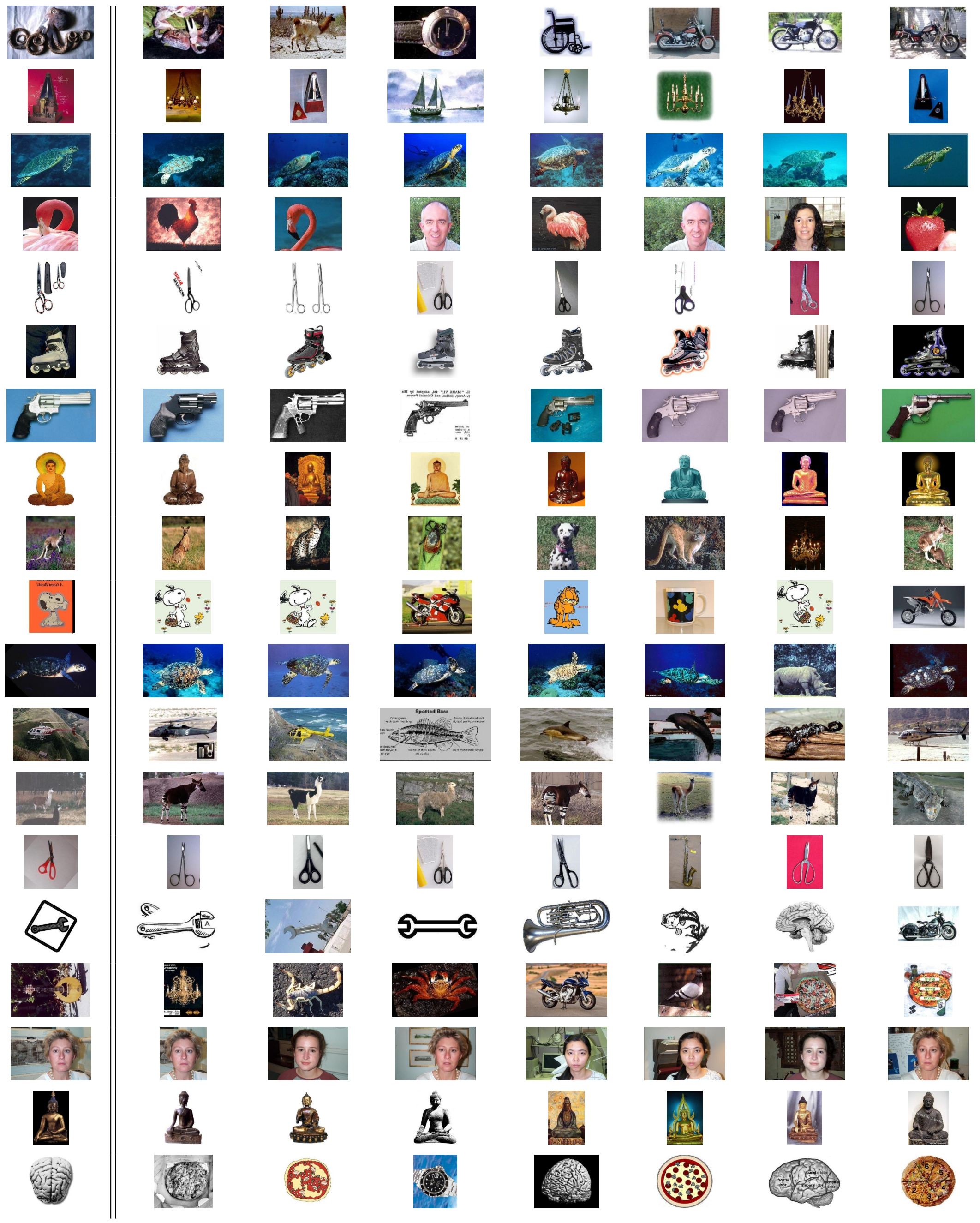


Figure 4.4: A selection of focal ranking results, chosen uniformly at random from 1,515 training images. The left column are the focal images, and the rankings from best to worst are shown across the row, from left to right. As in Figures 4.2 and 4.3, the rankings include a mix of training and testing images.

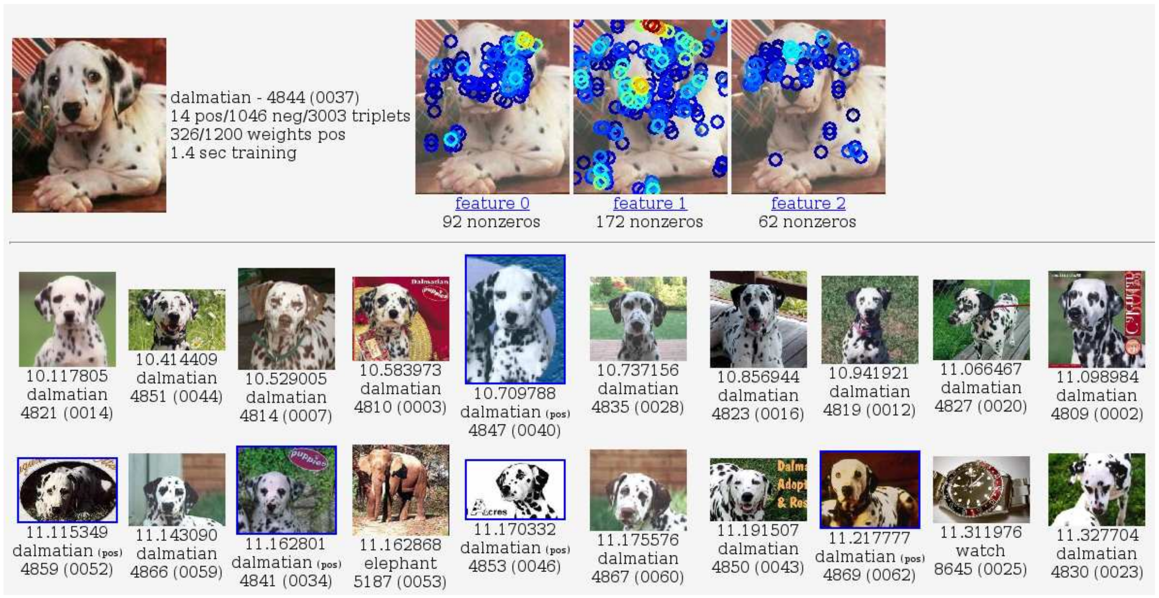
this image. The application would then show the ranking with that sunflower image as the focal image, shown in Figure 4.3. In this way, can help a user to navigate images using their visual similarity, which is based upon their family resemblances. Figure 4.5 shows two screen captures of a browsing application.

4.8 Retrieval and Classification

The distance functions only rank images with respect to a particular focal image. When given a novel image, we can place it in the rankings for all focal images, but the distances to that novel image are not directly comparable. This is because (1) the weight vectors for each of the focal vectors are not constrained to share any properties other than non-negativity, (2) the number of feature-to-image distance measures and their potential ranges are different for each focal image, and (3) some learned distance functions are simply better than others at characterizing similarity within their class. There is cause for hope, however, because the set of focal rankings is a rich source of information; every focal image ranks all other training images, and we know the position of a test image in each of these lists. In this section we show two ways to use these interrelated focal rankings to perform image retrieval and classification.

4.8.1 Retrieval and Classification I: Additional Training

One way to address this is to do a second round of training for each focal image where we fit a logistic classifier to the binary (in-class versus out-of-class) training labels and learned distances. This puts all distances on a $[0, 1]$ scale and fits the gradient to how well each focal ranking ranks training images of its own class. Now, given a query image $\mathcal{Q}$, we can compute a probability that the query is in the same class as each of the focal (training) images, and we can use these probabilities to rank the training



dalmatian - 4844 (0037)
14 pos/1046 neg/3003 triplets
326/1200 weights pos
1.4 sec training

feature 0
92 nonzeros

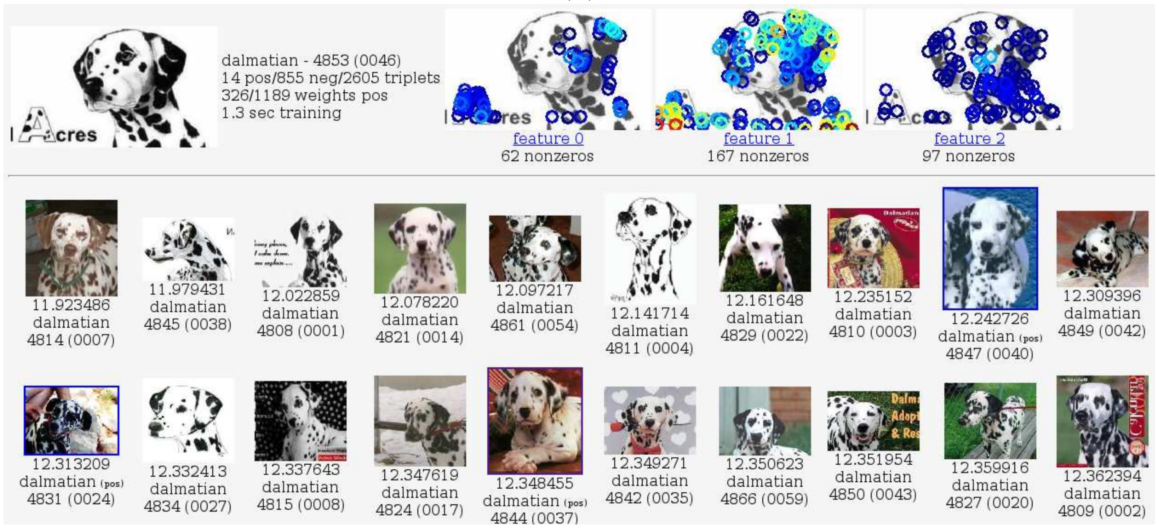
feature 1
172 nonzeros

feature 2
62 nonzeros

10.117805 dalmatian 4821 (0014)
10.414409 dalmatian 4851 (0044)
10.529005 dalmatian 4814 (0007)
10.583973 dalmatian 4810 (0003)
10.709788 dalmatian (pos) 4847 (0040)
10.737156 dalmatian 4835 (0028)
10.856944 dalmatian 4823 (0016)
10.941921 dalmatian 4819 (0012)
11.066467 dalmatian 4827 (0020)
11.098984 dalmatian 4809 (0002)

11.115349 dalmatian (pos) 4859 (0052)
11.143090 dalmatian 4866 (0059)
11.162801 dalmatian (pos) 4841 (0034)
11.162868 elephant 5187 (0053)
11.170332 dalmatian (pos) 4853 (0046)
11.175576 dalmatian 4867 (0060)
11.191507 dalmatian 4850 (0043)
11.217777 dalmatian (pos) 4869 (0062)
11.311976 watch 8645 (0025)
11.327704 dalmatian 4830 (0023)

(a)



dalmatian - 4853 (0046)
14 pos/855 neg/2605 triplets
326/1189 weights pos
1.3 sec training

feature 0
62 nonzeros

feature 1
167 nonzeros

feature 2
97 nonzeros

11.923486 dalmatian 4814 (0007)
11.979431 dalmatian 4845 (0038)
12.022859 dalmatian 4808 (0001)
12.078220 dalmatian 4821 (0014)
12.097217 dalmatian 4861 (0054)
12.141714 dalmatian 4811 (0004)
12.161648 dalmatian 4829 (0022)
12.235152 dalmatian 4810 (0003)
12.242726 dalmatian (pos) 4847 (0040)
12.309396 dalmatian 4849 (0042)

12.313209 dalmatian (pos) 4831 (0024)
12.332413 dalmatian 4834 (0027)
12.337643 dalmatian 4815 (0008)
12.347619 dalmatian 4824 (0017)
12.348455 dalmatian (pos) 4844 (0037)
12.349271 dalmatian 4842 (0035)
12.350623 dalmatian 4866 (0059)
12.351954 dalmatian 4850 (0043)
12.359916 dalmatian 4827 (0020)
12.362394 dalmatian 4809 (0002)

(b)

Figure 4.5: Two screen shots from a simple image browsing application, where the focal rankings were learned for the prototype image. The prototype image in the bottom ranking was selected from the second row in the top ranking. Note that the top ranking does not contain many illustrations, and browsing using an illustration has brought up more illustrations of dalmatians.

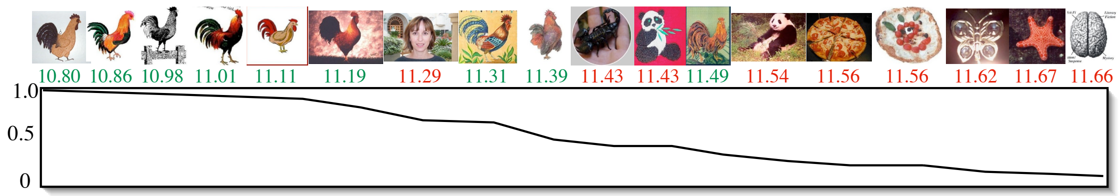


Figure 4.6: An illustration of approach I to retrieval (Section 4.8.1) where we fit a logistic to each focal ranking. This figure shows part of the ranking for one focal image, and the distances to each of the training images in the ranking. Distances are shown in red for negative training examples, and green for positive training examples. The graph is the logistic function that maps from the training distances to the $[0, 1]$ scale (the graph is not smooth because it is shown sampled at the given distances).

images relative to one another. Figure 4.6 shows a function learned for one focal image. In addition to putting the probabilities on the same scale, the logistic also helps to penalize poor focal rankings.[1]

To classify a query image, we first run the retrieval method above to get the probabilities for each training image. For each class, we sum the probabilities for all training images from that class, and the query is assigned to the class with the largest total. Formally, if p_j is the probability for the jth training image $\mathcal{I}_j$, and $\mathcal{C}$ is the set of classes, the chosen class is $\arg\max_{\mathcal{C}} \sum_{j:\mathcal{I}_j \in \mathcal{C}} p_j$. This can be shown to be a relaxation of the Hamming decoding scheme for the error-correcting output codes in [Allwein *et al.*, 2000] in which the number of focal images is the same for each class. This decoding procedure was also used in [Zhang and Malik, 2003].

4.8.2 Retrieval and Classification II: Two Heuristics

The second approach for turning focal rankings into retrieval results makes use of two simple heuristics that use the K focal distance functions and the position of a

[1]We experimented with abandoning the max-margin optimization and just training a logistic for each focal image; the results were far worse, perhaps because the logistic was fitting noise in the tails.

query image Q in those rankings to order the K training images. To understand the heuristics, it helps to think of each focal distance function as assigning a real number to other images, and also as inducing a ranking over images.

The first heuristic attempts to put each of the K spaces on the same scale by normalizing each of the distance functions by a constant. We took a very simple approach, and normalized each distance function by the distance between the focal image and the closest training image according to the focal image's distance function. If the distance to the closest training image is zero, then we take the smallest nonzero distance from the ranking.

The second heuristic attempts to penalize the focal images which do not rank the test image well relative to the rankings of other training images. The distance function for one focal image induces a ranking that includes the query image $\mathcal{Q}$. If $\mathcal{Q}$ is very similar to the focal image, then there should be few dissimilar training images ranked above $\mathcal{Q}$. To capture this quantitatively, we simply count the number of training images that are labeled as out-of-class that are ranked above $\mathcal{Q}$. The larger the value, the less similar we believe $\mathcal{Q}$ is to the focal image, relative to the other focal images. We call this the *error penalty* for the focal image. For example, Figure 4.2 shows the raw distances for each of the images to the focal image in the upper-left corner. There are three negative training examples in this ranking, the lotus in the 3rd position, the sunflower in the 7th position, and the sunflower in the 13th position. The test images in the 1st and 2nd positions would be given an error penalty of zero, and the test image of the sunflower in the 8th position (first image, second row) would be given an error penalty of two.

These two heuristics are complementary, and to use both, we generate a score from $\mathcal{Q}$ to each focal image by simply multiplying the normalized distance by the error penalty plus one (to avoid zeros). It is interesting to note that these two heuristics capture the same information as the logistic approach. The first heuristic normalizes the space to make values across focal images more comparable, which the logistic achieves by putting all scores on a $[0, 1]$ scale. The second heuristic adjusts those scores to represent the quality of the ranking, which the logistic achieves by changing the inflection point of the curve depending upon how the training examples were ranked.

Given retrieval results for a query image, we can assign a label to the query using a k-nearest-neighbor classifier. In our experiments, we use a modified 3-NN algorithm, where if two of the top three images have the same category label, that label is assigned to the query image. If not, then we look down the list until either (1) we find a training image with the same label as an image earlier in the list, or (2) we reach the tenth item without finding two that agree, in which case we assign the label of the first image in the list.

These heuristics work surprisingly well, giving results that are comparable to or better than the results using the more principled logistic training above. These experiments were run using a different training/testing regime and so are not directly comparable to the results reported in the graphs. But when both approaches were tested using that regime, the heuristic approach performed better than the logistic version when using five training examples, and only slightly worse when using fifteen examples.

4.8.3 Classification Results

In our classification experiments, we ignore the background class and work in a forced-choice scenario with the 101 object categories, i.e. a query image is assigned to exactly one of the categories. We use the same testing methodology and mean recognition reporting described in Grauman et al. [Grauman and Darrell, 2006b]: we experiment with different training set sizes (given in number of examples per class), and in each training scenario, test with all other images in the Caltech101 dataset, except the `BACKGROUND_Google` class. Recognition rate per class is computed, then averaged across classes. This normalizes the overall recognition rate so that the performance for categories with a larger number of test images does not skew the mean recognition rate.

We ran a series of classification experiments using all features and the logistic training method from Section 4.8.1, each with a different number of training images per category (either 5, 15, or 30), where we generated 10 independent random splits of the 8,677 images from the 101 categories into training and test sets. We report the average of the mean recognition rates across these splits as well as the standard deviations. We determined the C parameter of the training algorithm using leave-one-out cross-validation on a small random subset of 15 images per category, and our final results are reported using the best value of C found (0.1). In general, however, the method was robust to the choice of C, with only changes of about 1% in recognition with an order of magnitude change in C near the maximum. The graph in Figure 1.2 shows these results with most of the published results to date for the Caltech 101 dataset.

In the 15 training images per category setting, we also performed recognition experiments on each of our features separately, the combination of the two shape features, and the combination of two shape features with the color features, for a total of five different feature combinations. We performed another round of cross-validation to determine the C value for each feature combination[2]. Recognition in the color-only experiment was the poorest at 6% (0.8% standard deviation)[3] The next

[2]For big geometric blur, small geometric blur, both together, and color alone, the values were C=5, 1, 0.5, and 50, respectively.

[3]Only seven categories did better than 33% recognition using only color: `Faces_easy`, `Leopards`, `car_side`, `garfield`, `pizza`, `snoopy`, and `sunflower`. Note that all `car_side` exemplars are in black and white.

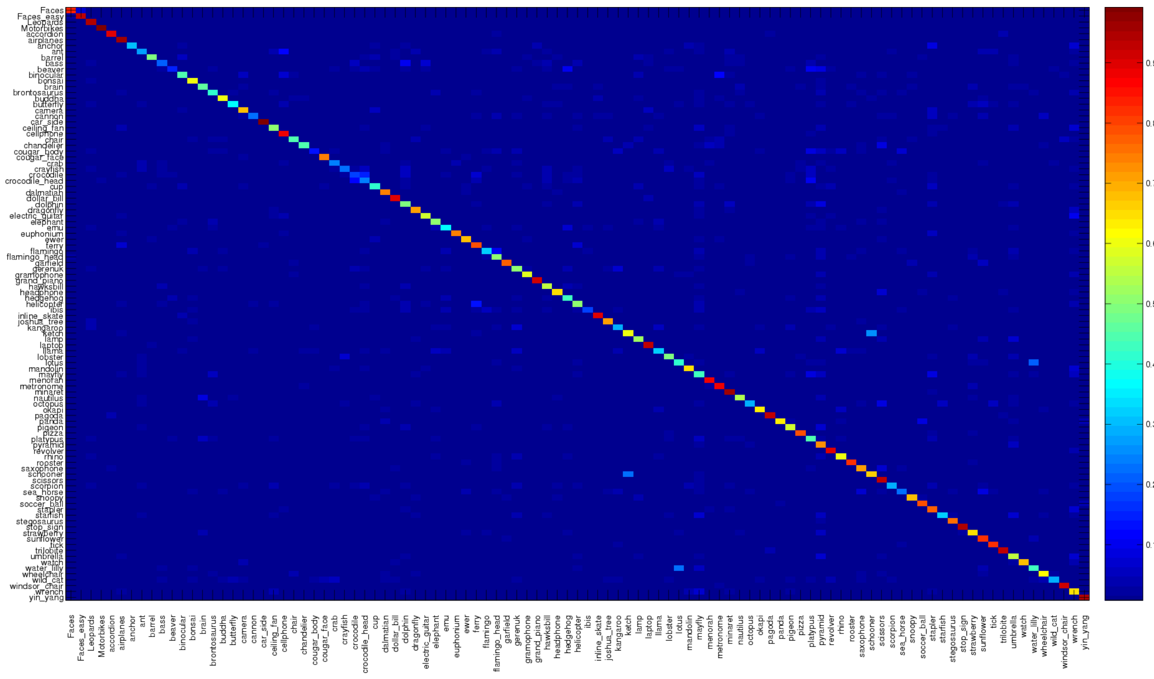


Figure 4.7: Average confusion matrix for 15 training examples per class, across 10 independent runs. Shown in color using Matlab's jet scale, shown on the right side.

best performance was from the bigger geometric blur features with 49.6% (±1.9%), followed by the smaller geometric blur features with 52.1% (±0.8%). Combining the two shape features together, we achieved 58.8% (±0.8%), and with color and shape, reached 60.3% (±0.7%), which is better than the best previously published performance for 15 training images on the Caltech 101 dataset [Zhang *et al.*, 2006]. Combining shape and color performed better than using the two shape features alone for 52 of the categories, while it degraded performance for 46 of the categories, and did not change performance in the remaining 3. In Figure 4.7 we show the confusion matrix for combined shape and color using 15 training images per category. The ten worst categories starting with the worst were `cougar_body`, `beaver`, `crocodile`, `ibis`, `bass`, `cannon`, `crayfish`, `sea_horse`, `crab`, and `crocodile_head`, nine of which are animal categories.

Almost all the processing at test time is the computation of the feature-to-image distances between the focal images and the test image. In practice the weight vectors that we learn for our focal images are fairly sparse, with a median of 69% of the elements set to zero after learning, which greatly reduces the number of feature comparisons performed at test time. We measured that our unoptimized code takes about 300 seconds per test image.[4] After comparisons are computed, we only need to compute linear combinations and compare scores across focal images, which amounts to negligible processing time. This is a benefit of our method compared to the KNN-SVM method of Zhang, et al. [Zhang *et al.*, 2006], which requires the training of a multiclass SVM for every test image, and must perform all feature comparisons.

[4]To further speed up comparisons, in place of an exact nearest neighbor computation, we could use approximate nearest neighbor algorithms such as locality-sensitive hashing or spill trees.

Chapter 5

Global Learning of Local Distance Functions

5.1 Introduction

This chapter presents in greater detail the second formulation illustrated in Figure 3.10 for learning distance functions. The formulation in the previous chapter was the first of the two, and it is most well-suited to image browsing because it independently learned a distance function for each training image. To apply that formulation to image retrieval and classification, we must employ heuristics or another round of learning. In contrast, the formulation in this chapter is designed to learn distance functions which apply directly to retrieval and nearest-neighbor classification. This is possible because it learns all the distance functions together in one large optimization, which enforces that they are globally-consistent.

5.2 Reference Images

In the top row of Figure 5.1 we repeat the second choice for the triplet configuration shown in Figure 3.10. In this formulation, the arrows representing the direction of the distance functions point toward the center image, which we now call a *reference image*. For this triplet, we learn the distance functions for images j and k, but not for image i. Image i is present to ensure that the distance functions learned for j and k are calibrated correctly to one another, given that image j should consider image i to be closer than image k should. Thus, they are "referencing" image i in order to learn their distance functions.

In order for such a formulation to work, we need many triplets where images alternately play the roles of images i, j, and k. When a given image is in the role of image j, it is adjusting its distance function so that it is closer to a positive example. In the role of image k, it is adjusting its distance function such that it is further from a negative example. And when it is in the role of image i, it is serving to calibrate two other images to one another, to ensure that the distances learned are consistent. In the rest of Figure 5.1, we show the rest of the possible triplets for four training images. By including our training images in these various roles, we can create a rich tapestry of distance constraints between our training images that interrelate them all.

When we test with a new image, that test image replaces the reference image in the triplet. Given a test image q, if $D_{jq} < D_{kq}$, then image j is more similar to q than is k, according to our learned distance functions, which gives us an ordering over training images for performing retrieval. Note that this allows a direct application of the distances we have learned, without any further manipulation. Also, say j is the *most* similar image of all our training images. In that case, it would be reasonable to guess that image q is the same class as image j (*e.g.*, a `dalmatian` if we are talking

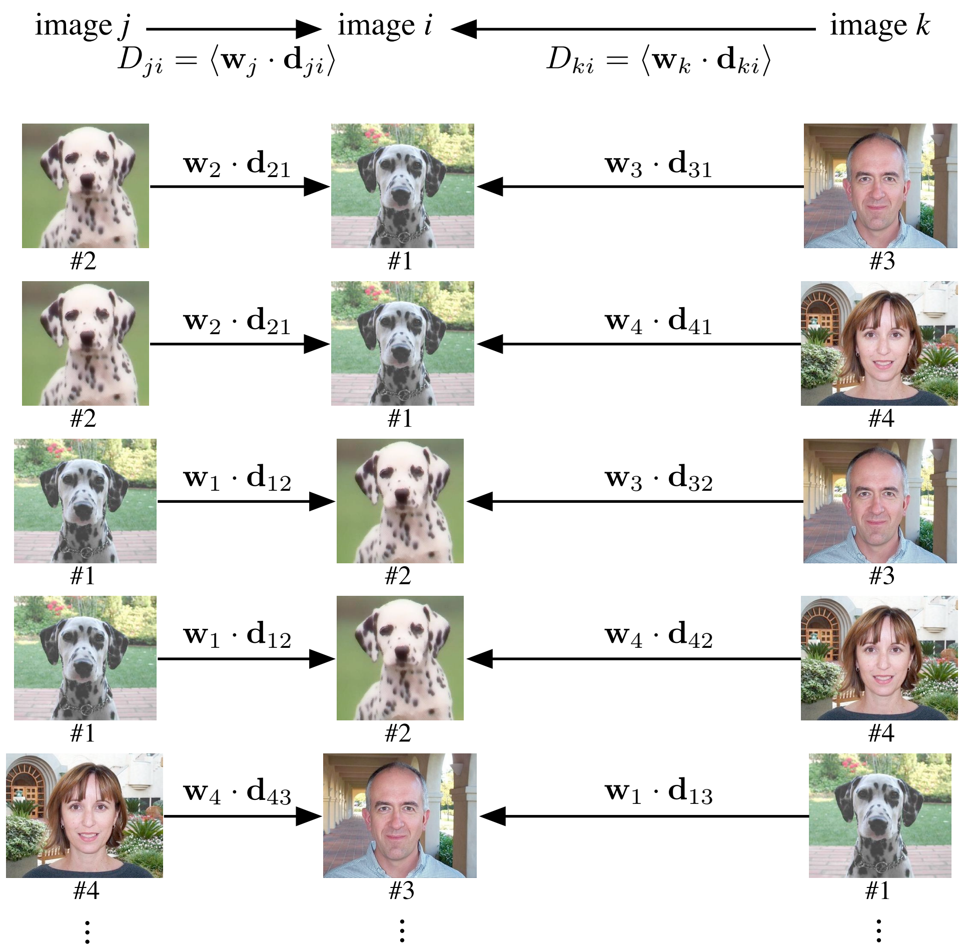


image j
image i
image k
$D_{ji} = \langle \mathbf{w}_j \cdot \mathbf{d}_{ji} \rangle$
$D_{ki} = \langle \mathbf{w}_k \cdot \mathbf{d}_{ki} \rangle$
$\mathbf{w}_2 \cdot \mathbf{d}_{21}$
$\mathbf{w}_3 \cdot \mathbf{d}_{31}$
#2 #1 #3
$\mathbf{w}_2 \cdot \mathbf{d}_{21}$
$\mathbf{w}_4 \cdot \mathbf{d}_{41}$
#2 #1 #4
$\mathbf{w}_1 \cdot \mathbf{d}_{12}$
$\mathbf{w}_3 \cdot \mathbf{d}_{32}$
#1 #2 #3
$\mathbf{w}_1 \cdot \mathbf{d}_{12}$
$\mathbf{w}_4 \cdot \mathbf{d}_{42}$
#1 #2 #4
$\mathbf{w}_4 \cdot \mathbf{d}_{43}$
$\mathbf{w}_1 \cdot \mathbf{d}_{13}$
#4 #3 #1

Figure 5.1: The top row gives the triplet relationships used in the example global triplet in Figure 3.10. Here we have numbered each image and generated five of the eight possible triplets for this set of four images.

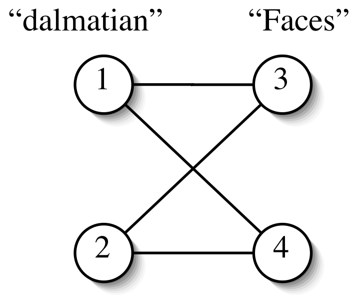


Figure 5.2: This figure shows, for the triplets in Figure 5.1, which weight vectors are linked by a triplet constraint. Even for the incomplete set shown in that figure, we have a complete bipartite graph for the four images shown, with a disjoint set for each category. If we were to use all triplets for 101 different categories, our constraints would form a 101-partite graph.

about the images in the top triplet in Figure 5.1). Thus, the distance functions can also be used directly to perform a nearest-neighbor classification.

Ideally we would like to learn a parameterized distance function for every training image such that the relationships for all triplets of training images hold:

$$D_{ji} < D_{ki} \tag{5.1}$$

As described in Chapters 3 and 4, we parameterize our distance functions to be weighted combinations of patch distances. Using the vector notation introduced in Equation 3.3, we can express this as

$$\langle \mathbf{w}_k \cdot \mathbf{d}_{ki} \rangle > \langle \mathbf{w}_j \cdot \mathbf{d}_{ji} \rangle \tag{5.2}$$

With the constraint in this form, it is perhaps easier to see a key difference between this and the formulation in the previous chapter: this constraint involves *two* sets of weights while the previous formulation involved only one. This serves to underscore that the learning problem does not break apart as in the previous formulation; the weights we are trying to learn are tied together in the constraints.

As we also saw in the last chapter, the ideal case described above is seldom possible for natural images. We again turn the triplet inequalities into constraints for a large-margin optimization.

5.3 Large-Margin Formulation

Our derivation of the large-margin formulation largely follows that in Section 4.3. We can again turn the above constraint into a soft constraint with a slack variable:

$$\langle \mathbf{w}_k \cdot \mathbf{d}_{ki} \rangle - \langle \mathbf{w}_j \cdot \mathbf{d}_{ji} \rangle \geq 1 - \xi_{ijk} \tag{5.3}$$

the primary difference being that a single constraint now involves two weight vectors. We can follow the derivation from Section 4.3 to arrive at the following optimization:

$$\begin{array}{rl} \min\limits_{\{\mathbf{w}_i\},\boldsymbol{\xi}} & \frac{1}{2}\sum_i \|\mathbf{w}_i\|^2 + C\sum_{ijk}\xi_{ijk} \\ \text{s.t.} & \forall i,j,k: \ \xi_{ijk} \geq 0 \\ & \forall i,j,k: \ \langle \mathbf{w}_k \cdot \mathbf{d}_{ki} \rangle - \langle \mathbf{w}_j \cdot \mathbf{d}_{ji} \rangle \geq 1 - \xi_{ijk} \\ & \forall i,m: \ w_{i,m} \geq 0 \end{array} \tag{5.4}$$

However, now the optimization is over *all* weight vectors simultaneously, and the constraints range over all triplets, so the constraints are for all values of i, j, and k.

This may appear to be a more complicated optimization than in Equation 4.4, but with a small amount of manipulation, it can be massaged into the same form. We denote by $\mathbf{W}$ the vector which is the concatenation of the image-specific vectors $\mathbf{w}_i$ for every image of our training set. Thus, each image-specific vector corresponds to a subrange of $\mathbf{W}$. We also need to introduce a similar expansion for the distances. Let $\mathbf{X}_{ijk}$ denote a vector of the same length as $\mathbf{W}$ such that all of its entries are 0 except the subranges corresponding to images k and j, which are set to $\mathbf{d}_{ki}$ and $-\mathbf{d}_{ji}$, respectively. It is straightforward to verify that the term $\mathbf{w}_k \cdot \mathbf{d}_{ki} - \mathbf{w}_j \cdot \mathbf{d}_{ji}$ can now be simply written as $\mathbf{W} \cdot \mathbf{X}_{ijk}$ and Eq. 5.3 distills to $\mathbf{W} \cdot \mathbf{X}_{ijk} \geq 1 - \xi_{ijk}$. We arrive at the following form, which differs from Equation 4.4 only in that it iterates over three image indexes instead of two:

$$\begin{aligned} \min_{\mathbf{W},\boldsymbol{\xi}} \quad & \tfrac{1}{2}\left\|\mathbf{W}\right\|^2 + C\sum_{ijk}\xi_{ijk} \\ \text{s.t.} \quad & \forall i,j,k: \; \xi_{ijk} \geq 0 \\ & \forall i,j,k: \; \mathbf{W}\cdot\mathbf{X}_{ijk} \geq 1 - \xi_{ijk} \\ & \forall m: \; W_m \geq 0 \end{aligned} \tag{5.5}$$

where C controls the trade-off between the loss and regularization terms and is an input to the optimization (Section 5.4.4 discusses the choice of C parameter).

5.4 Solving the Optimization

As in Chapter 4, we solve this optimization using a dual method. The dual problem of the primal in Eq. 5.5 is

$$\begin{aligned} \max_{\boldsymbol{\alpha},\boldsymbol{\mu}} \quad & -\tfrac{1}{2}\left\|\sum_{ijk}\alpha_{ijk}\mathbf{X}_{ijk} + \boldsymbol{\mu}\right\|^2 + \sum_{ijk}\alpha_{ijk} \\ \text{s.t.} \quad & \forall i,j,k: \; 0 \leq \alpha_{ijk} \leq C \\ & \forall m: \; \mu_m \geq 0 \end{aligned} \tag{5.6}$$

The size of the problem is our primary hurdle. The large number of constraints results in a longer running time for all optimization methods we investigated. Also fitting the data required into memory posed a significant challenge. We use the same updates as in the last chapter but with minor adaptations. In the following subsections, we describe modifications and techniques that made the problem tractable.

5.4.1 Selecting Triplets

As we discussed in Section 4.5, some triplets are extraneous (easy negative examples) and some can cause the set of constraints to be unsatisfiable (hard positive examples). As the in the focal learning version, extraneous triplets are harmless, except that they add to the size of the problem. This was not as much of a concern in the focal learning setting because the problems were small, but in the global learning setting we are solving a much larger optimization.

If we were to use the exhaustive set of triplets formed from every combination of reference, positive, and negative images from the training set, for an experiment using 15 images per category, we would have 15 reference images per category (1515) times the number of positive examples for each reference image ($\times 14$), times all negative examples for each of those positive pairs ($\times 1500$) for a total of about 31.8 million triplets. If the training data is too large to fit into memory, then we need to perform disk seeks within our iterations. Also, the amount of time for each iteration over the data increases linearly with the number of triplets, and the time required to run to completion or reach a good answer may increase super-linearly. We again prune the possible set of triplets using the feature-to-set distances. For an image j, for each feature, we order the other images in the training set by their feature-to-set distance and make use of the top N closest for each feature of j. Consider a positive example in this short list: we know that this positive pair is similar according at least one feature, so it is likely we could find a weighting that makes the distance small. A negative example in this short list is also a good candidate because it will probably give a constraint close to the margin, which the algorithm should focus on. Thus, given that image i was in the short list for j, we want to use the distance vector $\mathbf{d}_{ji}$ in some of our triplets. We group these pairs by the reference images (i in this case) and then form triplets from all pairs involving that reference image. We chose a depth of

$N = 5$ without testing other parameter choices, and found it to give a good reduction in the number of triplets. For 15 images per category, we reduce the set of triplets by roughly half, to “only” 15.7 million.

5.4.2 Arranging the Data

Again we organize the dual updates into epochs and sweeps, as described in Section 4.4.4, but within each sweep, we arrange the data to further speed computation. In the focal version, we could compute off-line the difference of the distance vectors for each triplet (the $\mathbf{x}_{jk}$ vector introduced in Equation 4.7), and we could iterate through these in any order. While there are many triplets and an $\mathbf{x}_{jk}$ vector for each, the individual learning problems are small, so it is reasonable to precompute these and hold them in memory.

This is not possible in the global version because the $\mathbf{d}$ vectors involved in a single constraint can only be combined after they are multiplied by their weight vectors, and each time a triplet is revisited, the weight vectors may be changed. Instead, we need to store the the $\mathbf{d}$ vector for each pair of images that are included in the chosen set of triplets, and we structure the sweep such that we work with contiguous chunks of data. The easiest way to arrange the data for the global problem is by reference image, and inside each sweep is a set of three nested loops:

1. Loop over the reference images involved in “active” constraints, randomly ordered at the beginning of each epoch. Load all $\mathbf{d}$ vectors for which this is the reference image.

2. Loop over the “positive” training images that use this reference image and are involved in active constraints.

3. Loop over the “negative” training images that use this reference image and are

involved in active constraints.

Inside the inner loop, we process the update for one active constraint. If we cannot load all the data for the entire optimization into memory, we can at least load all the data for a set of updates at once, reducing the number of disk seeks inside the inner loop. If we can load all the data into memory, this still reduces the number of cache misses in the inner loop.

The dual updates we use are very similar to the updates used in on-line algorithms (e.g., [Crammer *et al.*, 2006]), and regular structure in the data can skew the performance or the convergence of these algorithms. While we process all constraints for one reference image at a time, updates are performed to the weight vectors for several training images. Our hope is that any affect regularity in the data order has on convergence is outweighed by the speed at which we can process the data.

5.4.3 Early Stopping

The same KKT conditions (Section 4.4.3) apply as in the focal version, and the dual solver terminates when it can make a full pass over all constraints without any updates. A given constraint may not change because either (1) it has satisfied the KKT conditions within some precision [Platt, 1998], or (2) the update to the dual variable falls below a threshold for a "useful" update (we use the threshold from [Platt, 1998]). The solver often stops before full convergence, but for large data set sizes it still takes a long time to run. Using 5 images per category, the optimization ran to completion in about 11 minutes. Using 15 images per category, it took 10 hours, and with 20 images, approximately 16 hours.

An advantage of the dual solver is that, like online learning methods such as [Shalev-Shwartz *et al.*, 2004], it finds a near-optimal solution very quickly. We record the value of the dual objective after every pass over the data, and we use the rate of change of the dual as an indicator of progress; when the rate of change of the dual becomes small (*e.g.* 0.001% of the value of the dual), most of the progress has already been made, and we can use the weights learned up to that point. In practice this works well; with 15 images per category, the recognition performance was the same using weights taken after running to completion (10 hours) and weights sampled after one hour of training. The results reported in this paper for 20 images per category are from our only sample of the weights, taken after about $2\frac{1}{2}$ hours of training.

5.4.4 Setting the Trade-Off Parameter

As in the last chapter, our algorithm has one free parameter, the trade-off parameter C in Equations 5.5 and 5.6, and it plays a crucial role. Using a large value for C might put too much emphasis on the empirical loss which often results in over-fitting the training set. An excessively small value as the choice of C typically yields an over-regularized setting which leads again to poor performance in practice. A popular and practical approach for choosing C is to run the full learning procedure with multiple suggestions for C on a held-out portion of the training set, also called a validation set. This approach entertains some formal properties [Kearns and Ron, 1999] and often yields very good results in practice. However, the approach is quite time consuming as it requires running the training algorithm several times for different partitions of the data.

Due to the size of our problem we chose an alternative approach which is based on recent advances in research on online learning algorithms and fits nicely with our dual formulation of the problem. For a choice of C, we make *one pass* over our set of triplets, and for each triplet i, j, k, we (1) evaluate the loss for that example using the formula $[1 - \mathbf{W} \cdot \mathbf{X}_{ijk}]_+$ then (2) make an update to its α_{ijk} dual variable (effectively updating the weight vector $\mathbf{W}$). In this way, every example in the set serves once as a “held-out” example before contributing to the model. By taking the average loss across the training examples after one sweep through the data, we get a number that we can compare to runs with different values of C. The value of C that gives the smallest average loss is the parameter that we use to run the full learning algorithm. The predictions of this online algorithm, known as Passive-Aggressive [Crammer *et al.*, 2006], are guaranteed to be competitive with the predictions of the optimal solution of Eq. 5.5. Though we did not make use of it, the vector W obtained after a single online pass through the training set can serve as a very good initialization for the batch optimization process. We sampled values of C very coarsely, at half-orders of magnitude. For the data set using 20 images per category, each C test took only a couple minutes.

5.5 Caltech101 Experiments

We performed experiments on the Caltech 101 data set, similar to those in the last chapter. We visualize the learned weights using colored circles; the circles are centered on the patches (though are not the same size as the patches used to compute the features), and the color illustrates the weight of the patch. Figure 3.9 showed three such hand-picked results from the 15 images/category experiments for one of the features. In Figure 5.3, we show the weights for all three types of features, for a randomly-selected sample of 1,515 training images. The colors are on the Matlab jet scale, where the largest value is in dark red, and the gradient runs from warm to cool, to dark blue as the smallest non-zero weight. Features with zero weight are not shown. The color mapping is normalized per image, so the colors can be compared across features for a given image, but cannot be directly compared across images.

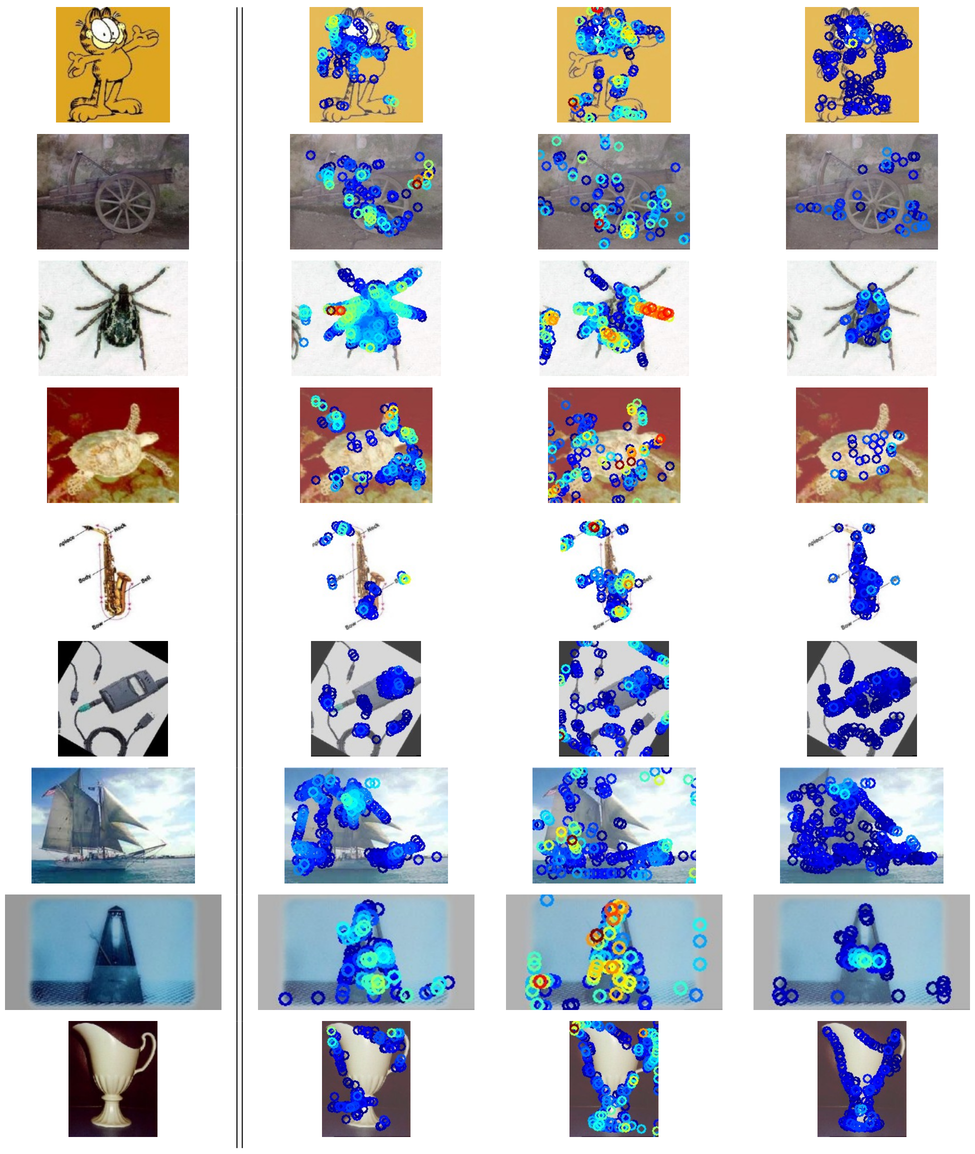


Figure 5.3: Weights learned for a randomly-selected sample of 1,515 training images. The original images are shown in the left column, and the right three columns illustrate weights assigned to the three types of features described in Section 3.6. In order from left to right they are the big geometric blur feature, the small geometric blur, and the color feature. The centers of the features are marked by the circles, though they are not scaled to the size of the feature extents. The color indicates the relative value of the weight, using the Matlab jet scale (shown in Figure 5.5). For each image, the weights are normalized; dark red is the largest weight assigned to any feature for the image, and the scale progresses from warm to cool colors with dark blue indicating the smallest non-zero weights. The weights are not normalized across images. Features assigned a zero weight are not shown.

5.5.1 Retrieval

The distances from training images to a query image can be used directly to perform retrieval. In Figure 5.4 we show retrieval results for a randomly-selected subset of test images. These are from the 15 images/category experiment, and were chosen from among the 7,162 available test images.

5.5.2 Classification

We performed experiments using 5, 10, 15, and 20 images per category, using the remaining images in the dataset for testing, as in Section 4.6. For each test image, we order the training images according to their distance to the test image using their learned distance functions. We performed classification using the modified 3-NN algorithm described at the end of Section 4.8.2. The number of test images varies between classes, with some of the easiest classes having the greatest number, so we compute our average recognition as in [Grauman and Darrell, 2006b] by first computing the percentage correct for each class, and then averaging those numbers to get mean recognition. This is equivalent to computing the average of the diagonal of the confusion matrix. We use all 101 categories in training and testing, but do not make use of the background class. Most results to date have been reported using all 101 categories except [Griffin *et al.*, 2007] which omits the `Faces_easy` category, a confuser for the `Faces` category.

The graph in Figure 1.2 shows the results using the global learning with several of the results published in the last few years. At five and ten images per category, we perform below the best results from [Zhang *et al.*, 2006], and cross somewhere between ten and fifteen images per category. At fifteen images per category, we achieve a recognition rate of 63.2%, about 3% better than the best published results, and at 20, we achieve 66.6%, almost 10% better than the next best result that trains

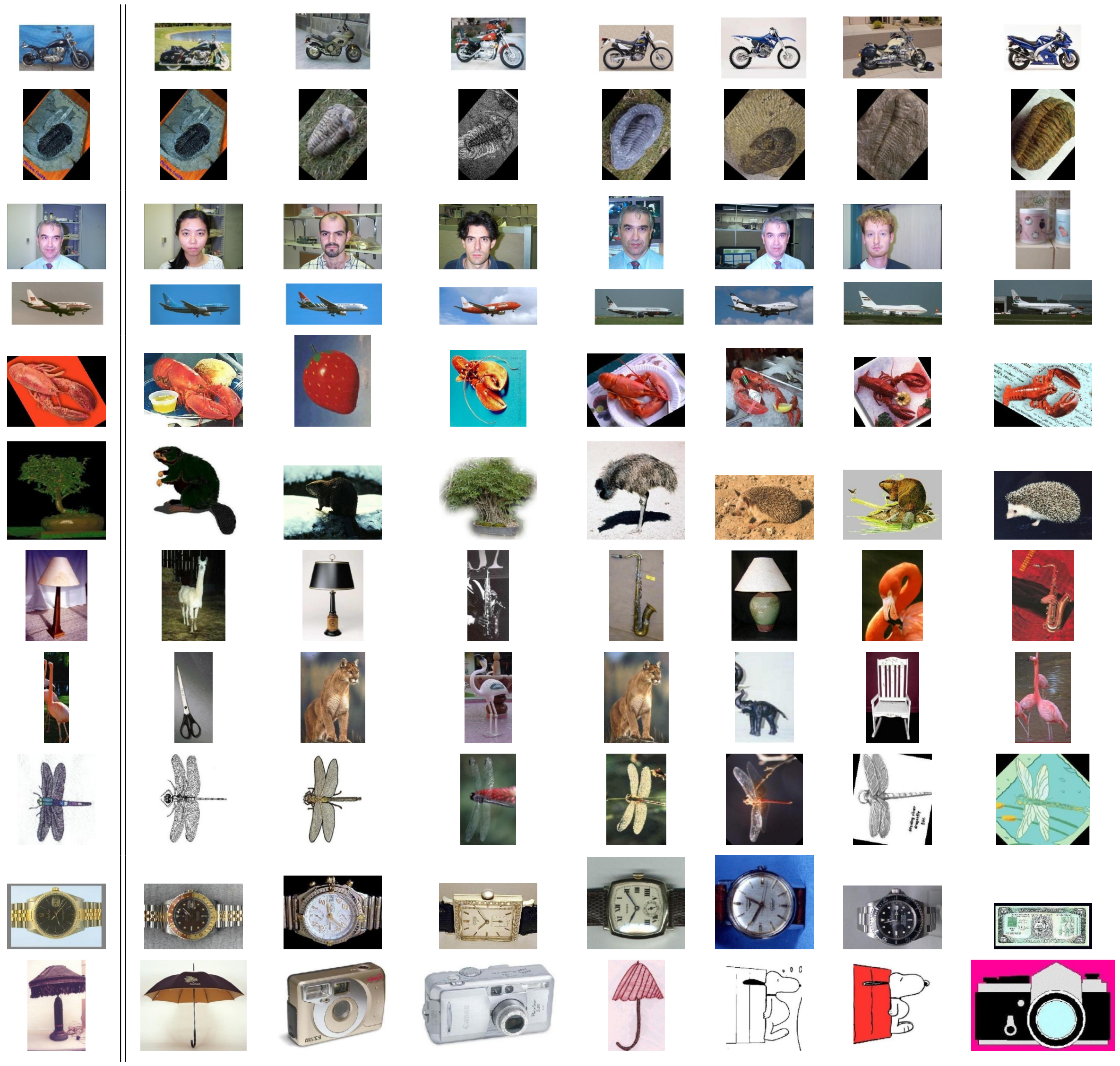


Figure 5.4: A randomly-selected sample of retrieval results from the 15 images/category experiment. The left column are test images, and the row to the right of each shows the top seven ranked training images according to the learned distance functions.

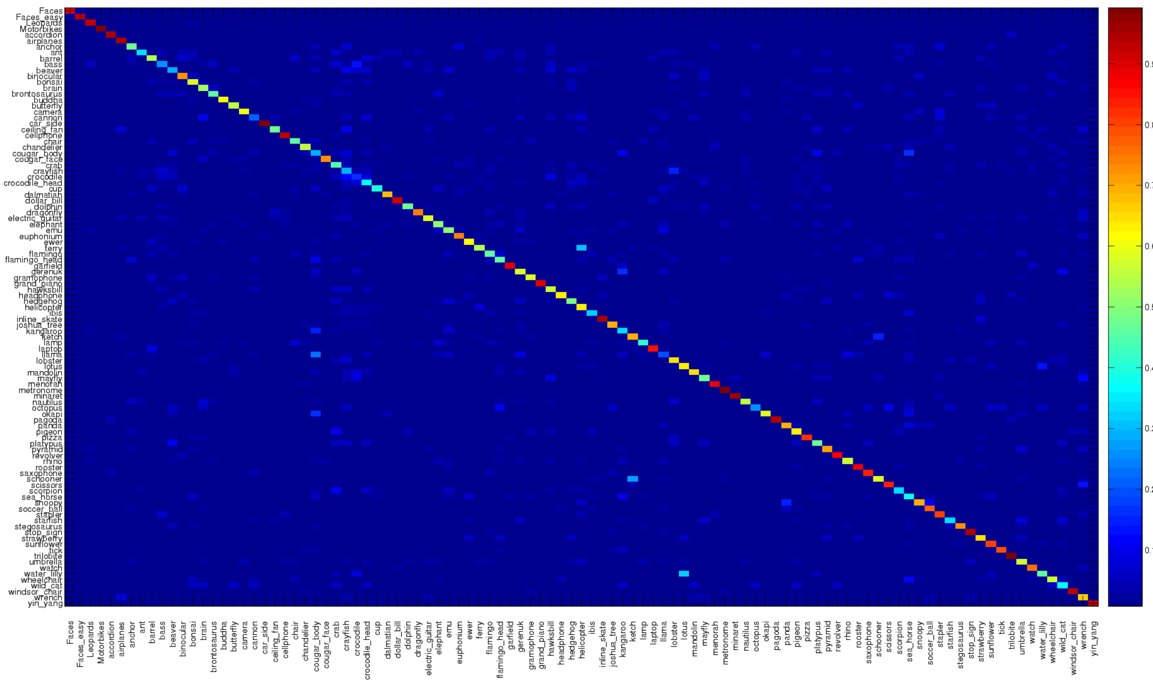


Figure 5.5: Confusion matrix for 15 images per category, shown using the jet color scale from Matlab. Dark red indicates 100% while dark blue indicates 0%, with a gradient from warm to cool colors in between (see scale, right). A perfect matrix would be dark blue matrix except for a dark red diagonal.

and tests on all 101 categories. It makes sense that our performance would increase dramatically with the number of categories: we use a nearest-neighbor classifier, plus at the core of our method is an assumption that there are pairs of similar images in the training set. This is more likely to be true as the number of training images grows.

All other approaches other than [Frome *et al.*, 2006] use only shape features, whereas our work and [Frome *et al.*, 2006] make use of rudimentary color features. Most of our performance is gained from the shape features; at 15 images per category, the color features add only 2% to the mean recognition. At the same time, the fact that these simple color features improve performance at all demonstrates that our method is able to naturally combine features of very different types. In Figure 5.5, we show our confusion matrix for 15 images per category.

Chapter 6

Conclusion: What Is Missing

The main contribution of this thesis is an algorithm for learning weights for features in training images and using the weighted feature sets to perform exemplar-based categorization. Fellow researchers have argued that exemplar-based methods will lose out to approaches that generalize at the class level, and that with so many parameters to learn, such an approach is doomed to overfit. Perhaps the true contribution of this work is just the demonstration that it *can* work well. Still, there is much missing from this approach and the recent work in the field of object recognition. The goal of this chapter is simply to point out some of what is missing.

If our ultimate goal is to replicate animal or human visual systems, some of the issues we need to tackle are:

1. objects can be observed at different distances, which, from the observer's standpoint, is seen as a change in scale;

2. 3D objects can be observed in different poses, the effect of out-of-plane rotation and, in the case of non-rigid objects, articulation; most commonly these variations are addressed by training with a set of 2D images that sample the possible poses;

3. objects exist as part of a larger scene, which requires first detecting the object of interest and also makes it possible to use the surrounding context to aid in recognition; and

4. objects are part of categorical hierarchies

As we discussed in the first chapter, machine object recognition is defined by the data we use to train and test, and the difficulty inherent in the data set will shape our algorithms. In the last four years, many researchers have focused on the Caltech 101 data set, which has little rotational, scale, pose, or positional variation, and most of the recent approaches assume and exploit at least one of those regularities. Our work is no exception. It is generally believed that scale invariance is the easiest to tackle, just computationally expensive, so if the data set does not require it to be handled, it is neatly swept under the rug. Approaches such as [Lazebnik *et al.*, 2006], [Mutch and Lowe, 2006], and [Zhang *et al.*, 2006] use the rough absolute positions of the features, thus exploiting the regular positions of objects in the frame, whereas approaches such as ours do not use any positional information, so they are not guilty of exploiting positional regularity, but they are also ignoring valuable geometric information. In fact, none of the new approaches since [Berg *et al.*, 2005] shown in 1.2 have provided a way of incorporating relative geometric relationships between features. Lastly, the data set is constructed so that there is ostensibly only one object in each image, though it is more correct to say there is only one right answer for each image. This is not necessarily a flaw; perhaps it is a fair assumption if we are automatically labeling web images for a search engine since, after all, these images were gathered by scraping a web image search engine. But perhaps this task should more properly be called "image classification" instead of "object recognition". All of the top-performing algorithms exploit this aspect of the data set.

The Caltech 256 data set seeks to address some of the shortcomings of the Caltech

101 data set, though it is still an "image classification" task. It has more rotational, scale, and positional variation, and more than double the number of classes, which should help to weed out approaches that are too computationally expensive to scale to larger numbers of classes. Approaches such as [Grauman and Darrell, 2006a] and [Lazebnik *et al.*, 2006] which do not compare all features to one another are better suited to larger data sets, and the method of [Lazebnik *et al.*, 2006] is the only approach that we know of to date for which there are published results on Caltech 256 [Griffin *et al.*, 2007]. By using the approximate techniques in Chapter 2 such as LSH to compute the feature-to-image distances used in the distance learning, we may be able to greatly increase the size of the data set we can process with our method as well. However, all of these ultimately scale linearly with the number of classes. If we are to believe that humans recognize about 30,000 visual categories [Biederman, 1987], and this is our ultimate goal, then we have a long way to go still.

For those that wish to pursue object recognition in a setting closer to human object recognition, there are the VOC (PASCAL) [1] and TRECVID [2] data sets, which have been introduced as part of object recognition competitions. The VOC 2006 data set contains ten classes (bicycles, buses, cats, cars, cows, dogs, horses, motorbikes, people, and sheep) with a great deal of appearance, pose, scale, and positional variation, and many images contain more than one target object. The TRECVID data set is of a similar nature, with 39 different categories, though some of them (such as `Natural-Disaster`) are not really visual categories. It would be possible to apply approaches such as ours to these types of data sets, perhaps by using a "sliding window" approach, though there is significant work to be done to make it computationally efficient. If our ultimate goal is to design general object recognition systems, we need to focus on efficiently recognizing multiple objects in larger scenes.

[1] http://www.pascal-network.org/challenges/VOC/databases.html

[2] http://www-nlpir.nist.gov/projects/t01v/

Perhaps what is missing is an efficient way to gather a good sample of natural scenes representing more than 200 visual classes.

Perhaps our best hope for large-scale object recognition lies in part with hierarchical categorization. In categorization tasks that involve very large numbers of categories, it should be possible to make very rough distinctions very quickly and focus our learning efforts on the finer distinctions. In Chapter 3 we made the argument that which features are important varies with the image being considered, but we ignored that the importance of features also depends upon to what the image is being compared. The features that are important for telling a `ketch` (a type of sailboat) from a `flamingo` are probably very different from those used to tell it apart from a `schooner` (another type of sailboat). Our approach does not address this; it tries to sort out all distinctions all at once. The approach in [Zhang and Malik, 2003], and in general, multi-way classification methods based on pairwise classifiers, are closer in spirit to a hierarchical approach in that they learn parameters to tell apart pairs of classes, though training $\frac{N(N-1)}{2}$ classifiers as N approaches 30,000 is not realistic. Perhaps the next frontier are algorithms that separate out the easy from the hard decisions inside the loop. Perhaps then we can finally begin to address superordinate, basic-level, and subordinate categories in a manner closer to humans.

Bibliography

[Allwein *et al.*, 2000] Erin L. Allwein, Robert E. Schapire, and Yoram Singer. Reducing multiclass to binary: A unifying approach for margin classifiers. *JMLR*, 1:113–141, 2000.

[Arya *et al.*, 1998] Sunil Arya, David M. Mount, Nathan S. Netanyahu, Ruth Silverman, and Angela Y. Wu. "an optimal algorithm for approximate nearest neighbor searching fixed dimensions". *Journal of the ACM*, 45(6):891–923, November 1998.

[Belongie *et al.*, 2001] Serge Belongie, Jitendra Malik, and Jan Puzicha. Matching shapes. In *Eighth IEEE International Conference on Computer Vision*, volume 1, pages 454–461, July 2001.

[Belongie *et al.*, 2002] Serge Belongie, Jitendra Malik, and Jan Puzicha. Shape matching and object recognition using shape contexts. *PAMI*, 24(4):509–522, April 2002.

[Bengio *et al.*, 2004] Yoshua Bengio, Jean-François Paiement, Pascal Vincent, Olivier Delalleau, Nicolas Le Roux, and Marie Ouimet. Out-of-sample extensions for lle, isomap, mds, eigenmaps, and spectral clustering. In *NIPS*, 2004.

[Berg and Malik, 2001] Alexander Berg and Jitendra Malik. Geometric blur for template matching. In *CVPR*, pages 607–614, 2001.

[Berg *et al.*, 2004] Alexander C. Berg, Tarama L. Berg, and Jitendra Malik. Shape matching and object recognition using low distortion correspondence. Technical Report UCB//CSD-04-1366, University of California Berkeley, Computer Science Division, Berkeley, California, USA, December 2004.

[Berg *et al.*, 2005] Alexander Berg, Tamara Berg, and Jitendra Malik. Shape matching and object recognition using low distortion correspondence. In *CVPR*, 2005.

[Biederman, 1987] Irving Biederman. Recognition-by-components: A theory of human image understanding. *Psychological Review*, 94(2):115–147, 1987.

[Censor and Zenios, 1998] Yair Censor and Stavros A. Zenios. *Parallel Optimization: Theory, Algorithms, and Applications*. Oxford University Press, 1998.

[Crammer and Singer, 2001] Koby Crammer and Yoram Singer. On the algorithmic implementation of multiclass kernel-based vector machines. *JMLR*, 2:265–292, 2001.

[Crammer *et al.*, 2006] Koby Crammer, Ofer Dekel, Joseph Keshet, Shai Shalev-Shwartz, and Yoram Singer. Online passive aggressive algorithms. volume 7, Mar 2006.

[Dietterich and Bakiri, 1995] T. Dietterich and G. Bakiri. Solving multiclass learning problems via error-correcting output codes. pages 263–286, January 1995.

[Fei-Fei *et al.*, 2004] Li Fei-Fei, Rob Fergus, and Pietro Perona. Learning generative visual models from few training examples: an incremental bayesian approach testing on 101 object categories. In *Workshop on Generative-Model Based Vision, CVPR*, 2004.

[Fergus *et al.*, 2003] R. Fergus, P. Perona, and A. Zisserman. Object class recognition by unsupervised scale-invariant learning. In *CVPR*, 2003.

[Friedman, 1996] J. Friedman. Another approach to polychotomous classification. Technical report, Stanford University, 1996.

[Frome *et al.*, 2004] Andrea Frome, Daniel Huber, Ravi Kolluri, Thomas Bülow, and Jitendra Malik. Recognizing objects in range data using regional point descriptors. In *ECCV*, volume III, pages 224–237, May 2004.

[Frome *et al.*, 2006] Andrea Frome, Yoram Singer, and Jitendra Malik. Image retrieval and classification using local distance functions. In *NIPS*, 2006.

[Gionis *et al.*, 1999] Aristides Gionis, Piotr Indyk, and Rajeev Motwani. Similarity search in high dimensions via hashing. In *Proceedings of 25th International Conference on Very Large Data Bases (VLDB)*, pages 518–529, 1999.

[Globerson and Roweis, 2005] Amir Globerson and Sam Roweis. Metric learning by collapsing classes. In *NIPS*, 2005.

[Grauman and Darrell, 2006a] Kristen Grauman and Trevor Darrell. Approximate correspondences in high dimensions. In *NIPS*, 2006.

[Grauman and Darrell, 2006b] Kristen Grauman and Trevor Darrell. Pyramic match kernels: Discriminative classficiation with sets of image features (version 2). Technical Report MIT_CSAIL_TR_2006-020, MIT, March 2006.

[Griffin *et al.*, 2007] G. Griffin, A. Holub, and P. Perona. Caltech-256 object category dataset. Technical Report UCB/CSD-04-1366, California Institute of Technology, 2007.

[Hastie and Tibshirani, 1997] Trevor Hastie and Robert Tibshirani. Classification by pairwise coupling. In *NIPS*, 1997.

[Holub *et al.*, 2005] Alex D. Holub, Max Welling, and Pietro Perona. Combining generative models and fisher kernels for object recognition. In *ICCV*, 2005.

[Huber and Hebert, 2003] Daniel F. Huber and Martial Hebert. Fully automatic registration of multiple 3D data sets. *Img. and Vis. Comp.*, 21(7):637–650, July 2003.

[Huttenlocher and Ullman, 1990] Daniel P. Huttenlocher and Shimon Ullman. Recognizing solid objects by alignment with an image. *IJCV*, 5(2):195–212, November 1990.

[Indyk and Motwani, 1998] P. Indyk and R. Motwani. Approximate nearest neighbor - towards removing the curse of dimensionality. In *Proceedings of the 30th Symposium on Theory of Computing*, pages 604–613, 1998.

[J. Ponce and Zisserman, 2006] M. Everingham D. A. Forsyth M. Hebert S. Lazebnik M. Marszalek C. Schmid B. C. Russell A. Torralba C. K. I. Williams J. Zhang J. Ponce, T. L. Berg and A. Zisserman. *Toward Category-Level Object Recognition*, chapter Dataset Issues in Object Recognition. 2006.

[Johnson and Hebert, 1999] Andrew E. Johnson and Martial Hebert. Using spin images for efficient object recognition in cluttered 3d scenes. *PAMI*, 21(5):433–449, 1999.

[Kay and McDaniel, 1978] Paul Kay and Chad K. McDaniel. The linguistic significance of the meanings of basic color terms. *Language*, 54:610–646, 1978.

[Kazhdan *et al.*, 2003] Michael Kazhdan, Thomas Funkhouser, and Szymon Rusinkiewicz. Rotation invariant spherical harmonic representation of 3d shape descriptors. In *Proceedings of the Eurographics/ACM SIGGRAPH symposium on Geometry processing*, pages 156–164. Eurographics Association, 2003.

[Kearns and Ron, 1999] Michael Kearns and Dana Ron. Algorithmic stability and sanity-check bounds for leave-one-out cross-validation. *Neural Computation*, 11:1427–1453, 1999.

[Lazebnik *et al.*, 2006] Svetlana Lazebnik, Cordelia Schmid, and Jean Ponce. Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories. In *CVPR*, 2006.

[Liu *et al.*, 2004] Ting Liu, Andrew W. Moore, Alex Gray, and Ke Yang. An investigation of practical approximate nearest neighbor algorithms. In *Advances in Neural Information Processing Systems*, December 2004.

[Lowe, 1999] David Lowe. Object recognition from local scale-invariant features. In *ICCV*, pages 1000–1015, Sep 1999.

[Martin *et al.*, 2004] David Martin, Charless Fowlkes, and Jitendra Malik. Learning to detect natural image boundaries using local brightness, color and texture cues. *TPAMI*, 26(5):530–549, May 2004.

[Mervis and Rosch, 1981] Carolyn Mervis and Eleanor Rosch. Categorization of natural objects. *Annual Review of Psychology*, 32:89–115, 1981.

[Mori and Malik, 2003] Greg Mori and Jitendra Malik. Recognizing objects in adversarial clutter: Breaking a visual captcha. In *CVPR*, 2003.

[Mori *et al.*, 2001] Greg Mori, Serge Belongie, and Jitendra Malik. Shape contexts enable efficient retrieval of similar shapes. In *CVPR*, volume 1, pages 723–730, 2001.

[Mutch and Lowe, 2006] Jim Mutch and David G. Lowe. Multiclass object recognition with sparse, localized features. In *CVPR*, 2006.

[Platt *et al.*, 2000] John C. Platt, Nello Cristianini, and John Shawe-Taylor. Large margin dags for multiclass classification. In *NIPS*, 2000.

[Platt, 1998] John Platt. *Advances in Kernel Methods - Support Vector Learning*, chapter Fast Training of Support Vector Machines using Sequential Minimal Optimization, pages 185–208. MIT Press, 1998.

[Rosch and Mervis, 1975] Eleanor Rosch and Carolyn Mervis. Family resemblances: Studies in the internal structure of categories. *Cognitive Psychology*, 7:573–605, 1975.

[Rosch, 1978] Eleanor Rosch. *Cognition and Categorization*, chapter Principles of Categorization, pages 27–48. Erlbaum, 1978.

[Roweis and Saul, 2000] Sam Roweis and Lawrence Saul. Nonlinear dimensionality reduction by locally linear embedding. *Science*, 290(5500):2323–2326, December 2000.

[Ruiz-Correa *et al.*, 2003] Salvador Ruiz-Correa, Linda Shapiro, and Marina Miela. A new paradigm for recognizing 3d object shapes from range data. In *ICCV*, Oct 2003.

[Schmid and Mohr, 1996a] C. Schmid and R. Mohr. Combining greyvalue invariants with local constraints for object recognition. In *CVPR*, pages 872–877, June 1996.

[Schmid and Mohr, 1996b] C. Schmid and R. Mohr. Image retrieval using local characterization. In *International Conference on Image Processing*, 1996.

[Schultz and Joachims, 2003] Matthew Schultz and Thorsten Joachims. Learning a distance metric from relative comparisons. In *NIPS*, 2003.

[Serre *et al.*, 2005] Thomas Serre, Lior Wolf, and Tomaso Poggio. Object recognition with features inspired by visual cortex. In *CVPR*, 2005.

[Shalev-Shwartz *et al.*, 2004] Shai Shalev-Shwartz, Yoram Singer, and Andrew Ng. Online and batch learning of pseudo-metrics. In *International Conference on Machine Learning*, 2004.

[Tenenbaum *et al.*, 2000] J. B Tenenbaum, V. de Silva, and J. C. Langford. A global geometric framework for nonlinear dimensionality reduction. *Science*, 290(5500):1319–2323, December 2000.

[Viola and Jones, 2004] Paul Viola and Michael J. Jones. Robust real-time face detection. *IJCV*, 57(2):137–154, May 2004.

[Wang *et al.*, 2006] G. Wang, Y. Zhang, and L. Fei-Fei. Using dependent regions for object categorization in a generative framework. In *CVPR*, 2006.

[Weinberger *et al.*, 2005] Kilian Q. Weinberger, John Blitzer, and Lawrence K. Saul. Distance metric learning for large margin nearest neighbor classification. In *NIPS*, 2005.

[Wittgenstein, 2001] Ludwig Wittgenstein. *Philosophical Investigations.* Blackwell Publishing, 3rd edition, 2001.

[Xing *et al.*, 2002] Eric Xing, Andrew Ng, and Michael Jordan. Distance metric learning with application to clustering with side-information. In *NIPS*, 2002.

[Zhang and Malik, 2003] Hao Zhang and Jitendra Malik. Learning a discriminative classifier using shape context distances. In *CVPR*, 2003.

[Zhang *et al.*, 2006] Hao Zhang, Alex Berg, Michael Maire, and Jitendra Malik. Svm-knn: Discriminative nearset neighbor classification for visual category recognition. In *CVPR*, 2006.