University of California

Los Angeles

Finding Categories

A dissertation submitted in partial satisfaction
of the requirements for the degree
Doctor of Philosophy in Computer Science

by

Brian Daniel Fulkerson

2010

The dissertation of Brian Daniel Fulkerson is approved.

[illegible]

Mark Hansen

[illegible]

Demetri Terzopoulos

[illegible]

Deborah Estrin

[illegible]

Stefano Soatto, Committee Chair

University of California, Los Angeles

2010

Table of Contents

List of Figures

List of Tables

Acknowledgments

My advisor Stefano Soatto has consistently provided a focused lens to view problems I've approached. His ability to quickly assess and bring detailed insights has often provided clarity when I was uncertain a solution was possible.

Much of the research in this thesis would have been impossible without my frequent collaborator Andrea Vedaldi. Our software collaborations especially have been among some of my most useful lessons in grad school. His "never give up" mantra kept me inspired and working many late nights before deadlines, and our research discussions have always seemed to lead to more fresh ideas than anyone could reasonably hope.

I'm grateful to the members of the Golem Group for giving me a chance to participate in a crazy robot race through the desert. Collaborations with members of the group, including but certainly not limited to Jim Radford, Eagle Jones, and Bill Caldwell have solidified my views on systems and software development.

Thanks to my friend Kamil Wnuk for convincing me to start surfing. I will always remember the vision lab surf colloquiums, which have produced more than a few good ideas.

I want to thank all of the UCLA Vision Lab members with whom I've been fortunate enough to interact. Thanks to Jeremi Sudol, Teresa Ko, Michalis Raptis, Alper Ayvaci, Ganesh Sundaramoorthi, Jason Meltzer, and all the other lab members who provided an endless supply of good feedback and advice.

Louka Dlagnekov selflessly provided his editing services for this thesis, in addition to his friendship, encouragement, and advise.

And of course, I want to thank my family. Their unwavering support and confidence in me have always been an immeasurable source of strength.

Vita

1982	Born, San Diego, California, USA.
2004	B.S., Computer Science and Engineering University of California, San Diego
2004–2006	DARPA Grand Challenge Golem Group Southern California
2006	M.S. Computer Science University of California, Los Angeles
2010	Ph.D. Computer Science University of California, Los Angeles

Publications and Presentations

M. Allen, R. Vargas, E. Graham, W. Swenson, M. Hamilton, M. Taggart, T. Harmon, A. Rat'Ko, P. Rundel, B. Fulkerson, and D. Estrin. "Soil Sensor Technology: Life within a Pixel". BioScience, Vol. 57, No. 10, November 2007.

B. Fulkerson and S. Soatto. "Really Quick Shift: Image Segmentation on a GPU". In Proceedings of the Workshop on Computer Vision using GPUs, held with the European Conference on Computer Vision, September 2010.

B. Fulkerson, A. Vedaldi, and S. Soatto. "Class Segmentation and Object Localization with Superpixel Neighborhoods". In Proceedings of the International Conference on Computer Vision, 2009.

B. Fulkerson, A. Vedaldi, and S. Soatto. "Localizing Objects With Smart Dictionaries". In Proceedings of the European Conference on Computer Vision, 2008.

E. Jones, B. Fulkerson, E. Frazzoli, D. Kumar, R. Walters, J. Radford, and R. Mason. "Autonomous off road driving in the DARPA Grand Challenge". In Proceedings of the IEE/ION Position, Location, and Navigation Symposium, April 2006.

R. Mason, J. Radford, D. Kumar, R. Walters, B. Fulkerson, E. Jones, D. Caldwell, J. Meltzer, Y. Alon, A. Shashua, H. Hattori, N. Takeda, E. Frazzoli, and S. Soatto. "The Golem Group / UCLA Autonomous Ground Vehicle in The DARPA Grand Challenge". Journal of Field Robotics, Special Issue on the DARPA Grand Challenge, 2005.

S. Reddy, G. Chen, B. Fulkerson, S.J. Kim, U. Park, N. Yau, J. Cho, M. Hansen, and J. Heidemann. "Sensor Internet Share and Search: Enabling Collaboration of Citizen Scientists". IPSN DSI, Cambridge, Massachusetts, 2007.

A. Vedaldi and B. Fulkerson. "VLFeat - An Open and Portable Library of Computer Vision Algorithms". In Proceedings of the 18th annual ACM international conference on Multimedia, October 2010.

A. Vedaldi and B. Fulkerson. **Tutorial:** "VLFeat - An Open and Portable Library of Computer Vision Algorithms". Part of the Open Source Vision Software, Intro and Training Tutorial, held with the Conference on Computer Vision and Pattern Recognition, San Francisco, CA, June 2010.

K. Wnuk, B. Fulkerson, and J. Sudol. "A Scalable Architecture for Multi Agent Vision Based Robot Scavenging". Proceedings of the National Conference on Artificial Intelligence (AAAI 06) Workshop on Robotics, Boston, MA, 2006.

Abstract of the Dissertation

Finding Categories

by

Brian Daniel Fulkerson

Doctor of Philosophy in Computer Science

University of California, Los Angeles, 2010

Professor Stefano Soatto, Chair

Assigning categorical labels to objects in images has proven to be a significant challenge for automated systems. As cameras rapidly proliferate our society, however, we will necessarily depend more heavily on computers to help us label and sort our images. This work addresses the problem of trying to assign categorical labels to images. We contend that to do this task effectively, we should consider also which part of the image contains the object.

We examine the sensitivity of feature detection to nuisances and propose a new feature detector based on a tree of segmentations. When a detector is not required, we describe a fast adaptation that extracts a popular descriptor (SIFT) on a dense grid on the image. Next, we show that a dictionary constructed for the task of categorization can be both smaller and more accurate than one constructed to represent the data alone. We explore splitting descriptors along segmentation boundaries, and show that knowing which part of an image contains the object can make a large difference in accuracy. With these pieces, we construct a fast and accurate pixel-level categorization technique. Then, we move from pixels to small homogeneous collections of pixels (superpixels) and exploit the neighborhood structure of these to form precise superpixel-level categorization.

Finally, the appendix discusses open software we have developed and released including a GPU implementation of a segmentation algorithm (quick shift) and a MATLAB experiment framework (Blocks) which implements the techniques described in the thesis.

CHAPTER 1

Introduction

With the constant emergence of smaller and cheaper cameras, images have become ubiquitous. In 2008 alone, one billion images were uploaded to Flickr, and Facebook members added two terabytes of photos to the service every day. Our ability to capture images has outpaced our ability to label and sort them. This thesis focuses on automatically determining what kinds of things are present in an image and where in the image they are located.

In the literature, the problem of identifying the content of an image goes by many names: object recognition, category recognition, (category) localization, and class segmentation. In object recognition, we seek to find a specific thing we have seen before, for example a specific book. Often, the location of the object in the image is often also found. Category recognition instead assumes the image contains a single object and tries to assign a category label to it (e.g. book, rather than a specific book by a specific author). When we also wish to know roughly where an instance of a specific object category is, the task is called localization. In this case, we seek to draw one (or more) boxes in the image around objects of the category of interest. Finally, class segmentation seeks to label each pixel of the image with a class or as background.

These tasks are related, and so they naturally share many common elements. Typically, the image is first reduced to a set of local *features* which describe salient parts of the images in a way that is invariant to nuisance phenomena (such as small lighting changes). This set of features and their locations form our representation of the image. When we are looking for a specific object (object recognition), we can often just look for geometric arrangements of features which are consistent with our model of the object. This requirement is relaxed when we are looking for an object class, usually by considering histograms of quantized features (*bags of features* [CDD04]). Sometimes, weak geometric constraints are imposed on the configuration of the features (e.g. [LSP06]). Any of these approaches can be adapted to localization in a straightforward fashion by passing a fixed size window over the image and computing a score for the class of interest at each pixel. The pixel with the best score provides the location of the object class. The approaches can be similarly adapted to class segmentation by giving each pixel a label corresponding to the window of features around that pixel.

The next section provides an introduction to the bag of features pipeline as it applies to categorization. In subsequent chapters we examine and build on each piece of the system. In Chapter 2, we define and examine sensitivity measures for feature detection in the context of recognition tasks. Chapter 3 investigates using boundary information from segmentations to split features along occluding object boundaries for the purpose of improving image categorization, and shows that it is important to consider the extent of the object when attempting to do categorization. Chapter 4 focuses primarily on improving the quality of the dictionary common to these tasks. It then takes the resulting general purpose discriminative dictionaries, couples them with a novel extraction method for fast dense SIFT [Low04] features and produces a pixel-level localization method. Chapter 5 addresses the issue of how to include spatial information in the pipeline. It presents a class segmentation problem, and uses a conditional random field (CRF) framework on classifiers built on *neighborhoods* of superpixels to achieve robust, detailed results. In Appendix A, we present a GPU implementation of

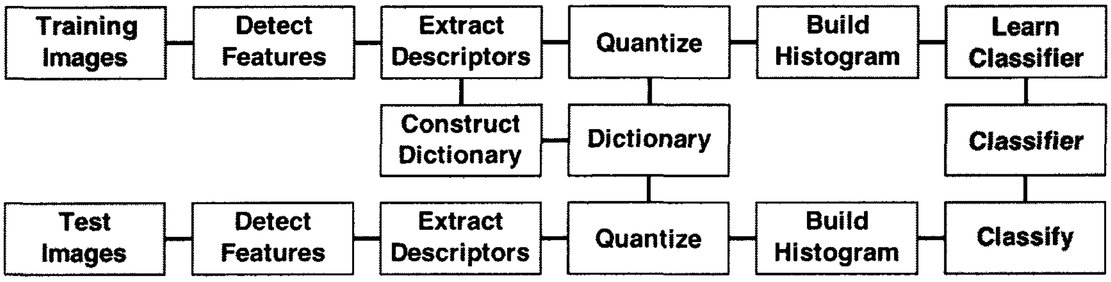


Figure 1.1: **Bag of Features Pipeline.** Here we show the bag of features pipeline. The color indicates the major area of each step. Blue boxes indicate data, which is partitioned into training and testing sets. The light orange boxes involve features, which are extracted from both sets. Yellow boxes correspond to steps involving the dictionary, including building the dictionary during training and quantizing the data using the dictionary for both training and testing. Green boxes show histogram construction, and orange boxes denote the classifier.

quick shift [VS08], a mode seeking segmentation algorithm which we use to obtain superpixels in Chapter 5. Finally, we provide reference implementations of all algorithms described in this thesis as part of two open source projects: VLFeat [VF08, VF10] and Blocks [FV09] (described further in Appendix B).

1.1 Categorization with Bags of Features

In this section, we provide the reader with a review of bag of features classification [CDD04]. This will form a basic tool that we will modify and use throughout the thesis. The application we will focus on is object category recognition, though the process can be easily extended to all of the other problems we are interested in. A color coded block diagram of the process is shown in Figure 1.1.

1.1.1 Features

A large area of research in computer vision focuses on decomposing an image into a set of *features* which are invariant (or at least insensitive) to transformations of the image. By adopting these features as our representation of the image, we are able to eliminate some nuisance factors (such as small changes in pose or lighting) before we proceed with matching. Features that find and describe local image regions are by far the most common, though other types are possible (for example, features which describe a contour [SBC05] or an edge [MS08]).

The *feature detector* seeks to find points of an image that may be reliably found in another image. How easily this feature can be found is known as its repeatability. Two common examples of feature detectors are the Scale Invariant Feature Transform (SIFT [Low04]) and Maximally Stable Extremal Regions (MSER [MCU02]). The output of the detector is typically a set of locations in the image and associated regions (often defined by an ellipse). Some example SIFT detections are shown in Figure 1.2. In Chapter 2, we propose and examine two measures of sensitivity of feature detection for recognition tasks.

Once we have found a set of features in an image, the *feature descriptor* seeks to form a description of each region that will provide insensitivity to a set of transformations and facilitate the matching process. In the bag of features setting, the set of these *descriptors* are the final representation of the image. All spatial information is discarded, as though the descriptors have been put in a "bag", and comparing images is the same as comparing the contents of the bag which represents each image.

In Chapter 4 we propose a SIFT-like descriptor which may be quickly extracted on a grid (potentially each pixel of the image). Our publicly available library VLFeat [VF08] implements this descriptor as well as SIFT and the MSER

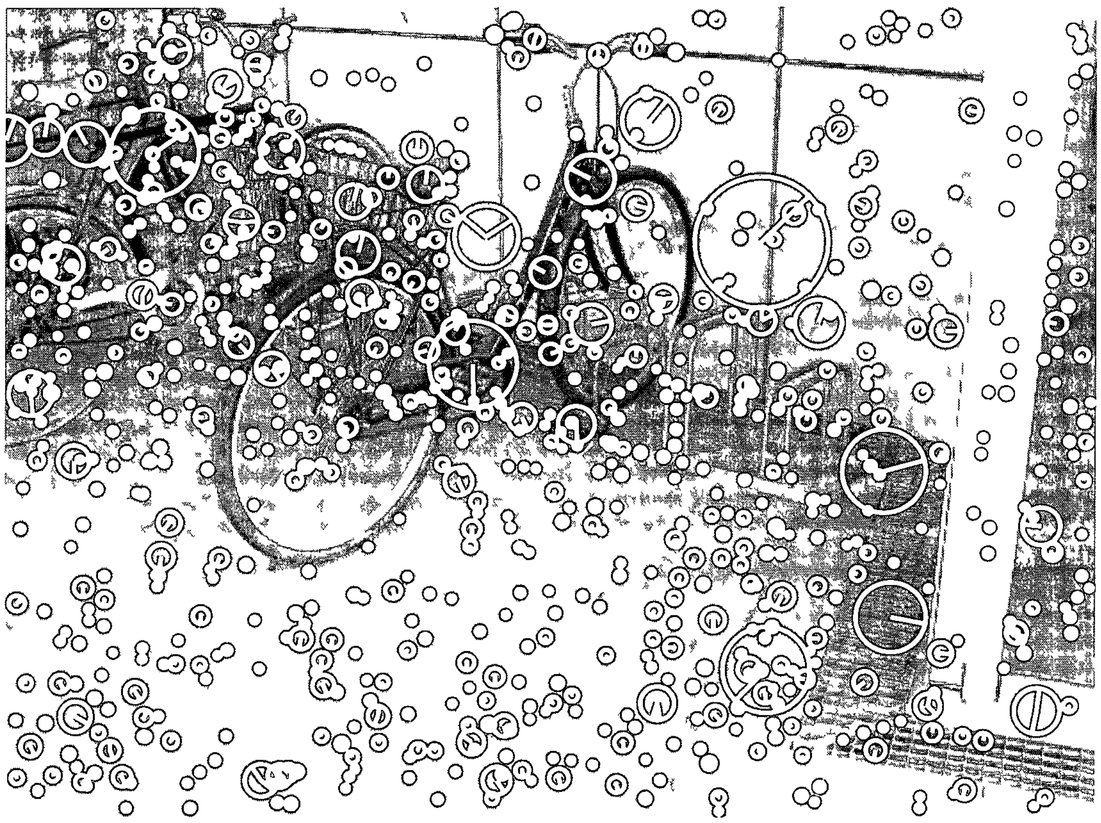


Figure 1.2 SIFT Detections. A sample image from the bike category of the Graz-02 dataset and its detected SIFT features. The scale of each feature is the radius of the circle, and the orientation of the feature is denoted by a line through its center.

detector. In Chapter 3 we explore "splitting" SIFT descriptors at possible occlusion boundaries found using a segmentation of the image.

1.1.2 Dictionary

The fundamental assumption of bag of features classifiers is that the image may be represented by a distribution of local features. However, the feature space is often quite large, and the number of samples relatively small. With SIFT, descriptors are vectors in $\mathbb{R}^{128}$. If we use the SIFT detector we may have approximately 1000 descriptors per image. In order to compute a statistic on this space, we assume its intrinsic dimension is much smaller than $\mathbb{R}^{128}$ and instead compute the distribution of features on this reduced space. Often, this reduced space is discrete: we form a dictionary from the training data and represent each data point with the dictionary element which is closest to it in the feature space.

Most techniques which perform this projection onto a lower dimensional space are generative: they seek to find a representation of the data which is able to most closely synthesize the input data. Techniques of this type, such as k-means, are not well suited for the task of recognition.

In Chapter 4 we propose and explore a discriminative dictionary which uses information about class labels to inform the dictionary construction. This allows us to create small informative dictionaries without sacrificing performance in the recognition task.

1.1.3 Histogram Construction

In bag of features, histogram construction is straightforward. We create a bin in the histogram for each element in our dictionary, and count the number of occurrences of each dictionary element in the image. This is the representation of each image we compare during classification.

In Chapter 4 we extend the concept of integral images to propose a fast method for the extraction of histograms on arbitrary sub-windows of an image. In Chapter 5, for the task of class segmentation we modify the histogram construction step to include neighborhoods around a local region which are defined by the superpixelization of the image.

1.1.4 Classification

Once we have reduced the image to a histogram, assign a class label to it becomes a machine learning problem. In Chapter 4 we use two of the most popular approaches: nearest neighbor and a support vector machine (SVM). In a k-nearest neighbor classification setting, we measure the distance between our test image histogram and our training image histograms, and assign to it the majority label of the k-nearest neighbors. A support vector machine instead attempts to find a set of separating hyperplanes in a *kernel* space that maximize the margin between data points with different class labels. In our applications, we most often use an exponential χ^2 kernel.

CHAPTER 2

BIBO and Structural Stability for Feature Detection

2.1 Introduction

Visual decision tasks encompass a range of problems, including detection, localization, categorization and recognition of objects in images or video. These are all *classification* problems, where in some cases the class is a singleton (recognition), in other cases it can be quite general depending on functional or semantic properties of objects (categorization). Conceptually, they all require the evaluation and learning of the *likelihood* of the data (images I) given the class label c: $p(I|c)$. Minimizing the conditional risk of classification is equivalent to maximizing the *posterior* $p(c|I)$, which in turn (under equi-probable priors $P(c = 1) = P(c = 0) = 1/2$) is equivalent to maximizing the *likelihood* $p(I|c)$:

$$\boxed{\widehat{c} = \arg \max_{c \in \{0,1\}} p(I|c).} \tag{2.1}$$

So, in a sense, the problem of visual decision-making, including detection, localization, recognition, categorization, is encapsulated in (2.1). That would be easy enough to solve if we could compute the likelihood.

The difficulty in visual decision problems arises from the fact that the image I depends on a number of *nuisance factors* that do not depend on the class, and yet affect the data. These depend on the task, and may include viewpoint, illumination, partial occlusions, quantization etc. If we could, we would base our decision *not* on the data I, but on the hidden variables ξ that comprise the defining characteristics of the *scene* (object, category, location etc.) that depend on the class c, through a Markov chain $c \longrightarrow \xi \longrightarrow I$. This would correspond to a data generation model where by a sample c from $P(c)$ is selected, based on which a sample ξ from $dQ_c \doteq dP(\xi|c)$ is selected, from which a measurement I is finally sampled via an image-formation functional $I = h(\xi)$.

However, because of the nuisances, we have to instead consider a generative model of the form $I = h(\xi, \nu)$, where h is a functional that depends on the imaging device and ν are all the nuisance factors. It is convenient to isolate within the nuisance ν the additive noise component n arising from the compound effects of un-modeled uncertainty and the nuisances that act as a *group* on the scene, g, although we could lump them into the definition of ν. If we model the group and the noise explicitly, we have

$$\boxed{I = h(g\xi, \nu) + n.} \tag{2.2}$$

Using this model, we wish to find groups of nuisances which we can build co-variant detectors for, and try to build them such that they are insensitive to nuisances we cannot invert.

2.1.1 Features

Let a feature be any deterministic function of the data. We call a feature $\phi : \mathcal{I} \longrightarrow \mathbb{R}^K; I \longmapsto \phi(I)$. Note, the decision rule itself, $\widehat{c}(I)$ is a feature. However, it does not just depend on the datum I, but also on the entire training set. Therefore, we reserve the nomenclature "feature" for statistics that depend on the current data, not the training set.

In general, according to the data processing inequality [Sha98] the use of a feature comes with a loss of performance in the decision task. However, there are conditions when optimal performance can still be attained, namely when the feature is a sufficient statistic. Ideally, one would also want the feature to be invariant to the nuisances. One of the many possible ways of designing an invariant feature is to use the data I to "fix" a particular group element $\widehat{g}(I)$, and then "undo" it from the data. So, we define a *(co-variant) feature detector* to be a functional designed to choose a particular group action $\widehat{g}$, from which we can easily design an invariant feature, often referred to as an invariant (feature) descriptor.

Definition 1 (Co-variant detector) *With reference to the model (2.2), we define a* (G-)co-variant detector *to be a functional* $\psi : \mathcal{I} \times G \longrightarrow \mathbb{R}^{dim(G)}$; $(I, g) \mapsto \psi(I, g)$ *such that*

1. *The equation* $\psi(I, g) = 0$ *uniquely determines a group element* $\widehat{g} = \widehat{g}(I)$.
2. *If* $\psi(I, \widehat{g}) = 0$, *then* $\psi(I \circ g, \widehat{g} \circ g) = 0 \ \forall \ g \in G$, i.e., ψ *co-varies with* G.

where by the action $I \circ g$ *we mean the map* $(I, g) = (h(\xi, 0), g) \mapsto h(g\xi, 0) \doteq I \circ g$.

The first condition can be expressed in terms of "transversality" [GP74] of the operator ψ: i.e., it is equivalent to the determinant of the Jacobian of ψ with respect to g being non-singular:

$$\det\left(\frac{\partial \psi}{\partial g}\right) \doteq |\nabla \psi| \neq 0. \tag{2.3}$$

Definition 2 (Canonizability) *We say that the image I is G-canonizable (is canonizable with respect to the group G), and $\widehat{g} \in G$ is the canonical element, if there exists a covariant detector ψ such that $\psi(I, \widehat{g}) = 0$.*

Note that, depending on the functional ψ, the statistic may be local, *i.e. only depend on $I(x)_{|_{x\in\mathcal{B}\subset D}}$, i.e., on a restriction of the image to a subset $\mathcal{B}$ of its domain D. In the latter case we say that I is locally canonizable, or, with an abuse of nomenclature, we say that the* region *$\mathcal{B}$ is canonizable.*

The resulting descriptor is often called a *local invariant feature* The transversality condition (2.3) guarantees that $\widehat{g}$, the canonical element, is an isolated (Morse) critical point [Mil69] of the derivative of the function ψ via the Implicit Function Theorem [GP74]. So a co-variant detector is a statistic (a feature) that "extracts" a group element $\widehat{g}$.

With a co-variant detector we can easily construct an invariant descriptor as follows:

Definition 3 (Canonized descriptor) *For a given co-variant detector ψ that fixes a canonical element $\widehat{g}$ via $\psi(I, \widehat{g}(I)) = 0$ we call the statistic*

$$\boxed{\phi(I) \doteq I \circ \widehat{g}^{-1}(I) \quad | \quad \psi(I, \widehat{g}(I)) = 0.} \tag{2.4}$$

an invariant descriptor.

Theorem 1 (Canonized descriptors are complete features) *Let ψ be a co-variant detector. Then the corresponding canonized descriptor (2.4) is an invariant sufficient statistic.*

Proof: *To show that the descriptor is invariant we must show that $\phi(I\circ g) = \phi(I)$. But $\phi(I \circ g) = (I \circ g) \circ \widehat{g}^{-1}(I \circ g) = I \circ g \circ (\widehat{g}g)^{-1} = I \circ g \circ g^{-1}\widehat{g}^{-1}(I) = I \circ \widehat{g}^{-1}(I)$ To show that it is complete it suffices to show that it spans the orbit space $\mathcal{I}/G$, which is evident from the definition $\phi(I) = I \circ g^{-1}$.*

2.2 When are canonized features optimal?

The use of canonization to design invariant descriptors requires the image to support "reliable" (in the sense of Definition 1) co-variant detection. The challenge in canonization is *not* when the co-variant detector is unreliable, for that implies the image is "insensitive" to the action of G. Instead, the challenge is when the covariant detector reliably detects the *wrong* canonical element $\widehat{g}$, for instance where there are multiple repeated structures that are locally indistinguishable, as often the case in cluttered scenes.

The good news is that, when canonization works, it simplifies visual classification by eliminating the group nuisance without any loss of performance.

Theorem 2 (Invariant classification) *If a complete G-invariant descriptor $\widehat{\xi} = \phi(I)$ can be constructed from the data I, it is possible to construct a classifier based on the class-conditional distribution $dP(\widehat{\xi}|c)$ that attains the same minimum (Bayesian) risk as the original likelihood $p(I|c)$.*

The proof follows from the definitions and Theorem 7.4 on page 269 of [Rob01]. The classifier based on the complete invariant descriptor is called *equi-variant.*

An important caveat is that we have assumed that the non-invertible nuisance is absent, i.e. $\nu = 0$, or that, more generally, the canonization procedure for g is independent of ν, or "commutes" with ν, in a sense that we will make precise in Definition 4. In general, this cannot be done because some nuisances are clearly not invertible (occlusions, quantization, additive noise), and therefore they cannot be canonized. Worse, with or without canonization one can simply not construct a complete invariant to occlusions or to quantization. And even if only one non-invertible nuisance exists, once it is composed with other group nuisances, the entire composition becomes non-invertible.

To deal with the interaction between invertible and non-invertible nuisances, we relax the condition $\nu = 0$ and describe feature detectors that "commute" with ν. We show that the only subgroup of G that has this property is the isometric group of the plane (planar rotations, translations and reflections). Other nuisances, groups or not, have to be dealt with by marginalization or extremization if one wishes to retain optimal performance. This includes the similarity group of rotations translations and scale, that is instead canonized in [Low99], and the affine group, that is instead canonized in [MS04].

In order to simplify the derivation, we introduce the following notation, in part already adopted earlier. If $\widehat{I}$ is the "perfect" image (without nuisances), $\widehat{I} = h(\xi, 0)$, then

$$\widehat{I} \circ g \doteq h(g\xi, 0) \tag{2.5}$$

$$\widehat{I} \circ \nu \doteq h(\xi, \nu). \tag{2.6}$$

The operators $(\cdot \circ g)$ and $(\cdot \circ \nu)$ can also be composed, $\widehat{I} \circ g \circ \nu = h(g\xi, \nu)$ and applied to an arbitrary image; for instance, if $I = h(g\xi, \nu)$, then for any other $\tilde{g}, \tilde{\nu}$ we have

$$I \circ \tilde{g} \circ \tilde{\nu} = h(\tilde{g}g\xi, \tilde{\nu} \oplus \nu) \tag{2.7}$$

where $\oplus$ is a suitable composition operator that depends on the space where the nuisance ν is defined. Note that, in general, the action of the group and the other nuisances do not commute: $I \circ g \circ \nu \neq I \circ \nu \circ g$. When this happens we say that the group commutes with the (non-group) nuisance:

Definition 4 (Commutative nuisance) *A group nuisance $g \in G$* commutes *with a (non-group) nuisance ν if*

$$I \circ g \circ \nu = I \circ \nu \circ g. \tag{2.8}$$

Note that commutativity does not coincide with invertibility: A nuisance can be invertible, and yet not commutative (e.g. the scaling group does not commute with quantization).

For a nuisance to be canonizable (i.e. eliminated via pre-processing, or via a complete invariant feature) it not only has to be a group, but it also has to commute with the other nuisances. In the following theorem we show that the only nuisances that are canonizable are the isometric group of the plane. While it is common, following the literature on scale selection, to canonize it, scale is actually *not* canonizable, so the selection of a single representative scale is not advisable. Instead, a description of a region of an image at all scales should be considered, since scale, in a quantized domain, is a *semi-group*, rather than a group.

Theorem 3 (What to canonize) *The only nuisance that commutes with quantization is the isometric group of the plane, that is the group of rotations, translations and reflections.*

Proof: *We want to characterize the group g such that $I \circ g \circ \nu = I \circ \nu \circ g$ where ν is quantization. For a quantization scale σ, we have the measured intensity (irradiance) at a pixel x_i*

$$I \circ \nu(x_i) \doteq \int_{\mathcal{B}_\sigma(x_i)} I(x)dx = \int \chi_{\mathcal{B}_\sigma(x_i)}(x)I(x)dx = \int \chi_{\mathcal{B}_\sigma(0)}(x - x_i)I(x)dx$$
$$= \int \mathcal{G}(x - x_i;\sigma)I(x)dx \quad (2.9)$$

where $\mathcal{B}_\sigma(x)$ is a ball of radius σ centered at x, χ is a characteristic function that is written more generally as a kernel $\mathcal{G}(x;\sigma)$, allowing the possibility of more general quantization or sampling schemes, including soft binning based on a partition of *unity of D rather than simple functions χ. Now, we have*

$$(I \circ \nu) \circ g(x_i) = \left(\int \mathcal{G}(x - x_i; \sigma) I(x) dx \right) \circ g = \int \mathcal{G}(x - gx_i; \sigma) I(x) dx \quad (2.10)$$

whereas, with a change of variable $x' \doteq gx$, we have

$$(I \circ g) \circ \nu(x_i) = \int \mathcal{G}(x - x_i; \sigma) I(gx) dx = \int \mathcal{G}(g^{-1}(x' - gx_i); \sigma) I(x') |J_g| dx' \quad (2.11)$$

where $|J_g|$ is the determinant of the Jacobian of the group G computed at g, so that the change of measure is $dx' = |J_g| dx$. From this it can be seen that the group nuisance commutes with quantization if and only if

$$\begin{cases} \mathcal{G} = \mathcal{G} \circ g \\ |J_g| = 1. \end{cases} \quad (2.12)$$

That is, the quantization kernel has to be G-invariant, $\mathcal{G}(x; \sigma) = \mathcal{G}(gx; \sigma)$, and the group G has to be an isometry. *The only isometry of the plane is the set of planar rotations and translations (the Special Euclidean group $SE(2)$) and reflections. The set of isometries of the plane is often indicated by $E(2)$.*

Corollary 1 (Do not canonize scale (nor the affine group)) *The affine group does not commute with quantization, and in particular the scaling and skew sub-groups. As a consequence, neither do the more general projective group and the group of general diffeomorphisms of the plane. Therefore, scale should not be (globally) canonized and the scaling sub-group should instead be* sampled.

So, although [Soa09] suggests that invariant sufficient statistics can be devised for general viewpoint changes, this is only theoretically valid in the limit when there is no quantization and the data is available at infinite resolution. In the presence of quantization, canonization of anything more than the isometric group is not advisable.

2.3 Stability of feature detectors

The design of a feature detector consists of canonizing the canonizable nuisances in a way that is the least sensitive (or alternatively, most stable) to the non-invertible ones. We consider two qualitatively different measures of sensitivity.

Definition 5 (BIBO stability) *A G-covariant detector ψ (Definition 1) is* bounded-input bounded-ouput *(BIBO) stable if small perturbations in the nuisance cause small perturbations in the canonical element. More precisely, $\forall\ \epsilon > 0\ \exists\ \delta = \delta(\epsilon)$ such that for any perturbation $\delta\nu$ with $\|\delta\nu\| < \delta$ we have $\|\delta\widehat{g}\| < \epsilon$.*

Note that $\widehat{g}$ is defined implicitly by the functional equation $\psi(I, \widehat{g}(I)) = 0$, and a nuisance perturbation $\delta\nu$ causes an image perturbation $\delta I = \frac{\partial h}{\partial \nu}\delta\nu$. Therefore, we have from the Inverse Function theorem[1] [GP74]

$$\delta\widehat{g} = -|J_{\widehat{g}}|^{-1}\frac{\partial h}{\partial \nu}\delta\nu \doteq K\delta\nu \tag{2.13}$$

where J_g is the Jacobian (2.3) and K is called the *BIBO gain*. As a consequence of the definition, $K < \infty$ is finite. The BIBO gain can be interpreted as the sensitivity of a detector with respect to a nuisance. Most existing feature detector approaches are BIBO stable with respect to simple nuisances. Indeed, we have the following

Theorem 4 (Covariant detectors are BIBO stable) *Any covariant detector is BIBO-stable with respect to noise and quantization.*

[1]One has to exercise some care in defining the proper (Frechèt) derivatives depending on the function space where ψ is defined. The implicit function theorem can be applied to infinite-dimensional spaces so long as they have the structure of a Banach space (Theorem A.58, page 246 of [Kir96]). Images can be approximated arbitrarily well in $L^1(\mathbb{R}^2)$, that is Banach.

Proof: *Noise and quantization are additive, so we have $\frac{\partial h}{\partial \nu}\delta\nu = \delta\nu$, and the gain is just the inverse of the Jacobian determinant, $K = |J_{\widehat{g}}|^{-1}$. Per the definition of co-varianat detector, the Jacobian determinant is non-zero, so the gain is finite.*

BIBO stability is reassuring, and it would seem that a near-zero gain is desirable, because it is "maximally (BIBO)-stable." However, simple inspection of (2.13) shows that $K = 0$ is not possible without knowledge of the "true signal." In particular, this is the case for quantization, when the operator ψ must include spatial averaging with respect to a shift-invariance kernel (low-pass, or anti-aliasing, filter). However, *a non-zero BIBO gain is irrelevant for recognition,* because it corresponds to an additive perturbation of the domain deformation, which is a nuisance to begin with (corresponding to changes of viewpoint [SPV09]). On the other hand, structural instabilities are the plague of feature detectors.

Definition 6 (Structural Stability) *A G-covariant detector $\psi \mid \psi(I, \widehat{g}(I)) = 0$ is Structurally Stable if small perturbations $\delta\nu$ preserve the rank of the Jacobian matrix:*

$$\exists\ \delta > 0 \mid |J_{\widehat{g}}| \neq 0 \Rightarrow |J_{\widehat{g}+\delta\widehat{g}}| \neq 0 \quad \forall\ \delta\nu \mid \|\delta\nu\| \leq \delta \tag{2.14}$$

with $\delta\widehat{g}$ given from (2.13).

In other words, a detector is structurally stable if small perturbations do not cause singularities in canonization. We define the maximum norm of the nuisance that does not cause a catastrophic change [PS78] in the detection mechanism the *structural stability margin.* This can serve as a score to rank features.

Definition 7 (Structural Stability Margin) *We call the largest δ that satisfies equation (2.14) the* structural stability margin*:*

$$\delta^* = \sup \|\delta\nu\| \quad \mid \quad |J_{\widehat{g}+K\delta\nu}| \neq 0 \tag{2.15}$$

A sound feature detector is one that identifies Morse critical points in G that are as far as possible from singularities. Structural instabilities occur where spurious extrema in the detector ψ arise that do not correspond to extrema in the underlying signal.

2.4 Maximally Stable Extremal Regions

The stability margins we have described relate to Maximally Stable Extremal Regions (MSER) [MCU02], where regions are created from the watershed of the intensity image, and "maximally stable" regions are selected based on the variation of their area as the watershed progresses. More formally, in MSER one first computes the connected components of the level sets of the intensity image: $\{R_j^l\}_{j=1}^{N_l} = \{x : I(x) \leq l\}$ where l is in $L = [0, 255]$. At each level, a parent-child relationship is formed between each region R_j^l and the region which subsumes it R_p^{l+1}. An (in)stability score v is formed for each region R_j^l:

$$v(R_j^l) = \frac{|R_p^{l+\Delta}| - |R_j^l|}{|R_j^l|} \tag{2.16}$$

The region is stable when its (in)stability score is less than a threshold v_+. The region is "maximally stable" when it is additionally more stable than its parent or any of its children.

2.4.1 Maximally Structurally Stable Extremal Regions

Intuitively, when $v(R_j^l)$ is small, it means that a region is BIBO stable with $K \propto v$ for a $\delta \propto \Delta$. While this is useful, $v(R_j^l)$ is related to K, not to δ. In other words, all regions which have $v(R_j^l) < v_+$ are measured to be equally stable in terms of δ. We define a new stability measure $q(R_j^l)$ which is correlated to the structural stability margin defined in (2.14).

Since we know that what we seek is a score related to δ, and we have observed that $\delta \propto \Delta$, we can define $q(R_j^l)$ to be the largest Δ which satisfies $v(R_j^l) < v_+$:

$$q(R_j^l) = \underset{\Delta}{argmax}\, v(R_j^l) \text{ subject to } v(R_j^l) < v_+ \tag{2.17}$$

This can be computed efficiently in the same union-find algorithm used for standard MSER. Once we have detected an extremal region, we continue to check all the ancestors of the region until the region grows too large ($v(R_j^l) \geq v_+$). In order to maintain the "maximal" part of the algorithm, we still keep the region which has the largest $q(R_j^l)$ of its children or parent. We call these regions Maximally Structurally Stable Extremal Regions (MSSER).

2.5 The segmentation tree

Rather than testing for canonizability (as done customarily in feature detection [Low99, BTG06]) one can test for stationarity (as done customarily in segmentation) and then construct features from the segmentation tree. The caveat is that, because of the interplay of the scale group with quantization, and of the translation group with occlusion, *no single segmentation* can be used as a viable canonization procedure, and instead the entire *segmentation tree* must be considered.

The starting point for this approach to canonization is an approximation model of the image as set of simple functions. These are constant functions on a compact domain, whose universal approximation properties in several measures are guaranteed by Weierstrass' theorem [Rud73]. In particular, let $\sigma > 0$ be a given "scale." Then, given an image, we can find a partition of the domain D

and constant values such that a combination of simple functions approximates the original image (or any statistic computed on it) to within σ in each region. Specifically, for a given $I \in \mathcal{I}$ and σ, we assume there is N and constants $\{\alpha_1, \dots, \alpha_N\}$ and a partition of the domain $\{S_1, \dots, S_N\}$ such that

$$|I(x) - \alpha_j| \leq \sigma \ \forall \ x \in S_j; \quad S_i \cap S_j = \delta_{ij}; \ \cup_{j=1}^{N} S_j = D, \tag{2.18}$$

where δ_{ij} is Kronecker's Delta. Then, if we define the simple functions as characteristic functions of S_j

$$\chi_{S_j}(x) = \begin{cases} 1, \ \forall \ x \in S_j \\ 0 \ otherwise \end{cases} \tag{2.19}$$

we can approximate the image with

$$\widehat{I}(x) = \sum_{j=1}^{N} \chi_{S_j}(x)\alpha_j. \tag{2.20}$$

This is true for scalar-valued images, but a similar construction can be followed to partition the domain into regions (often called *superpixels* [RM03]), based on σ-constancy of any other statistic, such as color or any other higher-dimensional feature ϕ. So, a superpixelization algorithm can be thought of as a quantizer, that is an operator that takes an image I and a parameter σ and returns a family of domain partitions $N = N(\sigma), \{\widehat{S}_j\}_{j=1}^{N}$ with

$$\{\widehat{S}_j\}_{j=1}^{N} = \phi_\sigma(I); \quad \widehat{\alpha}_j = \frac{1}{|\widehat{S}_j|} \int I(x)dx \tag{2.21}$$

where $|S_j|$ is the area of S_j. We can now use this functional to determine a co-variant detector $\psi(I, g)$. For the case of translation, $g = T$, we define (multiple) canonical elements T_j to be the centroids of the regions S_j, $\widehat{T}_j = \frac{1}{|S_j|} \int_{S_j} x dx$. Making the dependency on the image explicit, we have

$$\widehat{T}(I, \sigma) = \frac{\int_{\phi_\sigma(I)} x dx}{\int_{\phi_\sigma(I)} dx} \tag{2.22}$$

to which there corresponds the canonizing functional

$$\psi(I, \{T, \sigma\}) = \int_{\phi_\sigma(I)} x dx - T \int_{\phi_\sigma(I)} dx. \tag{2.23}$$

It can be easily verified that this functional is co-variant. For the case of translation (fixing σ), $g = T$, we have that, for $\widehat{g}$ that solves $\psi(I, \widehat{g}) = 0$, and for any g

$$\begin{aligned}\psi(I \circ g, \widehat{g} \circ g) &= \int_{\phi(I \circ g)} x dx - g\widehat{g} \int_{\phi(I \circ g)} dx \\ &= \int_{g\phi(I)} x dx - g\widehat{g} \int_{\phi(I)} dx \\ &= \int_{\phi(I)} gx' dx' - g\widehat{g} \int_{\phi(I)} dx \\ &= g \left(\int_{\phi(I)} x' dx' - \widehat{g} \int_{\phi(I)} dx \right) \\ &= g\psi(I, \widehat{g}) = 0 \end{aligned} \tag{2.24}$$

where we have used the fact that the group g is isometric, so $dx = dx'$. One may also believe that this functional yields isolated extrema, based on the fact that

$$|\nabla \psi| = |\frac{\partial \psi}{\partial T}| = \int_{\psi_\sigma(I)} dx > 0. \tag{2.25}$$

However, this result, as well as (2.24), is misleading because it assumes that the superpixelization $\phi_\sigma(I)$ is independent of (small variations in) T. More precisely, in order for translation to be canonizable, the canonization process *has to commute with quantization.* If we assume the the underlying "ideal image" $I(x)$, $x \in D$ is continuous, then the "discrete" (quantized) image $\widehat{I}(x_i) = \int_{\mathcal{B}_\epsilon(x_i)} I(x)dx/\epsilon, x_i \in \Lambda$ defined on the lattice Λ, is related to it via the mean-value theorem, that guarantees the existence, for each x_i, of a translation δ_i such that

$$\widehat{I}(x_i) \doteq \frac{1}{|\mathcal{B}_\epsilon|} \int_{\mathcal{B}_\epsilon(x_i)} I(x)dx = I(x_i + \delta_i) = I(x_i) + n_i \doteq I \circ \nu \tag{2.26}$$

where ν denotes the quantization nuisance. For the canonization process to be viable we must have

$$\phi_\sigma(I) = \phi_\sigma(I \circ \nu) \tag{2.27}$$

which is clearly not the case in general for a superpixelization algorithm. In fact, if we apply small perturbations to the levels n_i, from (2.26) we get small perturbations in the location of the boundaries ∂S_j, and in the location of their centroid, $T_j \rightarrow T_j + \delta_i$. Since $n_i = \nabla I(x_i)\delta_i$, we see that for a superpixelization procedure ϕ_σ to provide a viable canonization mechanism, it has to place the boundaries in such a way that $\frac{\partial I}{\partial x}$ is negligible within each region, and as large as possible at region boundaries. We will see this spelled out more in detail shortly. The sensitivity of the boundary location as a function of a perturbation can be phrased in terms of BIBO stability, introduced in Definition 5. Specifically, if ϕ_σ is a quantization or superpixelization operator acting on an ϵ-quantized image (2.26), and $\{S_j\}_{j=1}^N \doteq \phi_\sigma(I)$ and $\{\tilde{S}_j\}_{j=1}^N \doteq \phi_\sigma(I+n)$ the corresponding partitions, then ϕ_σ is BIBO stable if, according to (2.13),

$$\|n\| \leq \sigma \Rightarrow |\tilde{S}_j - S_j| \leq \epsilon \tag{2.28}$$

where $|S_1 - S_2|$ denotes the area of the set-symmetric difference of the two sets S_1 and S_2.

More in general, consider a perturbation of the image $\tilde{I}(x) = I(x) + n(x)$, with $n(x)$ assumed to be small in some norm. Then after quantization of the domain, using the Mean Value Theorem, we have

$$\tilde{I}(x_i) = \int_{\mathcal{B}_\epsilon(x_i)} \tilde{I}(x)dx = \int_{\mathcal{B}_\epsilon(x_i)} I(x)dx + \int_{\mathcal{B}_\epsilon(x_i)} \nabla I(x)dx\delta_i \tag{2.29}$$

where we have assumed that $\delta(x)$ is constant within each quantization region $\mathcal{B}_\epsilon(x_i)$. Now, if we approximate the piecewise constant function in each $\mathcal{B}_\epsilon(x_i)$

into a piecewise constant function in the partition $\{S_j\}_{j=1}^N$, for a given N, we have

$$\sum_{j=1}^{N} \chi_{\tilde{S}_j}(x_i)\tilde{\alpha}_j = \sum_{j=1}^{N} \chi_{S_j}(x_i)\alpha_j + \sum_{j=1}^{N} \chi_{S_j}(x_i) \int_{\mathcal{B}_\epsilon(x_i)} \nabla I(x) dx \delta(x_i) \tag{2.30}$$

from which we can obtain, defining $\delta S_j = \tilde{S}_j - S_j$ the set-symmetric difference between corresponding regions, and measuring the area in each region,

$$\sum_{j=1}^{N} \sum_{x_i \in S_j} \chi_{\delta S_j}(x_i)\alpha_j = \sum_{j=1}^{N} \int_{S_j} \nabla I(x) dx \delta_j \tag{2.31}$$

where we have now assumed that $\delta(x_i)$ is constant within each $x_i \in S_j$, and we have called that constant δ_j. So, we have that for a partitioning to be BIBO Stable, we must have

$$\sum_{j=1}^{N} |\delta S_j| \leq \sum_{j=1}^{N} \int_{S_j} \|\nabla I(x)\| dx \tag{2.32}$$

which is guaranteed so long as the image is smooth and the magnitude of the gradient is low within each region S_j (but it can be discontinuous across the boundary ∂S_j). Indeed, any reasonable segmentation procedure would attempt, *for any given (fixed) N*, to place the discontinuities of the (true underlying) image I at the boundaries ∂S_j, therefore guaranteeing stability of the boundaries with respect to small perturbations of the image per the argument above. In particular, for $N = 2$, there are algorithms that guarantee a globally optimal solution [CV01] that is, by construction, the most stable with respect to small perturbations of the image. These results are summarized into the following statement.

Theorem 5 (BIBO Stability of the segmentation tree) *A quantization / superpixelization operator, acting on a piecewise smooth underlying field and subject to additive noise, is BIBO stable at a fixed scale for a fixed tolerance ϵ (or complexity level N) if and only if it places the boundary of the quantized regions/superpixels at the discontinuities of the underlying field.*

However, our concern is not just stability with respect to the partition $\{S_j\}_{j=1}^N$ *for a fixed* N, but also stability with respect to *structural perturbations* that change the cardinality of the partition (a phenomenon linked to scale since $N = N(\sigma)$). The superpixelization procedure should be designed to be stable with respect to N, which is not a test we can write in terms of differential operations on the image. However, one can use a greedy method to construct a tree which is designed to be stable with respect to both regular (bounded) and structural perturbations.

First, form a set of superpixels $\{S_j\}_{j=1}^N$ at scale σ which are, by construction, BIBO stable for a given σ. These are the leaves of the tree. At each iteration, add a new parent to the tree which represents the merging of two nodes which are minimally different (e.g. $\alpha_i \approx \alpha_j$). The nodes remaining after any merge form a partition of the domain which is itself BIBO stable, for a different σ. The "cost" of merging two regions at each iteration is a measure of its stability to structural perturbations: since the minimal cost merge is always chosen, high cost merges represent regions which were structurally stable.

So we have shown that canonizing translation via the centroid of superpixels is automatically viable, per (2.24) and (2.25), but only so long as the superpixelization is stable with respect to additive noise, for instance generated by quantization mechanisms.

Theorem 6 (Canonization via superpixels) *Let $\phi_\sigma(I) = \{S_j\}_{j=1}^N$ be a partition of the domain into superpixels. Then (2.23) is a (local, translation) covariant detector so long as it is BIBO stable.*

That (2.23) is covariant follows from (2.24), provided that (2.13) is satisfied.

2.6 Repeatability of nodes in segmentation trees

We now wish to show empirically that under large additive noise, regions which we deem structurally stable still persist. To this end, we perform an experiment on the MSRC dataset [SWR06]. MSRC contains 592 images of scenes which include objects from 21 categories. We take each image and segment it with one of two segmentation methods (described in Section 2.6.1) before and after the addition of uniformly distributed noise (between -20 and 20 on a 0-255 scale).

Nodes which are repeated are those whose best match intersection-union score is greater than 0.6. We measure the percentage of nodes per image which are repeated according to this criteria, and present the average repeatability for the leaves of the tree, the full tree, and only structurally stable nodes in Table 2.1. The methods we use to identify structurally stable nodes are described in Section 2.6.2.

2.6.1 Segmentation methods

We chose two BIBO stable superpixelization methods to form the leaves of our tree and form the internal nodes with the iterative merge procedure described in section 2.5. The first is **mean shift** [CM02]. Since we wish to obtain superpixels rather than large regions, we set the spatial and range bandwidth parameters to be small (both are set to 5), and the minimum region area to 10 pixels. Our second method of forming superpixels is **Laplacian of Gaussian (LoG) watershed**. We compute the response of an image to a LoG filter with standard deviation of 1. We use the watershed transform on this response to form our initial regions.

Tree type	Leaves	Full tree	Cost-iteration	Delta-area
Mean shift	9.0%	8.7%	23.0%	60.8%
LoG watershed	52.4%	34.9%	57.5%	80.7%

Table 2.1: **Stability on MSRC.** Average percentage of repeatable regions under uniform noise on approximately 600 images from MSRC.

2.6.2 Stability measures

Cost-iteration. Here, stability is measured by the cost of a merge times the number of iterations between when the child and parent node were formed. Nodes which have a high cost and remained unchanged for many iterations are deemed stable. This is related to the life-span measure discussed in the persistent topology literature [ELZ02].

Delta-area. Stability is measured by the cost to merge a node with its parent. We iteratively select the most stable node, excluding all nodes which are ancestors or descendants of the node which have less than Δ difference in area. In our experiments we set Δ to 1.2, or in other words we select the most stable nodes which differ in area by at least 20%.

2.6.3 Discussion

Results of the experiments may be found in Table 2.1. We can see that we have introduced a sufficiently large perturbation to cause repeatability problems in a single segmentation (the leaves). Further, the percentage of repeated nodes in the full tree is worse than the leaves alone. Mean shift trees seem to be less stable than trees constructed from LoG regions, leading us to believe that mean shift is less BIBO stable than LoG. For both segmentation methods, selecting stable nodes using either of our criteria always provides improvement in repeatability. The delta-area method is always better, and the best overall repeatability is found with LoG superpixels and delta-area stability, so we adopt this combination in our categorization experiments which follow.

2.7 Categorization with superpixel trees

In this section, we evaluate the effectiveness of the superpixel tree as a detector in a full recognition pipeline for Caltech-101. We compare LoG superpixel trees with and without stable node selection, the SIFT detector, densely selected features, and randomly selected regions using the recognition framework of Vedaldi and Fulkerson [VF10]. Once features are detected with the detector we are evaluating, descriptors are constructed using the SIFT implementation of VLFeat. Note that the orientation is fixed to 0, since we assume all images were taken with an upright camera. The features are quantized into a k-means dictionary with $k = 600$. These quantized features are aggregated into a 4x4 spatial histogram which forms the final representation of the image. Histograms are classified using a χ^2 SVM (transformed to a linear one as in [VZ10]). For each category, 15 random training images and 15 random testing images are selected.

2.7.1 Superpixel tree detector

We construct a superpixel tree using LoG seeded watershed regions as described in Section 2.6.1. Each node in the tree defines a region, so the set of all nodes in the tree may be viewed as a detector. However, before we can use the regions in this way, there are a two issues to address.

First, we need to define the size of the region we intend to describe. We fit a circle to the pixels which form the region, and enlarge this circle by 3 times (in the SIFT setting, this is called the "magnification" of the descriptor). But, picking one size for each region is another way of saying that we canonize scale, which is something we should not do. So, we also consider enlarging the enclosing circle by a number of different factors and describing each separately. In Table 2.2 we show results using both single and multiple scales for each region.

Second, we need to decide if we should describe all regions in the tree, or just the ones we deem to be stable. We select stable regions using the delta-area measure of structural stability described in 2.6.2, and the results are shown in Table 2.2 for both single and multiple scales.

2.7.2 Random features

In Table 2.2, we show that the number of features extracted tends to have a strong correlation to the resulting accuracy. When we extract more features, seemingly regardless of where they are located, there is an improvement in the precision of the resulting classifier. To investigate this effect in the context of our detectors, we include results for a random feature detector. The random detector randomly samples a number of positions and scales which form the detected features. We show results for 200, 1000, 10000, and 40000 features per image both to provide a baseline for comparison at the amounts of features which we encounter with our other detectors, and to show that the improvement does not continue indefinitely (40000 features produce similar results to 10000).

Method	Accuracy	Features per image
SIFT detector	33.14%	220
Delta-area stable nodes	37.98%	197
Delta-area stable nodes (Multi-scale)	48.89%	1577
Superpixel tree	53.07%	1331
Superpixel tree (Multi-scale)	58.04%	10648
Superpixel tree (PHOW)	60.65%	9850
PHOW	62.68%	9285
200 random features	25.95%	200
1,000 random features	35.39%	1000
10,000 random features	42.03%	10000
40,000 random features	42.61%	40000

Table 2.2: **Comparison of detectors for 15 training images on Caltech-101**. For a small number of features, a detector which selects stable segments is better than the SIFT detector. However, if we allow the number of features to grow significantly (from 200 to 10,000), dense features (PHOW) perform best.

2.7.3 Discussion

In Table 2.2 we evaluate the detectors described above on Caltech-101. When we only want to extract a small number of features, our stable segment detector outperforms the SIFT detector by approximately 5% (33% v.s. 38%). If there is no constraint on the number of features allowed, we can improve further both by using the whole tree (53%) or multi-scale stable nodes (49%). Using the whole tree and features extracted at multiple scales, we have a comparable number of features and performance (58%) to the best performing cases which both use PHOW. The PHOW (63%) features are extracted on multiple predefined scales on a 5x5 grid of the image (after resizing it to be at most 480 pixels tall). Superpixel tree PHOW regions (61%) also extract features on a grid, but scale the size of the grid and the scale of the features depending on the size and scale of the region that is being described. All detectors perform better than random features with approximately the same number of features per image.

2.8 Conclusion

We have proposed a model for understanding sensitivity to group nuisances in feature detection as it applies to recognition tasks. We have shown that one should only canonize the isometric group of the plane (rotations, translations, and reflection). Two stability criteria for detectors were proposed: the bounded-input bounded output (BIBO) gain, and structural stability. We showed that Maximally Stable Extremal Regions are BIBO stable, and redefined the stability score to create Maximally Structurally Stable Extremal Regions, which we demonstrated correlated to the empirical stability under perturbations due to noise. Finally, we proposed a new feature detector based on a tree of segmentations, and showed that this detector is both stable and better than the SIFT detector for categorization tasks.

CHAPTER 3

Categorization with Segmentation

3.1 Introduction

In image categorization, the methods which are currently among the most successful are largely based on the concept of matching unordered collections (bags) of features. Typically, features are extracted from training images and passed to a machine learning component that tries to learn which features are important in discriminating between classes. Approaches that fall into this category include [GD06] and [ML06]. Mutch and Lowe [ML06] build a multi-resolution patch based feature representation and train a support vector machine to learn to discriminate based on the maximal responses of their feature patches. Grauman and Darrell [GD06] also use a support vector machine, proposing a new kernel which they use for fast, robust approximate matching between SIFT [Low04] descriptors.

But, no matter what machinery is used for learning, bags of features have a significant limitation. By design, they completely discard spatial relationships between local features even though we know that spatial cues play an important role in our perception of objects. To address this, the object category recognition community has begun to incorporate spatial information into their techniques. On the Caltech-101 dataset [FFP04], all of the best performing algorithms depend heavily on spatial information. Zhang *et al.* [ZBM06] construct a geometric blur descriptor which computes a representation of local context sampled along edges and combines it with the response of a filter bank as a kernel for their SVM-KNN classifier. Lazebnik *et al.* [LSP06], inspired by the multi-scale matching of Grauman and Darrell [GD06], construct histograms in a pyramid of spatial bins, in effect computing a bag of features representation on subdivided windows of each image. Our own Caltech-101 classifier (built into VLFeat [VF10]) uses spatial binning of multi-scale dense SIFT features.

Another limitation of image categorization is that there cannot be excessive clutter in the scene. That is, the object of interest is assumed to be prominent and the rest of the image is assumed to be simple and homogeneous. Most schemes are at least somewhat robust to nuisances such as those described in Chapter 2, but none are immune. In fact, there is no way to decide if an image has too many nuisances without applying the algorithm to the image and observing the result.

We contend that even on datasets that have limited clutter and canonical object poses such as Caltech-101 [FFP04], nuisance features are still a significant problem. To demonstrate this, we adopt the spatial pyramid matching method of Lazebnik *et al.* [LSP06] and alter it to take advantage of segmentations provided with the Caltech-101 dataset, as well as two segmentations of our own.

We show that if we have the correct foreground/background segmentation, the results that can be obtained are significantly improved. Unfortunately, not just any segmentation will do. We will detail our experiences with two other segmentations.

3.2 Spatial pyramid matching

Here we describe our implementation of spatial pyramid matching [LSP06], and in doing so provide both a reminder of how spatial pyramid matching works and our interpretation of the philosophy behind it.

In spatial pyramid matching, the goal is to match images by aggregating information about the locations of quantized local features. Rather than focus on how to match the features, this method focuses on how to collect spatial information about the features in such a way that it can be easily and robustly used in the image matching process. Spatial pyramid matching consists of four main steps: features are extracted, a dictionary is constructed, spatial histograms are created from the dictionary, and finally a support vector machine with a kernel tailored to the spatial histograms is applied.

3.2.1 Feature extraction

The first step of spatial pyramid matching is to extract the features that will form our image representation. For each image, we extract SIFT descriptors [Low04] on a regular 8×8 pixel grid at a fixed scale and fixed orientation. We fix the support region of the SIFT descriptor at 16×16 pixels and the orientation at 0 degrees. We experimented with allowing SIFT to add rotation invariance by calculating the dominant angle of the gradient of each patch and aligning the descriptor to that, but this only hurt our performance. Some of our experiments use a more dense sampling (a 4×4 pixel grid), but the scale of the patches remains the same. SIFT descriptors with low norm, which correspond to patches with low contrast, are discarded. Since we are extracting fixed scale and fixed orientation patches, we can use the fast dense SIFT method described in Chapter 4.

3.2.2 Dictionary construction

After the features have been extracted, a small percentage of them (1-2%) are sampled from the training set and used to construct a Gaussian mixture model with M centers for feature quantization. As in [LSP06], we use $M = 200$. This dictionary is subsequently used to quantize all of the features in the training and testing data. Here we could also use k-means or our compressed dictionary described in Chapter 4, but we focus on reproducing the results of the paper directly.

As with any bag of words model, by adopting this scheme we are implicitly assuming that any quantized descriptor matches all other descriptors which are quantized to the same bin, regardless of their distance in the feature space. We will revisit this later, when we discuss splitting SIFT descriptors.

3.2.3 Spatial histograms

Once we have quantized the features, we construct the spatial pyramid. At the bottom level of the pyramid ($L = 0$), we simply have a histogram of size M containing the quantized feature occurrences over the entire image. At each higher level, each image is partitioned into four regions, with each region receiving a histogram of size M. In all of our experiments, we use $L = 2$.

We normalize the spatial histograms by dividing by the total number of features in each image. This accounts for variability in the number of features across all images.

Spatial histograms may be thought of as a coarse encoding of the spatial information contained in the features. Within the confines of a spatial bin, all of the feature locations could be permuted, and from the perspective of the representation, everything would be equivalent. In practice, this allows for variability in the appearance of a rigid object and minor articulations in an object whose pose has changed.

3.2.4 Spatial pyramid matching with a support vector machine

At the heart of spatial pyramid matching scheme is a support vector machine that uses a modified intersection kernel to compare pairs of spatial histograms. The final kernel $K(X,Y)$ is the weighted sum of the intersection of the spatial histograms X and Y. The weights are inversely proportional to the level l of the histogram. Matches which occur only in larger bins are less discriminative and are assigned less weight. The histogram intersection between X and Y at level l is denoted by I and is:

$$I(X^l, Y^l) = \sum_{i=1}^{D} min(X^l(i), Y^l(i)) \tag{3.1}$$

where D is the number of histograms at a given level. The pyramid match kernel $\kappa^L(X,Y)$ is simply the weighted sum of the histogram intersections at each level (which are denoted below with the shorthand I^L):

$$\kappa^L(X,Y) = \frac{1}{2^L} I^0 + \sum_{l=1}^{L} \frac{1}{2^{L-l+1}} I^l \tag{3.2}$$

Finally, in order to arrive at a kernel for spatial pyramid matching, we simply sum the results of the pyramid match kernel over all of the elements of the dictionary M:

$$K^L(X,Y) = \sum_{m=1}^{M} \kappa^L(X_m, Y_m) \tag{3.3}$$

Interested readers may refer to [GD06] for more details about this process.

In our implementation, we train a support vector machine for each category versus the rest using $K^L(X, Y)$ as the kernel function. During test time we try all classifiers and give the image of the label of the support vector machine which results in the largest positive margin.

3.3 Splitting SIFT descriptors

In our experiments, we evaluate splitting SIFT descriptors along region boundaries. To understand why we might want to do this, consider how errors in descriptor quantization might occur.

We want the descriptor from one part of an object to always be quantized to the same dictionary element (since matching is done by checking the identity of the label). But, when the region described crosses an object boundary, some portion of the descriptor will describe the object and the rest will describe the background. If the background changes, the descriptor may not be quantized to the same dictionary element, even if the object appearance is identical. Worse, it may form a spurious match with another descriptor, compounding the error. A SIFT descriptor which crosses an image boundary is very likely to have been computed on an occlusion boundary or other unstable area. By partitioning the descriptor along this boundary, we will be less likely to combine information from two distinct objects into one descriptor.

In order to partition the descriptor, we modify the SIFT algorithm as follows. First, in addition to accepting a source image to generate SIFT descriptors, we take as input a label image which contains pixels labeled by which region they belong to in the segmentation (see Figure 3.4 for a few examples). Next, after the

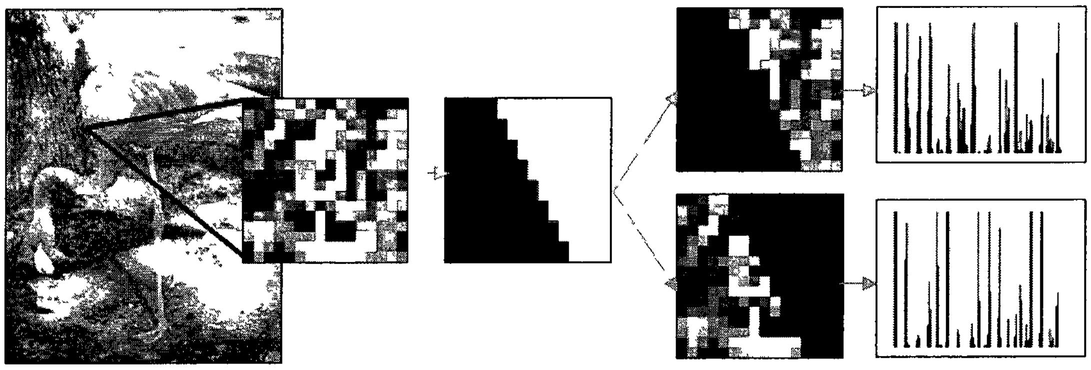


Figure 3.1: **Extracting and splitting a SIFT descriptor given a segmentation.** From the left: the original image, the gradient of the extracted support region, the segmentation information (a mask image), the two gradient regions after they have been split, and last, the descriptors. The contrast of the gradient has been significantly boosted here so that the partitioning process may be more easily visualized.

gradient has been computed for our support region, we initialize a descriptor for every distinct label value found in the support region. For each labeled descriptor, we set the gradient image to zero on all pixels where the label is not equal to our desired value and then compute the SIFT descriptor normally. We still discard low contrast descriptors, so it is possible for this operation to result in fewer descriptors than there are label values in a support region. See Figure 3.1 for an illustration of this process when there are two labels present.

3.4 Experiments

In this section, we report results on the Caltech-101 object category recognition dataset. Caltech-101 contains medium resolution images of 102 object categories

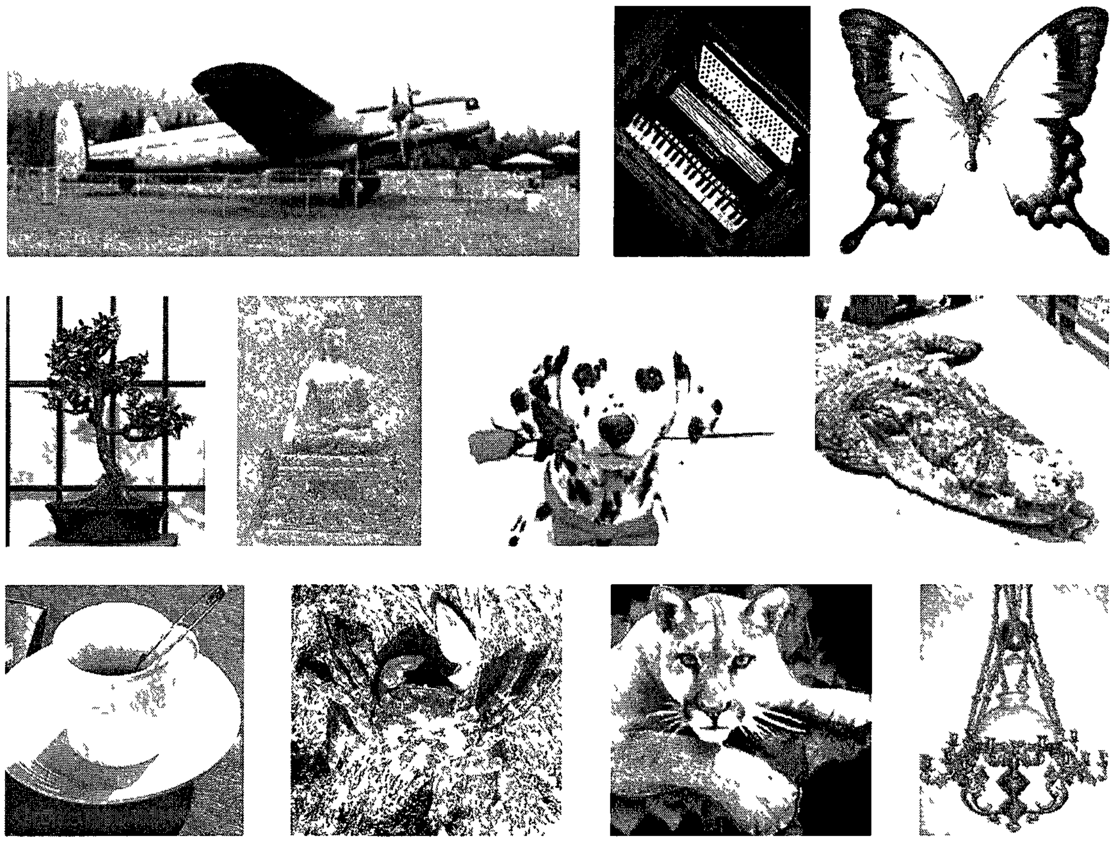


Figure 3.2: **Sample images from Caltech-101.** Categories from the top left: airplane, accordion, butterfly, bonsai, buddha, dalmatian, crocodile head, cup, crab, cougar face, chandelier.

(101 object categories and 1 background category) obtained via Google image search. The objects in this dataset have generally been centered in the images, aligned for pose, and are approximately the same size in the image. Some examples may be found in Figure 3.2.

We first show the results of our implementation of spatial pyramid matching without modifications. Next, we show how these results can be improved significantly by adding a ground truth segmentation, and discuss where this improvement comes from. Finally, we describe the results of splitting descriptors with two different segmentation algorithms: normalized cuts [SM00] and active contours without edges [CV01].

For all of the experiments in this section, we adopt the testing method of Grauman and Darrell [GD06]: We first select N training images at random from each category, then we take the rest of the images to be the testing category. In addition, we normalize the results per category and take the average recognition rate. This way we are not biased by some of the categories in Caltech-101 which are easier but contain a disproportionate number of testing images.

3.4.1 Baseline: Spatial pyramid matching

In order to verify our implementation of [LSP06] and provide a performance baseline for our later experiments, we evaluated our spatial pyramid matching implementation on Caltech-101. In addition to varying the training images, we tried two spatial resolutions for our sampling grid: 4×4 pixels and 8×8 pixels. We found that as we increased the resolution of our sampling grid, our performance increased.

When our features are sampled on a 4×4 grid, our results (62.74% for 30 training images) are comparable to those found in the work of Lazebnik *et al.*

[LSP06]. Having verified this, we now evaluate other schemes.

3.4.2 Ground truth segmentations

The creators of the Caltech-101 dataset provides ground truth annotations for their images [FFP04]. We take these annotations and convert them to mask images which correspond to a perfect segmentation of the object and the background for each image. See, for example, Figure 3.4.

With these segmentations we generate SIFT descriptors for the portion of the image corresponding to the foreground. When the descriptor contained both foreground and background, we split the descriptor as described above and keep only the foreground part. Additionally, we rescaled the spatial bins to a bounding box defined by the foreground segmentation. This corresponds to removing all of the clutter (background) and scaling the object so that it takes up the full image.

With these simple modifications, our performance jumps dramatically to nearly 76% for 30 training images. (see Figures 3.3 and 3.5). We found our results for this experiment were best on an 8×8 spatial grid of extracted features, the 4×4 grid decreased performance slightly and required more computation time.

These results indicate that the amount of clutter and scale variability in Caltech-101 is not negligible. The performance increase of more than 10% suggests that the clutter in the images was significantly reducing the accuracy of the spatial pyramid matching technique.

3.4.3 Normalized cuts

Our next experiments involved substituting an automatic segmentation method for the ground truth. We first chose normalized cuts [SM00]. We fixed the

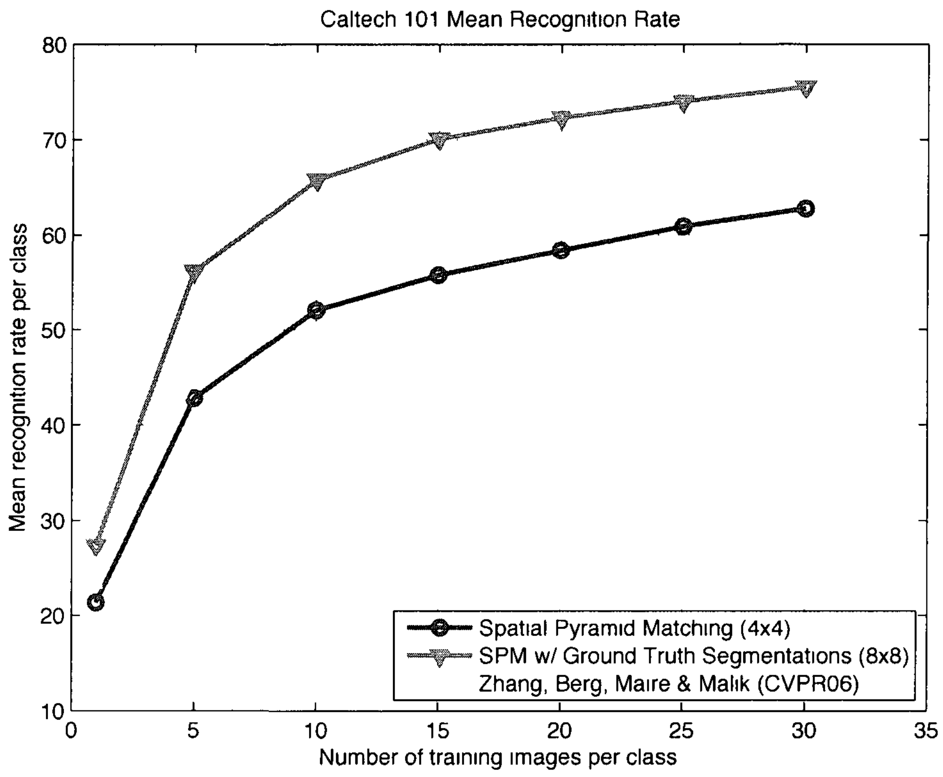
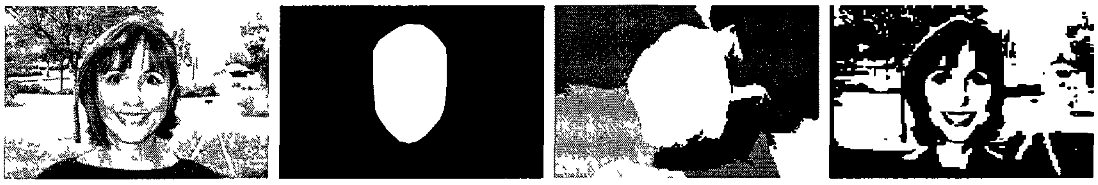


Figure 3.3: **Accuracy of our spatial pyramid matching implementation, as well as the results when ground truth segmentations are incorporated.** The results of Zhang, Berg, Maire & Malik [ZBM06] are also included for reference. Our results were averaged over 5 runs. This figure is best viewed in color.



(a) Original image (b) Ground truth annotation (c) Normalized cuts - 10 regions (d) Active contours without edges

Figure 3.4: **Example segmentations**. (a) is the original image, (b) is the provided annotation, (c) is the result of a normalized cuts segmentation and (d) corresponds to segmentation by active contours without edges.

number of training images at 30 for this experiment and split descriptors along the boundaries between segmented regions.

Here, we cannot filter out the background descriptors, or rescale the window to fit the foreground class because we do not know if a region belongs to foreground or background. The only possible source for a performance gain would be from the split descriptors reducing quantization error. For 30 training images with 10 regions per image, we achieve a rate of 62.38%, or in other words, no change from the baseline.

We believe the main reason for this is that our spatial histograms are normalized by the number of features. As more descriptors are added, peaks in the histogram which once corresponded to strong signals of a particular class are diminished. The extra discriminative power of the split SIFT descriptors compensates for this loss, but does not improve things further.

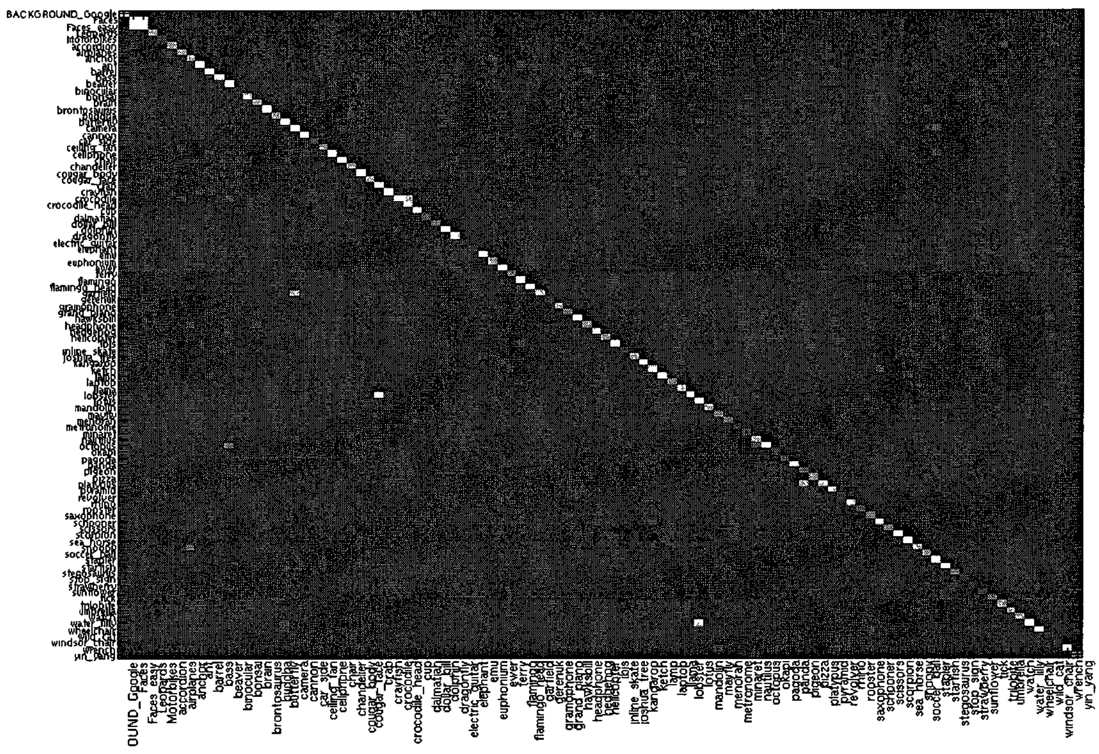


Figure 3.5: **Confusion matrix for the spatial pyramid match kernel with ground truth segmentations and 30 training images per class.** This figure is best viewed in color. The most confused pairs were Faces with Faces_easy. This is intuitive, because the Faces_easy images are just cropped versions of the Faces images. The next 3 top confused pairs were (water_lilly, lotus), (lobster, crab), (crocodile, crocodile_head).

3.4.4 Active contours without edges

After experimenting with normalized cuts, we suspected that there was a possibility that the relatively coarse segmentations it produced (see Figure 3.4 for comparative examples) were preventing us from splitting enough descriptors to noticeably boost our discriminative power. So, we ran an experiment using the segmentation method of Chan *et al.* [CV01]. The resulting segmentations are even less capable of being separated into foreground and background, but have the desirable property that nearly all of the major boundaries in the image translate to boundaries in the segmentation.

We again use 30 training images, and again achieve approximately the same rate (62.39%). Finally, we tried using both the split descriptors as well as the original descriptor before splitting, but this decreased our performance further (61.45%) which supports our hypothesis that more descriptors have a detrimental effect by "polluting" the histogram with potentially incorrect matches.

3.5 Discussion

We have demonstrated via the ground truth segmentations included in the Caltech-101 dataset that we can increase our classification accuracy provided that we have a reasonably consistent and good segmentation of the image into foreground and background regions. Our results suggest that the majority of the improvement comes from locating the spatial bins on a bounding box of the object and ignoring features that lie on the background. This means that knowing *where* an object is, even in relatively uncluttered images, can improve our ability to discern *what* the object is.

Recent work in categorization supports this conclusion. In the pair of companion papers by Li *et al.* [LCS10] and Carreira *et al.* [CS10], the authors propose an object recognition system which is guided by the generation of figure-ground hypotheses obtained using a constrained parametric min-cuts segmentation algorithm. They generate multiple segmentations into foreground and background and evaluate each in turn, choosing the one which is most likely.

This philosophy will be further explored in the next chapters, where we will localize target object classes with informative dictionaries (Chapter 4) and use superpixels and a conditional random field to label multi-class segmentations (Chapter 5).

CHAPTER 4

Discriminative Dictionaries and Pixel Categorization

4.1 Introduction

Bag of features methods have enjoyed great popularity in object categorization, owing their success to their simplicity and to surprisingly good performance compared to more sophisticated models and algorithms. Unfortunately, such methods can answer whether an image contains an object of a certain category but do not offer much insight as to where that object might be within the image. In other words, because the representation discards spatial information, bag of features methods cannot be used for localization directly.

That is, unless one could devise an object categorization method efficient enough to test at a window centered on each pixel of an image. In that case, one would be able to exploit the co-occurrence of features within a local region and localize the object, pixel by pixel. However, with many features detected in each image and quantized into thousands or tens of thousands of "words," this does not appear to be a viable proposition, especially in light of recent results that advocate using very large visual dictionaries [NS06, PCI07, TS07].

But what if we could reduce the size of a dictionary from tens of thousands of words to a few hundred and maintain improved localization? After all, dic-

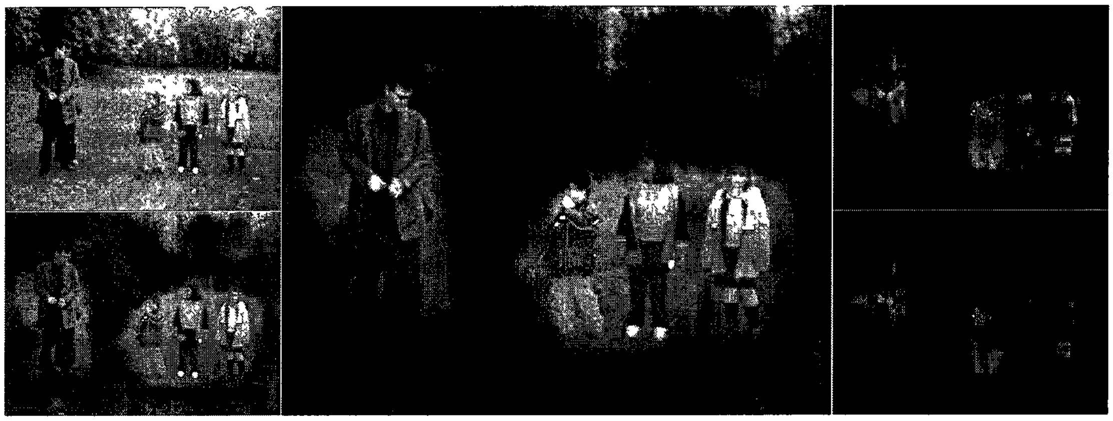


Figure 4.1: **Upper Left:** Original image. **Middle:** Labeling weighted by the confidence for the class "person". **Lower Left:** Labeling weighted by the confidence, with low confidence background pixels reclassified as foreground. **Right:** Labeling weighted by the confidence, with low confidence foreground pixels reclassified as background.

tionaries commonly used in bag of features are not designed for the specific task of categorization, so there may be gains to be found in creating "smarter" dictionaries that are tailored to the task. This is precisely what we set out to do. With this we can obtain robust, efficient localization and show that our scheme performs better than the state of the art [MS07] on a challenging dataset [OP05] despite its simplicity.

4.2 Contributions

In this section we propose a method for pixel-level category recognition and localization. We employ a simple representation (bag of features) and contribute three techniques that make the categorization process efficient. First, we extend integral images [VJ01] to windowed histogram-based classification. Second, we construct small dictionaries that maintain the performance of their larger counterparts by using agglomerative information bottleneck (AIB) [ST99]. In order to greatly reduce the bottleneck of quantizing features, we construct the large dictionaries using hierarchical k-means (HKM). We also propose an important speedup that makes it possible to compute AIB on large dictionaries with ease. Third, we show that we can compute SIFT features densely in linear time.

4.3 Related work

Lazebnik *et al.* [LR07] also perform discriminative learning to optimize k-means, but are limited to small dictionaries and visual words which are Voronoi cells. Leibe *et al.* [LMS06] also perform compression, but not in a discriminative sense. Finally, Winn *et al.* [WCM05] do discriminative compression in a similar fashion, but we show that we perform better and can scale to larger dictionaries. For the task of pixel-level localization, we show that our method outperforms k-means and Winn *et al.*, while being nearly two hundred times faster to construct. We compare our method directly to Winn *et al.* [WCM05] in Section 4.5.2.2.

Object categorization methods have matured greatly in recent years, going beyond bags of features by incorporating a spatial component into their model. Approaches are varied but broadly tend to include one of the following: interactions between pairs of features [MS06, LHS07, LS07a], absolute position of the features [LSP06], segmentation [CF07, RVG07], or a learned shape or parts model of the objects [MS07, LLS04, FMR07]. Our method exploits interaction between groups of features (all features in the window), without explicitly represent their configuration, in the spirit of achieving viewpoint-invariance for objects of general shape [VS05].

Regarding object localization, recent works are based on two different approaches: either they form a shape based model of the object class as in [MS07, LLS04], or they enforce spatial consistency using a conditional random field (CRF) [SWR06, HZC04]. We focus our comparisons on the method of Marszalek *et al.* [MS07], who forms a family of shape models for each category from the training data and casts these into the target image on sparse feature points to find local features that agree on the deformation of one of the learned models. Our approach improves performance by simply performing local classification at every pixel.

Along the way, we will construct a small, smart dictionary that is comprised of clusters of features from a much larger dictionary using AIB [ST99]. Liu *et al.* [LS07b] proposed a co-clustering scheme maximizing mutual information (MMI) for scene recognition. Agarwal *et al.* [AT05] cluster features to create a whole image descriptor called a "hyperfeature" stack. Their scheme repeatedly quantizes the data in a fixed pyramid, while our representation allows the computation of any arbitrary window without incurring any additional computational penalty. We can just as easily extract our bag of features for the whole image, blocks of the image, or (as we show) each pixel on a grid.

Lampert *et al.* [LBH08] use branch-and-bound to search all possible subwindows of an image for the window which best localizes an object. They do not seek to localize at the pixel level or handle multiple objects in one image. Shotton *et al.* [SJC08] perform pixel labeling as we do but use much simpler features combined with randomized decision forests. Because they use simple features, they must build their viewpoint invariance by synthetically warping the training images and providing them as new training examples. Our framework allows for that, but our descriptors already exhibit reduced sensitivity to viewpoint.

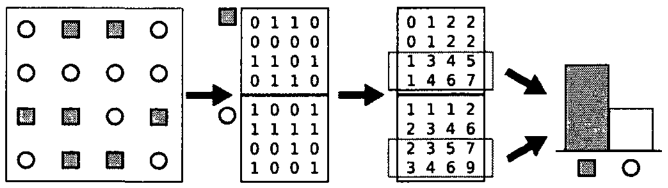


Figure 4.2: **Bag of features with integral images.** An illustration showing how integral images can be used for histogram construction in a bag of features setting. Features extracted on a regular grid have been quantized into either red squares or blue circles. Two images are constructed, one with occurrences of squares and the other with circles. The images are transformed into integral images, which are used to construct the histogram for a window covering the bottom half of the image.

4.4 Brute-force Localization

Our method uses bag of features as a "black box" to perform pixel-level category recognition. In the simplest case, this involves extracting features from the image and then for each window aggregating them into a histogram and comparing this histogram with the training data. But, extraction of a histogram at a window around each pixel of the image is prohibitively slow. To speed it up, we use integral images. However, this alone is not sufficient: Using large dictionaries in the setting we propose would be impossible, yet we need our dictionary to remain discriminative in order to be useful. To this end, in Section 4.5 we propose a method for building a compact, efficient and informative dictionary.

4.4.1 Integral Images

Viola *et al.* [VJ01] popularized the use of integral images for the task of feature extraction in boosting, and it has since been used by others [SWR06] for similar purposes. Integral images can also be used to quickly count events in image regions [WDS07], and Porikli [Por05] shows how to compute integral histograms in Cartesian spaces. We build integral images of spatial occurrences of features and use them to efficiently extract histograms of visual words on arbitrary portions of the image. For each visual word b in our dictionary, let $O_b(x, y)$ be the number of occurrences of b at pixel (x, y) (typically this number is either zero or one). Each image O_b is transformed into a corresponding integral image I_b by summing over all the pixels $(x', y') \leq (x, y)$ above and to the left of pixel (x, y):

$$I_b(x, y) = \sum_{x'<x} \sum_{y'<y} O_b(x', y')$$

Let R be a rectangular image region. The histogram $H_R(b)$ is the number of occurrences of b in R and can be quickly computed as:

$$H_R(b) = I_b(x_s, y_s) + I_b(x_e, y_e) - I_b(x_s, y_e) - I_b(x_e, y_s)$$

where (x_s, y_s) is the upper left corner and (x_e, y_e) is the lower right corner of R (Fig. 4.2). In this way we can extract a histogram of feature occurrences for a window of arbitrary size in constant time. The memory required scales with the size of the image and the size of the dictionary, and the constant time required to construct each histogram scales linearly with the size of the dictionary. This precludes the use of very large dictionaries, because each dictionary element that is included requires adding an integral image.

4.5 Informative, Compact, and Efficient Dictionaries

Our localization method directly benefits from having a small dictionary because the complexity is linear in its size. Yet many works [NS06, PCI07, TS07, MTJ06] indicate that large or very large dictionaries perform better for both object recog-

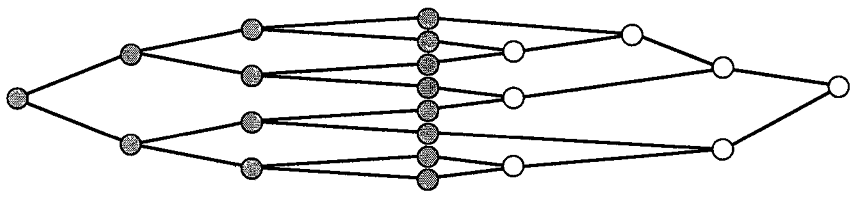


Figure 4.3: **Dictionary architecture.** We use hierarchical k-means (HKM) to build a vocabulary tree (left, red nodes) of finely quantized features by recursively partitioning the data. Next, we use AIB to build an agglomerative tree (right, blue nodes) of informative words. This architecture is efficient (in training and testing) and powerful.

nition and categorization. However, over-specific visual words should eventually over-fit the data, especially in categorization. We argue that one of the reasons why large dictionaries often outperform smaller ones is that dictionaries are usually not optimized for discrimination. If visual words could be tailored to discriminate different categories, a smaller number of them would be sufficient. Motivated by this idea, we seek to gain the performance increases of recent approaches using large dictionaries without their computational burden.

Winn *et al.* [WCM05] introduced the idea of constructing small and informative visual dictionaries by compressing larger ones. Here we propose a novel architecture and compression algorithm that has two key advantages: (i) it is very fast to project novel features onto the optimized dictionary and (ii) compression is several orders of magnitude faster, which makes it possible to operate on much larger dictionaries and datasets. In addition, we show that our method outperforms [WCM05] for the task of pixel level categorization (Section 4.6).

4.5.1 Fast Projection by Hierarchical k-means

In order to project N novel features $f \in \mathcal{F} \subset \mathbb{R}^n$ onto a visual dictionary of L elements, the required time is usually $O(NL)$. This is true even if the dictionary is eventually compressed into a smaller one [WCM05]. Since a large number of features N are typically extracted from an image, mapping features to the visual dictionary may become the bottleneck of the recognition pipeline.

Here we solve this problem by using an hierarchical k-means (HKM) [NS06] tree as the initial visual dictionary. Hierarchical k-means trees have shown excellent performance in object recognition [NS06, PCI07]. More importantly, they enable efficient projection of novel features, requiring only $O(N \log L)$ operations. Combining the HKM tree with the compression tree (Section 4.5.2), yields the coarse-to-fine-to-coarse architecture of Fig. 4.3.

4.5.2 Dictionary Compression

We compress a visual dictionary by merging visual words in such a way that the discriminative power of the dictionary is preserved. The discriminative power can be characterized in different ways, yielding different compression algorithms. Here we discuss and compare two: Agglomerative Information Bottleneck (AIB) [ST99] and the method from [WCM05], which we indicate with WCM. We also contribute a modification of the AIB algorithm that makes it feasible to process dictionaries of tens of thousands of elements. We show that the same fast algorithm can be used to speed-up WCM as well. However, even with this speedup we find that WCM is much slower than AIB (to the point of being infeasible for large datasets and dictionaries) and performs worse than AIB when applied to pixel-level categorization.

4.5.2.1 AIB Compression

AIB characterizes the discriminative power of the dictionary $\mathcal{X}$ as the mutual information $I(x, c)$ of the random variables x (visual word) and c (category):

$$I(x,c) = \sum_{x \in X} \sum_{c=1}^{C} P(x,c) \log \frac{P(x,c)}{P(x)P(c)}. \tag{4.1}$$

The joint probability $P(x, c)$ is estimated from data simply by counting the number of occurrences of each visual word $x \in \mathcal{X}$ in each category $c \in \{1, \ldots, C\}$. AIB iteratively compresses the dictionary $\mathcal{X}$ by merging the two visual words x_i and x_j that cause the smallest decrease D_{ij} in the mutual information (discriminative power) $I(x, c)$. Denoting $[x]_{ij}$ the random variable corresponding to the dictionary after the merge, the quantity D_{ij} is

$$D_{ij} = I(x,c) - I([x]_{ij}, c). \tag{4.2}$$

The information $I(x, c)$ is monotonically reduced after each merge. Merging is iterated until one obtains the desired number of words.

At test time, projecting a visual word $x \in \mathcal{X}$ onto the compressed dictionary requires constant time ($O(1)$). So, since we use HKM for the initial dictionary, the number of operations required to project N novel features on the compressed dictionary is only $O(N \log K)$, where K is the number of leaves of the HKM tree.

In Fig. 4.4 we show the effectiveness of this technique using a simple experiment on Graz-02. In all cases, we compress the dictionary significantly without losing any accuracy. In fact, in two of the three cases the results are slightly improved at some compression level.

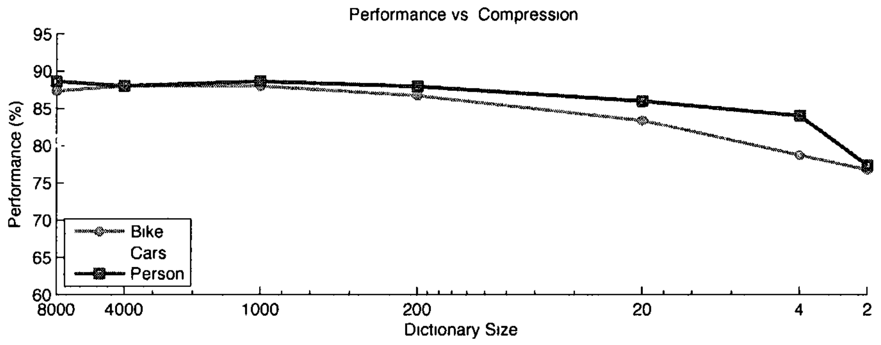


Figure 4.4: **Results of an experiment showing the performance of AIB as the dictionary is compressed.** We adopt the framework of [ZML06] on Graz-02, extracting SIFT descriptors on salient regions, quantizing them, and classifying the resulting histograms with an SVM. We vary the compression of the dictionary, starting from the full hierarchical k-means tree (8,000 leaves, K=20) and compressing to a dictionary with only 2 elements. In each case, we can compress the dictionary by a factor of 8 without losing any accuracy. In some cases (Cars, Bikes) we even increase performance slightly.

4.5.2.2 Fast AIB

The basic implementation of the AIB algorithm is prohibitively slow for very large dictionaries. The implementation proposed in Slonim *et al.* [ST99] stores the symmetric "distance" matrix $D = [D_{ij}]$ ($O(L^2)$ space).[1]

Then, at each iteration one only needs to update the row and column i, j of D which were involved in the last merge (since only words x_i and x_j change). This has complexity $O(LC)$. Searching for the minimal matrix element at each step is $O(L^2)$, and this process is iterated L times, so the overall complexity is $O(L(L^2 + LC))$ time and $O(L^2)$ space [Slo03].

A simple modification of the basic algorithm is far more efficient. For each i, we cache the index and value (k_i, D_{ik_i}) of the minimum distance along the row and do not store D. This reduces the time spent searching for the minimum element (i^*, j^*) of D from $O(L^2)$ to $O(L)$. Now, when we merge (i^*, j^*), we must update the entries (k_i, D_{ik_i}) for which either $k_i = i^*$ or $k_i = j^*$. This has time complexity $O(L(L + \gamma LC))$, where γ is the number of entries that need to be updated at each iteration. We find empirically that $\gamma \ll L$, so in practice the amount of time taken is approximately $O(L^2C)$ and the space complexity has been reduced to $O(L)$.

To get a sense of the advantages of this implementation, the original AIB algorithm [Slo03] requires L^2 elements of memory at each iteration, which meant that a 20,000 cluster case would require roughly 3.2GB of memory as opposed to 320kB with our modified approach. We also note that in the 10,000 cluster cases

[1]**Reciprocal Nearest Neighbor Clustering** [LMS06] proposes an efficient agglomerative clustering algorithm that can be applied whenever the distance matrix D_{ij} satisfies the *reducibility property* $D_{ij} \leq \min\{D_{ik}, D_{jk}\} \Rightarrow \min\{D_{ik}, D_{jk}\} \leq D_{\bar{ij},k}$, where $\bar{ij}$ denotes the merged dictionary entry. Unfortunately, AIB clustering violates this property. For a counter example, consider the case $C = 3$, $P(x_i) = P(x_j) = P(x_k) = 1/3$, $P(c = 1|x_k) = P(c = 2|x_k) = 1/3$, $P(c = 1|x_i) = P(c = 2|x_i) = 2/5$ and $P(c = 2|x_j) = P(c = 3|x_j) = 2/5$.

we test, we often find γ to be on the order of 5 and so the clustering process is very fast (about 5 minutes for 10,000 clusters on a 2.3Ghz Core 2 Duo). The basic implementation of AIB on the same task requires approximately a day.

4.5.2.3 WCM compression

WCM differs from AIB in the way it measures the discriminative power of the visual dictionary. This is motivated by the fact that in the bag of features setting images are represented by histograms of visual words rather than visual words in isolation. Thus, one is more interested in obtaining *informative histograms* than informative visual words. This notion could be captured, for instance, by considering the mutual information $I(h, c)$ in place of the information $I(x, c)$ used by AIB.

Due to the high dimensionality of the histograms, estimating $I(h, c)$ is nearly impossible without strong assumptions. WCM assumes that histograms are distributed according to a mixture of Gaussians, with one Gaussian per category. Moreover, they characterize the discriminative power of the dictionary by the category posterior probability $p(c|h)$ rather than by the information $I(h, c)$. This creates a mechanism for model selection which can automatically stop the merging procedure when a maximum of $p(c|h)$ is attained (in contrast, in AIB the information criterion $I(x, c)$ decreases monotonically). Finally, it is also possible to extend the fast AIB algorithm introduced in the previous section to WCM almost without changes.

Despite these appealing characteristics, WCM does not perform as well as AIB in our setting. First, despite our fast implementation, it is much slower than AIB on large datasets (in Section 4.6 we show it requires up to twelve days on a task that our fast AIB can solve in about five minutes). WCM is much slower because updating an entry of the D_{ij} matrix requires scanning the data to compute the linear correlation of bin i and j. This is due to the fact that WCM considers visual words in the context of histograms where AIB does not. Although the model assumes that histogram bins are statistically independent, they interact when merged. The update operation requires about $O(ML^2C)$, where M is the number of training histograms, as opposed to $O(L^2C)$ for AIB. Second, WCM model selection is not useful for our localization task as we are interested in obtaining dictionaries of a prescribed size (Section 4.6). Third, AIB compressed dictionaries result in better categorization results than WCM (Section 4.6; Table 4.1). This is probably because in our setting the assumptions made by WCM are not satisfied.

4.6 Experiments

Graz-02 [OP05] is a challenging dataset consisting of three categories (cars, bicycles, and people) with extreme variability in pose, scale and lighting. Our goal is the same as Marszalek *et al.* [MS07]: We wish to label each image pixel as either belonging to one of these categories or not. In order to compare directly to Marszalek *et al.* [MS07], we adopt their measure of performance: pixel precision-recall. Our features extraction and dictionary compression are implemented within VLFeat [VF08], and the rest of our implementation is available as a part of Blocks [FV09] which is described in more detail in Appendix B.

4.6.1 Training

We select the same training images as [MS07], namely the first 150 odd numbered images from each category. We compute dense SIFT descriptors and quantize them using our dictionary (see Section 4.6). Then, for each image we generate two histograms: The first aggregates all the features that belong to the background (based on the feature center and the ground truth object masks), and the second the features that belong to the object. This collection of histograms is used as training data for either an SVM classifier with χ^2 kernel or an inverse document frequency (IDF) [NS06] weighted k-nearest neighbor (KNN) classifier ($k = 10$).

4.6.2 Fast Dense Feature Extraction

We extract a SIFT descriptor [Low04] every four pixels. The support of each descriptor is a 16×16 patch. We do not compute the orientation of the descriptor since this has been shown to adversely affect other dense bag of features methods [ZML06]. Features that have low gradient magnitude before normalization are discarded as in [LSP06, TS07].

We introduce here a novel technique to compute dense SIFT descriptors very efficiently. Fast SIFT-like descriptors have been proposed by [BTG06, TS07] and recently [TLF08]. Our technique has the advantage of being fully equivalent to SIFT and still efficient: The complexity is only $O(Q^2R)$ compared to $O(Q^2R^2)$ of a direct implementation, where Q^2 the area of the image and R^2 the area of the descriptor support. Moreover, up to a small approximation, we can reduce the complexity to $O(Q^2)$, which is independent of the area of the descriptor support. Our implementation is included with VLFeat [VF08], an open source feature extraction library.

The idea is to reduce the calculation of the dense descriptors to a number of separable convolutions. Recall that the SIFT descriptor at location (x_0, y_0) is a three-dimensional histogram of the gradient $\nabla I(x, y)$ in a circular patch surrounding that point [Low04]. The histogram is indexed by the relative position

$(x - x_0, y - y_0)$ and orientation $\angle\nabla I(x, y)$ of the gradient $\nabla I(x, y)$ in the patch, weighed by the gradient magnitude $|\nabla I(x, y)|$ and by a Gaussian window centered at (x_0, y_0). The relative positions are quantized in 4×4 bins and the orientation in 8 bins using bilinear interpolation. For a given orientation, the data for a bin b is obtained by computing integrals like:

$$d(x_0, y_0, b) = \int g(x - x_0, y - y_0) h_b(x - x_0, y - y_0) f(x, y)\, dx\, dy \qquad (4.3)$$

where $f(x, y)$ is the mass of the gradient at that particular orientation, $g(x, y)$ is the Gaussian window and $h_b(x, y)$ is the product of two triangular windows resulting from the bilinear interpolation of bin b. Since both $h(x, y)$ and $g(x, y)$ are separable, the calculation requires only $O(Q^2 R)$ operations.

Notice that this requires $4 \times 4 \times 8$ separable convolutions in total. However, by dropping the Gaussian window $g(x, y)$ (the effect on the computed descriptors is modest), convolutions for different spatial bins at the same orientations are identical up to translation, and only 8 separable convolutions are sufficient. Moreover, recall that convolving by a rectangular kernel can be done very efficiently by integral images. Since convolving by a triangular kernel can be decomposed into convolving twice by rectangular ones, we obtain a final complexity of $O(Q^2)$. On 640×480 images, such as those in Graz-02, fast dense feature extraction takes 0.15 seconds for grayscale and 0.3 seconds for RGB.

Our RGB SIFT descriptors are formed by first transforming the (R, G, B) image into the normalized (r, g, b) space [EHD00] where:

$$
\begin{aligned}
r &= \tfrac{R}{R+G+B} && (4.4)\\
g &= \tfrac{G}{R+G+B} && (4.5)\\
b &= \tfrac{B}{R+G+B} && (4.6)
\end{aligned}
$$

SIFT descriptors are then extracted independently from the r and g channels and concatenated into one 256 dimensional descriptor. We do not include the b channel because the constraint $r + g + b = 1$ makes it redundant.

4.6.3 Dictionary Construction

We sample a large number of feature-category pairs from our training data and follow one of two approaches to construct a dictionary. As a baseline, we use k-means with $k = \{5, 40, 200\}$. Alternatively, we construct a hierarchical k-means dictionary with $k = 10$ and 10,000 leaf nodes, and then compress this dictionary to $N = \{5, 40, 200\}$ clusters (we experiment with both AIB and WCM). Notice that, in our application, the size of the dictionary is the primary factor in determining the speed and memory footprint of the classification algorithm.

4.6.4 Testing

We test on the first 150 even numbered images from each category. Every 4 pixels, we construct a histogram of feature occurrences within a window of 80×80 pixels using integral images (Section 4.4) and classify using either SVM or KNN. The classification returns a label and a score. The magnitude of the score indicates the confidence in the label and the sign of the score indicates the class (-1 is a fully confident classification of "background"). For pixels which do not lie on the grid, we interpolate the score from adjacent pixels.

We choose a range of confidence thresholds ρ and for each we classify as object all pixels which have a score greater than the threshold. These are compared to the ground-truth segmentation which provides us with pixel precision and recall for the testing data. We also use this threshold to create Fig. 4.1 and to generate the movie included as supplementary material.

4.6.5 Discussion

Table 4.1 reports the points where precision and recall are equal and compares our results to those of Marszalek *et al.* [MS07], the previous state of the art in pixel accurate localization on Graz-02. We also compare to (and outperform) Winn *et al.* (WCM), and while our dictionaries take roughly 5 minutes to construct, Winn *et al.* takes up to 12 days on this task. Fig. 4.6 shows the full precision-recall curves for each category and classification method. Our approach is fast, the times reported in Table 4.1 include dense feature extraction, quantization, and the classification of all pixels.

Although we do not have shape or even scale in our model, we still perform significantly better on all categories. Specifically, our best performing cases are 4.8% better on bikes, 0.9% better on cars, and 7.3% better on people. In each case, the compressed dictionary outperforms the k-means dictionary of equal size. The differences decrease as the final vocabulary size is increased, which is intuitive because the variability of the dataset can be better captured by k-means as we increase k, while the descriptive power of our rebuilt dictionary is upper bounded by that of the associated HKM tree.

Our approach naturally provides a confidence measure, so we can quantify the uncertainty in classification as shown in Fig. 4.5.

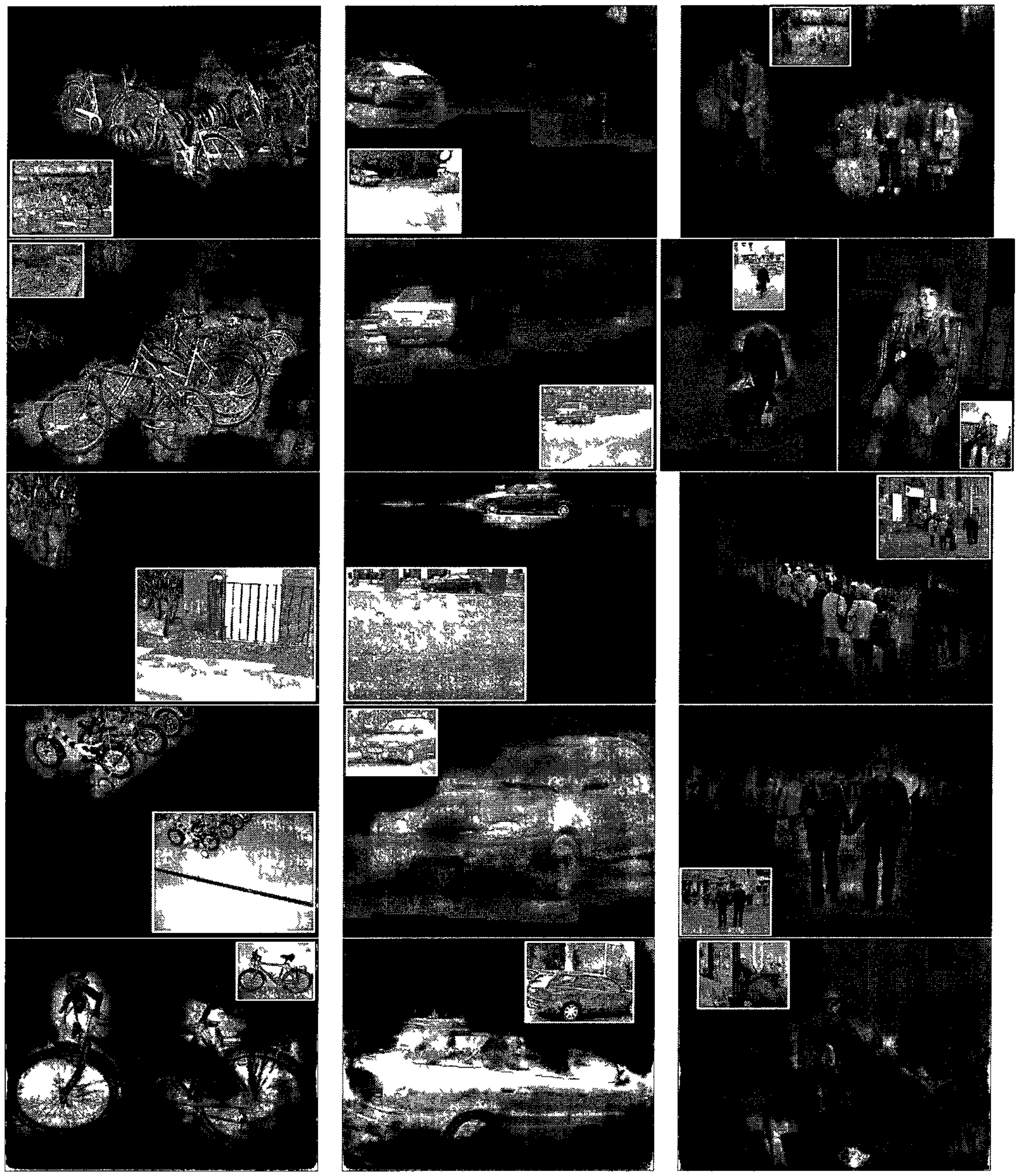


Figure 4 5 Selected results on Graz-02. Images are first masked by the classification then transformed to HSV The HSV images have their V channel weighted by the confidence in the classification, darkening the pixels which are less confident about the class All images shown were generated with the parameter set denoted AIB200RGB and classified with an SVM

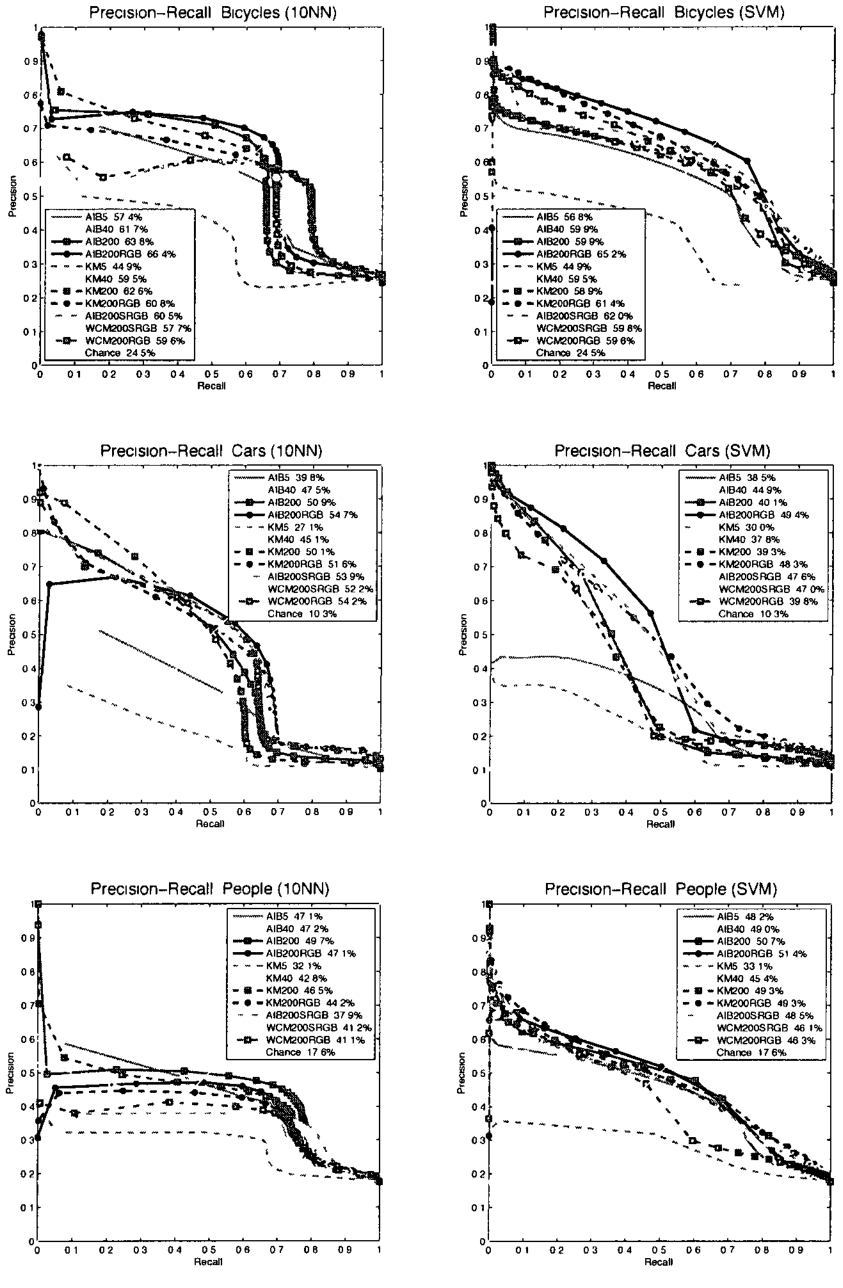


Figure 4.6: **Pixel precision-recall curves.** The row corresponds to the category and the column specifies which classifier was used. The yellow dashed line indicates the performance of random guessing of the pixel identity. When the vocabulary size is small, AIB helps greatly.

4.7 Conclusions

We have described and shown that an object localization framework that uses bag of features as a tool can successfully localize objects without making assumptions about the shape of the object, or explicitly performing segmentation. In order to make this possible, we have also shown a method that efficiently learns a dictionary which is tailored for the task of categorization. In spite of its simplicity, our approach produces pixel-accurate object localizations which exceed the state of the art on a challenging dataset.

Our experiments show that more care should be exercised in integrating shape information into generic object class representations. We believe shape is an important discriminant (See [VS05], Theorem 3), but our work should be viewed as a baseline method whose performance should be convincingly exceeded before justifying the additional complexity a shape-based model might bring.

object class	cars	people	bicycles	time
[MS07] no hyp. eval.	40.4%	28.4%	46.6%	-
[MS07] no evid. collect.	50.3%	40.3%	48.9%	-
[MS07] full framework	53.8%	44.1%	61.8%	-
AIB5-KNN	39.8%	47.1%	57.4%	0.5s
AIB5-SVM	38.5%	48.2%	56.8%	0.7s
KM5-KNN	27.1%	32.1%	44.9%	2s
KM5-SVM	30.0%	33.1%	44.9%	2s
AIB40-KNN	47.5%	47.2%	61.7%	0.5s
AIB40-SVM	44.9%	49.0%	59.9%	0.8s
KM40-KNN	45.1%	42.8%	59.5%	2s
KM40-SVM	37.8%	45.4%	59.5%	2.5s
AIB200-KNN	50.9%	49.7%	63.8%	1.1s
AIB200-SVM	40.1%	50.7%	59.9%	3.3s
KM200-KNN	50.1%	46.5%	62.6%	2.5s
KM200-SVM	39.3%	49.3%	58.9%	5s
AIB200RGB-KNN	**54.7%**	47.1%	**66.4%**	1.4s
AIB200RGB-SVM	49.4%	**51.4%**	65.2%	3.7s
WCM200RGB-KNN	54.2%	41.1%	59.6%	1.4s
WCM200RGB-SVM	39.8%	46.3%	59.6%	3.7s
KM200RGB-KNN	51.6%	44.2%	60.8%	3.5s
KM200RGB-SVM	48.3%	49.3%	61.4%	7s

Table 4.1: **A comparison of the pixel precision-recall equal error rates on Graz-02.** Although we do not represent shape explicitly, our results are competitive with [MS07]. The best performance is achieved using our compressed dictionary (Section 4.5).

CHAPTER 5

Superpixel Categorization

5.1 Introduction

Recent success in image-level object categorization has led to significant interest on the related fronts of localization and pixel-level categorization. Both areas have seen significant progress lately through object detection challenges including the PASCAL VOC [EVW]. So far, the most promising techniques seem to be those that consider each pixel of an image.

For localization, sliding window classifiers [DT05, BL08, LBH08] consider a window (or all possible windows) around each pixel of an image and attempt to find the classification which best fits the model. Lately, this model often includes some form of spatial consistency (e.g. [LSP06]). In this way, we can view sliding window classification as a "top-down" localization technique which tries to fit a coarse global model to each possible location.

In object class segmentation, the goal is to produce a pixel-level segmentation of the input image, so on the surface it seems that classifying each pixel may be a sensible thing to do. Most approaches here are built from the bottom up on learned local representations (e.g. TextonBoost [SWR06]) and can be seen as an evolution of texture detectors. Because of their rather local nature, they often introduce a conditional random field [LMP01] or some other model to enforce spatial consistency. For computational reasons, this usually operates on a reduced grid of the image, abandoning pixel accuracy in favor of speed. A current leader of the PASCAL VOC 2007 Segmentation Challenge is a scheme which falls into this category ([SJC08]).

We argue that pixels are discrete structures that are unrelated to the content of the image and therefore they represent a level of granularity that is poorly suited for operations on objects. Instead, we consider small regions obtained from a conservative over-segmentation, or "superpixels," to be the elementary unit of detection, categorization and localization schemes [RM03, FH04, MPW08].

On the surface, using superpixels as the elementary units seems counter-productive, because aggregating pixels into groups entails a decision that, in general, reduces the performance of any classifier downstream when compared to an equivalent one acting directly on the data ([Sha98], page 88 and [Rob01], Theorem 7.4).

However, aggregating pixels into superpixels captures all the "obvious" local information. The superpixels we construct are quite conservative, so we minimize the risk of merging unrelated pixels. At the same time, moving to superpixels allows us to measure feature statistics (in this case: histograms of visual words) on a naturally adaptive domain rather than a fixed one, such as a window. Since superpixels tend to preserve boundaries, we also have the opportunity to create a much tighter segmentation by simply finding the superpixels which are part of the object.

We show that by aggregating neighborhoods of superpixels we can create a robust region classifier which exceeds the state-of-the-art on Graz-02 pixel-localization and on the PASCAL VOC 2007 Segmentation Challenge. Our results can be further refined by a conditional random field (CRF) that operates on superpixels, which we propose in section 5.3.4.

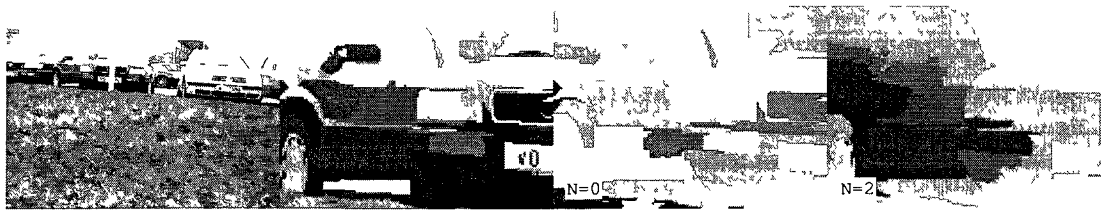


Figure 5.1: **Aggregating histograms.** An illustration of the detail of our superpixel segmentation and the effectiveness of aggregating histograms from adjacent segments. From left: the segmentation of a test image from Graz-02, a zoomed in portion of the segmentation, the classification of each segment where more red is more car-like, and the resulting classification after aggregating all histograms within $N = 2$ distance from the segment being classified.

5.2 Related Work

Sliding window classifiers have been well explored for the task of detecting the location of an object in an image [BL08, LBH08, DT05]. Lambert *et al.* [LBH08] have shown that it is feasible to search all possible sub-windows of an image for the one which best matches the object using branch and bound. Meanwhile, Blaschko *et al.* [BL08] present a way to learn a structured classifier whose output is a bounding box. However, for our purposes a bounding box is not an acceptable final output, even in the case of localization

Our localization capability is more comparable to Marszałek [MS07] or Fulkerson *et al.* [FVS08] (See also Chapter 4). Marszałek warps learned shape masks into an image based on distinctive local features Fulkerson performs bag of features classification within a local region, as we do, but the size of the region is fixed (a rectangular window) and how to determine the best size for this window is not clear. In contrast, our method provides a natural neighborhood size, expressed in terms of low level image regions (the superpixels). We show that we greatly improve on the results of Fulkerson *et al.* in Table 5.2.

Class segmentation algorithms that operate at the pixel level are often based on local features like textons [SWR06] and are augmented by a conditional random field or another spatial coherency aid [HZC04, KH05, HZR06, VT07, RVG07, GRC08] to refine the results. In this setting, Shotton *et al.* [SJC08] constructs semantic texton forests for extremely fast classification. Semantic texton forests are essentially randomized forests of simple texture classifiers which are themselves randomized forests. We compare our results with and without an explicit spatial aid (a CRF) with those of Shotton in Table 5.3. Another notable work in this area is that of Gould *et al.* [GRC08] who proposed a superpixel-based CRF which learns relative location offsets of categories. We eventually augment our model with a CRF on superpixels, but we do not model the relative locations of objects explicitly. Instead, we use much stronger local features and learn context via the connectedness in the superpixel graph.

A number of works utilize one or more segmentations as a starting point for their task. An early example is Barnard *et al.* [BDG03], who explore associating labels with image regions using simple color features and then merging regions based on similarity over the segment-label distribution. Russell *et al.* [RES06] build a bag of features representation on multiple segmentations to automatically discover object categories and label them in an unsupervised fashion. Similarly, Galleguillos *et al.* [GBR08] use Multiple Instance Learning (MIL) to localize objects in weakly labeled data. Both assume that at least one of their segmentations contains a segment which correctly separates the entire object from the background. By operating on superpixels directly, we can avoid this assumption and the associated difficulty of finding the one "good" segment.

Perhaps the most closely related work to this chapter is that of Pantofaru *et al.* [PSH08]. Pantofaru *et al.* form superpixel-like objects by intersecting multiple segmentations and then classify these by averaging the classification results from all of the member regions. Their model allows them to gather classification information from a number of different neighborhood sizes (since each member segment has a different extent around the region being classified). However, multiple segmentations are much more computationally expensive than superpixels, and we significantly exceed their performance on the VOC 2007 dataset (see Table 5.3).

Additionally, a number of authors use graphs of image structures for various purposes, including image categorization [HB07, NTU07] and medical image classification [AAB07]. Although we operate on a graph, we do not seek to mine discriminative substructures [NTU07] or classify images based on the similarity of walks [HB07]. Instead, we use the graph only to define neighborhoods and optionally to construct a conditional random field.

5.3 Superpixel Neighborhoods

5.3.1 Superpixels

We use the publicly available implementation of quick shift [VS08] to extract superpixels from our input images (See Appendix A for a fast GPU implementation). Our model is quite simple: we perform quick shift on a five-dimensional vector composed of the LUV colorspace representation of each pixel and its location in the image.

Unlike superpixelization schemes based on normalized cuts (e.g. [RM03]), the superpixels produced by quick shift are not fixed in approximate size or number.

A complex image with many fine scale image structures may have many more superpixels than a simple image, and there is no parameter that puts a penalty on the boundary, leading to superpixels which are quite varied in size and shape. Statistics related to our superpixels (such as the average size and degree in the graph) are detailed in Section 5.4.

This produces segmentations, like the one in Figure 5.1, which consist of many small regions that preserve most of the boundaries in the original image. Since we perform this segmentation on the full resolution image, we leave open the potential to obtain a nearly pixel-perfect segmentation of the object.

5.3.2 Classification

We construct a bag of features classifier that operates on the regions defined by the superpixels we have found. SIFT descriptors [Low04] are extracted for each pixel of the image at a fixed scale and orientation using the fast SIFT framework described in Chapter 5 and found in [VF08]. The extracted descriptors are then quantized using a k-means dictionary and aggregated into one histogram for each superpixel. Each superpixel is assigned the most frequent class label it contains, and a one-vs-rest support vector machine (SVM) with an RBF χ^2 kernel is learned on the labeled histograms.

The classifier which results from this is very specific. It finds superpixels which resemble superpixels that were seen in the training data without considering the surrounding region. This means that while a wheel or grill on a car may be correctly identified, the nearby hub of the wheel or the headlight can be detected with lower confidence or missed altogether (Figure 5.1).

Another drawback of learning a classifier for each superpixel is that the histograms associated with each superpixel are very sparse, often containing only a handful of non-zero elements. This is due to the nature of our superpixels; by definition they cover areas that are roughly uniform in color and texture. Since our features are fixed-scale and extracted densely, our superpixels sometimes contain tens or even hundreds of descriptors that quantize to the same visual word.

5.3.3 Superpixel Neighborhoods

We address both of the problems mentioned in the previous section by introducing histograms based on superpixel neighborhoods. Let $G(S, E)$ be the adjacency graph of superpixels $s_i \in S$ in an image, and H_i^0 be the unnormalized histogram associated with this region. E is the set of edges formed between pairs of adjacent superpixels (s_i, s_j) in the image and $D(s_i, s_j)$ is the distance in the graph between two superpixels. Then, H_i^N is the histogram obtained by merging the histograms of the superpixel s_i and neighbors who are less than N nodes away in the graph:

$$H_i^N = \sum_{s_j | D(s_i, s_j) \leq N} H_j^0$$

Using these histograms in classification addresses both of our previous issues. First, since adjacent superpixels must be visually dissimilar, histograms constructed from superpixel neighborhoods contain more diverse features and are therefore less sparse. This provides a regularization for our SVM, reducing overfitting. It also provides spatial consistency in our classification because as we increase N, histograms of adjacent superpixels have more features in common.

Second, because we are effectively increasing the spatial extent of the region considered in classification, we are also providing our classifier with a better description of the object. As we increase N, we move from the "part" level to the "object" level, and since not all training superpixels will lie on the interior of the object, we are also learning some "context".

However, note that as N becomes larger we will blur the boundaries of our objects since superpixels that are on both sides of the object boundary will have similar histograms. In the next section, we explore adding a CRF to reduce this effect.

5.3.4 Refinement with a CRF

In order to recover more precise boundaries while still maintaining the benefits of increasing N, we must introduce new information which will allow us to reduce misclassifications that occur near the edges of objects. Conditional random fields provide a natural way to incorporate such constraints by including them in the pairwise edge potential of the model. Let $P(\mathbf{c}|G; w)$ be the conditional probability of the set of class label assignments $\mathbf{c}$ given the adjacency graph $G(S, E)$ and a weight w:

$$-log(P(\mathbf{c}|G; w)) = \sum_{s_i \in S} \Psi(c_i|s_i) \ + \ w \sum_{(s_i, s_j) \in E} \Phi(c_i, c_j|s_i, s_j)$$

Our unary potentials Ψ are defined directly by the probability outputs provided by our SVM [CL01] for each superpixel:

$$\Psi(c_i|s_i) = -log(P(c_i|s_i))$$

and our pairwise edge potentials Φ are similar to those of [SWR06, BJ01]:

$$\Phi(c_i, c_j|s_i, s_j) = \left(\frac{L(s_i, s_j)}{1 + ||s_i - s_j||}\right)[c_i \neq c_j]$$

where $[\cdot]$ is the zero-one indicator function and $||s_i - s_j||$ is the norm of the color difference between superpixels in the LUV colorspace. $L(s_i, s_j)$ is the shared boundary length between superpixels s_i and s_j and acts here as a regularizing term which discourages small isolated regions.

In many CRF applications for this domain, the unary and pairwise potentials are represented by a weighted summation of many simple features (e.g. [SWR06]), and so the parameters of the model are learned by maximizing their conditional log-likelihood. In our formulation, we simply have one weight w that represents the trade-off between spatial regularization and our confidence in the classification. We estimate w by cross-validation on the training data. Once our model has been learned, we carry out inference with the multi-label graph optimization library of [BK04, KZ04, BVZ01] using α-expansion. Since the CRF is defined on the superpixel graph, inference is very efficient, taking less than half a second per image.

Results with the CRF are presented in Section 5.4 as well as Figures 5.2 and 5.3.

5.4 Experiments

We evaluate our algorithm for varying N with and without a CRF on two challenging datasets. Graz-02 contains three categories (bicycles, cars and people) and a background class. The task is to localize each category against the background class. Performance on this dataset is measured by the pixel precision-recall.

The PASCAL VOC 2007 Segmentation Challenge [EVW] is an extremely difficult class segmentation challenge which contains 21 categories and few training examples. While the challenge specifies that the detection challenge training data may also be used, we use only the ground truth segmentation data for training. The performance measure for this dataset is the average pixel accuracy: for each category the number of correctly classified pixels is divided by the ground truth pixels plus the number of incorrectly classified pixels.

MATLAB code to reproduce our experiments is available as part of Blocks [FV09] (See Appendix B).

5.4.1 Common Parameters

Experiments on both datasets share many of the same parameters which we detail here.

SIFT descriptors are extracted at each pixel with a patch size of 12 pixels and fixed orientation. These descriptors are quantized into a k-means dictionary learned on the training data. All experiments we present here use $k = 400$, though in Figure 5.1 we show that a wide variety of k produce similar results.

The superpixels extracted via quick shift are controlled by three parameters: λ, the trade-off between color importance and spatial importance, σ, the scale at which the density is estimated, and τ, the maximum distance in the feature space between members of the same region. We use the same parameters for all of our experiments: $\sigma = 2$, $\lambda = 0.5$, $\tau = 8$. These values were determined by segmenting a few training images from Graz-02 by hand until we found a set which preserved nearly all of the object boundaries and had the largest possible average segment size. In principle, we could do this search automatically on the training data, looking for the parameter set which creates the largest average segment size while ensuring that the maximum possible classification accuracy is greater than some desired level. In practice, the algorithm is not too sensitive to the choice of parameters, so a quick tuning by hand is sufficient. Note that the number or size of the superpixels is not fixed (as opposed to [GRC08]): the selected parameters put a rough bound on the maximum size of the superpixels but do not control the shape of the superpixels or degree of the superpixel graph.

Histograms for varying N are extracted as described in section 5.3.3 and labels are assigned to training superpixels by the majority class vote. We randomly select an equal number of training histograms from each category as the training data for our SVM.

We learn a one-vs-rest multi-class SVM on the histograms using libsvm [CL01]. Our kernel is RBF χ^2:

$$K(X,Y) = e^{-\gamma \frac{(X-Y)^2}{X+Y}}$$

with parameters learned via cross validation on the training histograms. During testing, we convert our superpixel labels into a pixel-labeled map and evaluate at the pixel level for direct comparison with other methods.

In both experiments, we take our final SVM and include it in the CRF model described in section 5.3.4.

5.4.2 Graz-02

On Graz-02, we use the same training and testing split as Marszałek and Schmid [MS07] and Fulkerson *et al.* [FVS08]. Our segment classifier is trained on 750 segments collected at random from the category and the background.

Graz-02 images are 640 by 480 pixels and quick shift produces approximately 2000 superpixels per image with an average size of 150 pixels. The average degree of the superpixel graph is 6, but the maximum degree is much larger (137).

In Table 5.2, we compare our results for varying size N with those of Fulker-

	Graz-02 $N =$					PASCAL 2007 $N =$				
	0	1	2	3	4	0	1	2	3	4
$k = 10$	37	44	47	51	49	10	10	12	12	12
$k = 100$	48	61	64	64	64	13	19	23	25	25
$k = 200$	49	63	66	66	64	13	20	25	26	25
$k = 400$	50	64	67	69	67	14	21	25	28	27
$k = 1000$	49	63	68	68	66	14	22	27	27	26

Table 5.1: **Effect of k.** Here we explore the effect of the dictionary size k on the accuracy of our method (without a CRF) for varying neighborhood sizes N. Increasing the size of the dictionary increases performance until we begin to overfit the data. We pick $k = 400$ for our experiments, but a large range of k will work well. Notice that even with $k = 10$ we capture some information, and increasing N still provides noticeable improvement.

	Cars	People	Bicycles
[MS07] full framework	53.8%	44.1%	61.8%
[FVS08] NN	54.7%	47.1%	66.4%
[FVS08] SVM	49.4%	51.4%	65.2%
$N = 0$	43.3%	51.3%	56.7%
CRF $N = 0$	46.0%	54.3%	63.4%
$N = 1$	62.0%	62.7%	67.6%
CRF $N = 1$	69.7%	63.8%	69.7%
$N = 2$	67.1%	65.4%	69.3%
CRF $N = 2$	71.2%	**66.3%**	71.2%
$N = 3$	68.6%	65.7%	71.7%
CRF $N = 3$	**72.2%**	66.1%	**72.2%**
$N = 4$	67.1%	62.7%	71.0%
CRF $N = 4$	71.3%	63.2%	71.3%

Table 5.2: **Graz-02 results.** The precision = recall points for our experiments on Graz-02. Compared to the former state-of-the-art [FVS08], we show a 17% improvement on Cars, a 15% improvement on People and a 6% improvement on Bicycles. N is the distance of the furthest neighboring region to aggregate, as described in section 5.3.3. Our best performing case is always the CRF-augmented model described in section 5.3.4.

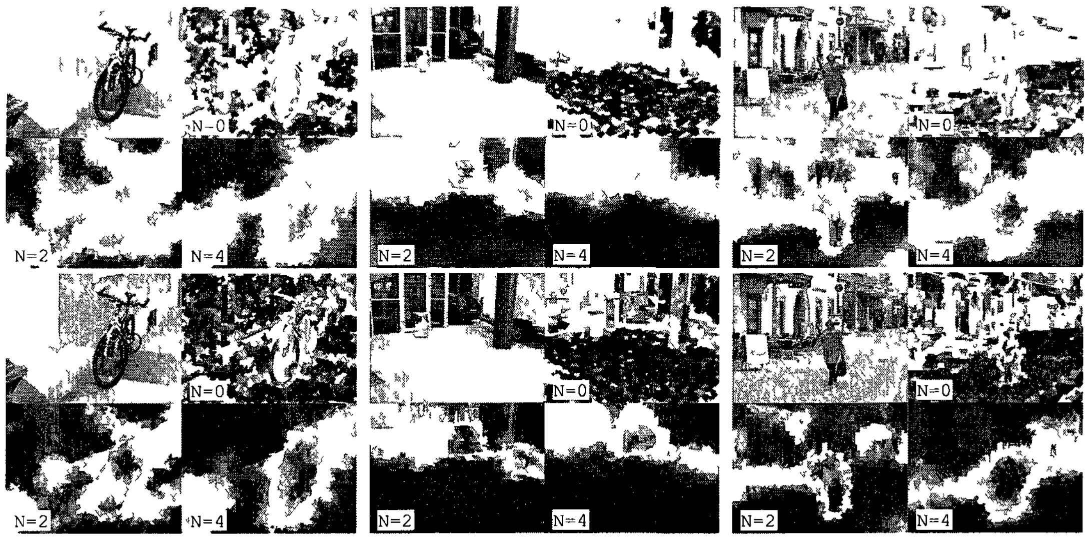


Figure 5.2: **Graz-02 confidence maps.** Our method produces very well localized segmentations of the target category on Graz-02. Here, a dark red classification means that the classifier is extremely confident the region is foreground (using the probability output of libsvm), while a dark blue classification indicates confident background classification. Notice that as we increase the number of neighbors considered (N), regions that were uncertain become more confident and spurious detections are suppressed. **Top two rows**: Without CRF. **Bottom two rows**: With CRF.

son *et al.* [FVS08] which uses a similar bag of features framework and Marszałek and Schmid [MS07] which warps shape masks around likely features to define probable regions. We convincingly improve upon the state-of-the-art in all categories (+17% on cars, +15% on people, and +6% on bicycles).

Example localizations may be found in Figures 5.1 and 5.2. Notice that although $N = 0$ produces some very precisely defined correct classifications, there are also many missed detections and false positives. As we increase the amount of local information that is considered for each classification, regions that were classified with lower confidence become more confident, and false positives are suppressed.

Adding the CRF provides consistent improvement, sharpening the boundaries of objects and providing further spatial regularization. Our best performing cases use $N = 2$ or $N = 3$, balancing the incorporation of extra local support with the preference for compact regions with regular boundaries.

5.4.3 VOC 2007 Segmentation

For the VOC challenge, we use the same sized dictionary and features as Graz-02 ($k = 400$, patch size $= 12$ pixels). The training and testing split is defined in the challenge. We train on the training and validation sets and test on the test set. Since there are fewer training images per category, for this experiment we train on 250 randomly selected training histograms from each category.

VOC 2007 images are not fixed size and tend to be smaller than those in Graz-02, so with the same parameters quick shift produces approximately 1200 superpixels per image with a mean size of 150 pixels. The average degree of the superpixel graph is 6.4, and the maximum degree is 72.

In Table 5.3 we compare with the only segmentation entry in the challenge (Oxford Brookes), as well as the results of Shotton *et al.* [SJC08], and Pantofaru *et al.* [PSH08]. Note that Shotton reports a set of results which bootstrap a detection entry (TKK). We do not compare with these results because we do not have the data to do so. However, because our classifier is simply a multi-class SVM, we can easily add either the Image Level Prior (ILP) or a Detection Level Prior (DLP) that Shotton uses. Even without the ILP, we find that we outperform Shotton with the ILP on 14 of the 21 categories and tie on one more. Our average performance is also improved by 8%. Compared to Shotton without ILP or Pantofaru, average performance is improved by 12%. Selected segmentations may be found in Figure 5.3.

This dataset is much more challenging (we are separating 21 categories instead of 2, with less training data and more variability) and because of this when $N = 0$ everything has very low confidence. As we increase N we start to see contextual relationships playing a role. For example, in the upper left image of Figure 5.4 we see that as the person classification gets more confident, so does the bike and motorbike classification, since this configuration (person above bike) occurs often in the training data. We also see that larger N tends to favor more contiguous regions, which is consistent with what we expect to observe.

On this dataset, adding a CRF improves the qualitative results significantly, and provides a consistent boost for the accuracy as well. Object boundaries become crisp, and often the whole object has the same label, even if it is not always the correct one.

5.5 Conclusion

We have demonstrated a method for localizing objects and segmenting object classes that considers the image at the level of superpixels. Our method significantly exceeds the state-of-the-art on Graz-02 and the PASCAL VOC 2007 Segmentation Challenge, even without the aid of a CRF or color information. When we add a CRF to include a boundary length penalty and color information, we consistently improve both our quantitative and especially our qualitative results.

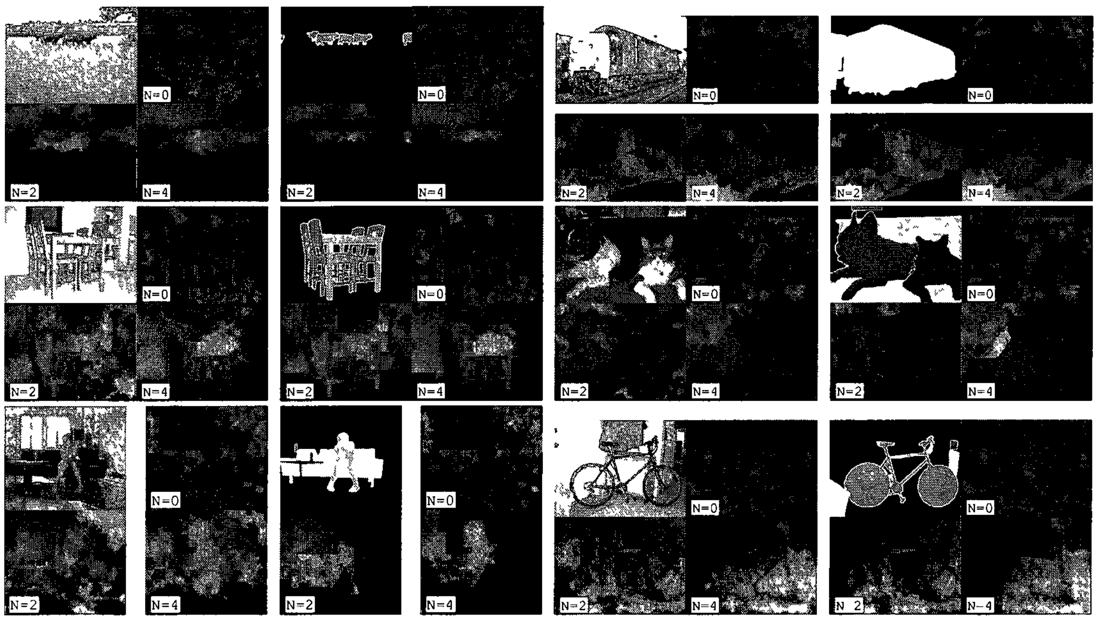


Figure 5.3: **PASCAL VOC 2007 + CRF.** Some selected segmentations for PASCAL. For each test image, the results are arranged into two blocks of four images. The first block (left-to-right) shows the results of the superpixel neighborhoods without a CRF. The second block uses the CRF described in section 5.3.4. Colors indicate category and the intensity of the color is proportional to the posterior probability of the classification.

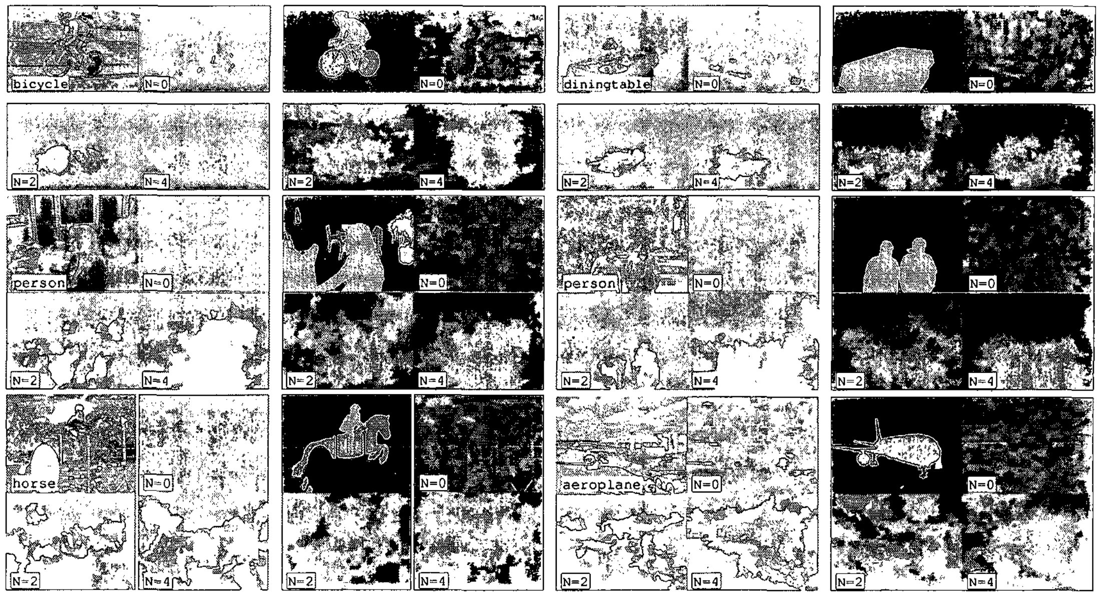


Figure 5.4: PASCAL VOC 2007 Confidence. Confidence maps for PASCAL. The results are arranged into two blocks of four images for each test image. The first block contains the input image, a category label, and the confidence map for that category for $N = 0, 2, 4$. The second block contains the ground truth labeling and our labellings with an intensity proportional to the confidence of the classification. Colors indicate category. For example, in the upper left we show the confidence for bicycle, and the classification which contains mostly bicycle (green) and some motorbike (light blue).

	background	aeroplane	bicycle	bird	boat	bottle	bus	car	cat	chair	cow	diningtable	dog	horse	motorbike	person	pottedplant	sheep	sofa	train	tvmonitor	**Average**
Brookes	**78**	6	0	0	0	0	9	5	10	1	2	11	0	6	6	29	2	2	0	11	1	9
[PSH08]	59	27	1	8	2	1	32	14	14	4	8	32	9	24	15	**81**	11	**26**	1	28	17	20
[SJC08]	33	46	5	14	11	14	34	8	6	3	10	39	40	28	23	32	19	19	8	24	9	20
[SJC08] + ILP	20	**66**	6	15	6	**15**	32	19	7	7	**13**	**44**	31	**44**	27	39	**35**	12	7	39	23	24
$N = 0$	21	14	8	8	**17**	14	10	7	19	13	**13**	7	16	9	13	2	10	23	34	17	20	14
CRF+$N = 0$	20	14	8	8	**17**	14	10	7	19	13	**13**	7	16	9	13	2	10	23	**34**	17	20	14
$N = 1$	27	27	20	17	14	12	18	11	37	18	7	14	26	19	35	18	13	21	25	31	25	21
CRF+$N = 1$	38	32	20	13	**17**	10	20	11	52	17	7	14	31	21	39	28	14	12	28	42	33	24
$N = 2$	36	27	26	15	11	5	26	29	42	**25**	9	15	36	23	58	32	17	11	20	37	29	25
CRF+$N = 2$	56	26	29	19	16	3	**42**	**44**	56	23	6	11	**62**	16	68	46	16	10	21	**52**	**40**	**32**
$N = 3$	47	22	24	17	11	6	35	25	46	19	8	19	33	29	62	47	16	20	26	37	29	28
CRF+$N = 3$	65	22	28	**32**	2	4	40	30	**61**	10	3	20	35	24	**72**	62	16	23	20	44	30	30
$N = 4$	51	20	22	18	7	2	39	25	49	15	6	14	36	28	64	56	15	17	21	40	23	27
CRF+$N = 4$	65	20	**30**	22	2	2	39	25	57	10	3	7	36	23	66	62	15	17	8	46	11	27

Table 5.3: **VOC 2007 segmentation results.** Our best overall average performance (CRF+N = 2) performs better than Shotton *et al.* [SJC08] with or without an Image Level Prior (ILP) on 14 out of 21 categories. Note that we could add ILP to our model. Similarly, we do not compare with the Shotton *et al.* results which used TKK's detection results as a Detection Level Prior (DLP) because TKK's detections were not available. We expect our method would provide a similar performance boost with this information. The CRF provides small but noticeable improvements on for all values of N.

CHAPTER 6

Conclusion

In this dissertation, we have proposed methods for improving image categorization, and have focused on those that lend themselves to the task of class segmentation. We briefly review each of the chapters in turn.

Stability in feature detection. Chapter 2 examines nuisances and shows that because of the quantization group in feature detection we can only canonize translation and rotation, and not the affine group or scale. We propose a detector which uses stable nodes in segmentation trees to define regions, and show that on Caltech-101 this detector is more effective than SIFT.

Categorization with segmentation. Chapter 3 proposes a method to split SIFT descriptors along segmentation boundaries. We find that knowing where an object is can help in determining what the object is, even for relatively aligned and uncluttered datasets like Caltech 101.

Discriminative dictionaries and pixel categorization. Chapter 4 constructs a fast and compact dictionary using agglomerative information bottleneck that maintains and sometimes improves in classification performance compared to a larger generative dictionary. This discriminative "smart" dictionary is combined with fast dense SIFT features and integral histograms to create a system which categorizes a window around each pixel in an image.

Superpixel categorization. Chapter 5 advocates the use of superpixels as the basic unit of a class segmentation scheme. It constructs a classifier on *local neighborhoods* of superpixels and shows these neighborhoods have a regularizing effect on the classifier. A conditional random field constructed on the superpixel graph assigns the most probable class label to groups of superpixels in an image, penalizing adjacent superpixels which have the approximately the same color but not the same label.

Implementations of our work are provided in VLFeat [VF10] and Blocks ([FV09] and discussed further in Appendix B). In the appendix, we also discuss a GPU based implementation of quick shift, which may be used to speed up the work in Chapter 5.

APPENDIX A

Really Quick Shift

A.1 Introduction

Segmentation algorithms have played an important role in computer vision research, both as an end goal [SM00, CM02, VS08] and more recently as a preprocessing step for other domains, including stereo [LSY06] and category-level scene parsing [FVS09, GRC08]. Breaking the image into smaller components, often called superpixels, allows algorithms to consider the image in meaningful chunks, rather than at the lowest common denominator (pixels).

Unfortunately, algorithms developed for segmentation are often quite costly in both memory usage and computation. This bottleneck limits the scale of the applications and data that they can be applied to.

In this work, we show that a GPU implementation of quick shift [VS08] can improve the performance of an already (relatively) fast segmentation algorithm by 10X-50X, opening up a host of potential new applications such as scene understanding in videos, and improved real time video abstraction [WOG06].

A.2 Related Work

Most related work involving GPUs for segmentation is in the medical imaging domain, where the extra dimension of data (a volume instead of an image) has made speed a requirement rather than an option [SHN03, CLW04, LCW03, LM08]. One notable exception found outside of medical imaging is that of Catanzaro *et al.* [CSS09] who adapt a boundary detection technique (gPb [MAF08]) to the GPU. While gPb can be used for segmentation [AMF09], our exact implementation of quick shift is over ten times faster on similar hardware.

In recognition, GPU based feature detectors and trackers [SFP06, HMS07] have been proposed, as have learning components such as support vector machines [CSK08] and k-nearest neighbors [GDB08]. Recently, Wojek *et al.* [WDS08] even proposed a GPU accelerated sliding window categorization scheme.

Other recent successes in using GPUs for vision include general purpose libraries such as OpenVIDIA [FM05], and specific applications which are often centered around video such as motion detection [YM08] or particle filtering [MT08].

Carreira *et al.* [Car06] have done work on approximating Gaussian Mean Shift (GMS) by decreasing the number of iterations required by the algorithm and the cost per iteration (by approximating the density). We effectively circumvent the need to optimize the number of iterations since quick shift only requires one iteration. Instead of approximating the density, we simply exploit the parallelism of the density computation to achieve a speedup by using hardware suited for the task (a GPU). We note that we could also approximate the density as in [Car06], and that would result in further speedups.

A.3 Quick shift algorithm

Quick shift is a kernelized version of a mode seeking algorithm similar in concept to mean shift [CM02, FH75] or medoid shift [SKK07]. Given N data points $x_1, \dots, x_N$, it computes a Parzen density estimate around each point using, for example, an isotropic Gaussian window:

$$P(x) = \frac{1}{2\pi\sigma^2 N} \sum_{i=1}^{N} e^{\frac{-\|x - x_i\|^2}{2\sigma^2}}$$

Once the density estimate $P(x)$ has been computed, quick shift connects each point to the nearest point in the feature space which has a higher density estimate. Each connection has a distance d_x associated with it, and the set of connections for all pixels forms a tree, where the root of the tree is the point with the highest density estimate.

Quick shift may be used for any feature space, but for the purpose of this paper we restrict it to one we can use for image segmentation: the raw RGB values augmented with the (x, y) position in the image. So, the feature space is five dimensional: (r, g, b, x, y). To adjust the trade-off between the importance of the color and spatial components of the feature space, we simply pre-scale the (r, g, b) values by a parameter λ, which for these experiments we fix at $\lambda = 0.5$.

To obtain a segmentation from a tree of links formed by quick shift, we choose a threshold τ and break all links in the tree with $d_x > \tau$. The pixels which are a member of each resulting disconnected tree form each segment.

A.3.1 Segmentation specific optimizations

In the case where our feature space is restricted to contain components which are defined on the image plane, and our set of data points are the set of pixels, we can immediately put some useful bounds on both the density computation and the neighbor linking process.

First, when computing the energy we can restrict the domain of pixels we consider to a window which is less than 3σ pixels away. Beyond this the contribution to the density is guaranteed to be small. Second, when linking the neighbors, there is also a natural bound for the search window. Pixels which are further than τ away in the image plane must be at least that far away in the feature space. Conceptually we will talk about the density computation and linking process as separate components of the algorithm because one (the density computation) must precede the other, and they operate on different domains of data. A pseudo-code implementation is shown in Figure A.2, and some segmentations with various parameters are shown in Figure A.1.

A.4 Quick shift on a GPU

Because quick shift operates on each pixel of an image, and the computation which takes place at each pixel is independent of its distant surroundings, it is a good candidate for implementation on a parallel architecture.

We use CUDA 3.0 to develop a first implementation which simply copies the image to the device and breaks the computation of the density and the neighbors into blocks for the GPU to process.

Although this is faster than the CPU version, the bottleneck is clearly memory latency. Global memory on GPUs is slow, requiring hundreds of cycles to access, and for each pixel quick shift needs to access ceil$((6 * \sigma)^2)$ neighbors.

To address this, one option is to load an *apron* of pixels surrounding the block being computed into shared memory, so that when an element of the block computes its similarity with a pixel outside of the block, the memory access is cached. However, because this operation is not easily separable, the shared memory requirement scales quadratically with sigma. Even modest values of sigma will quickly exhaust the 16000 bytes of shared memory available on modern

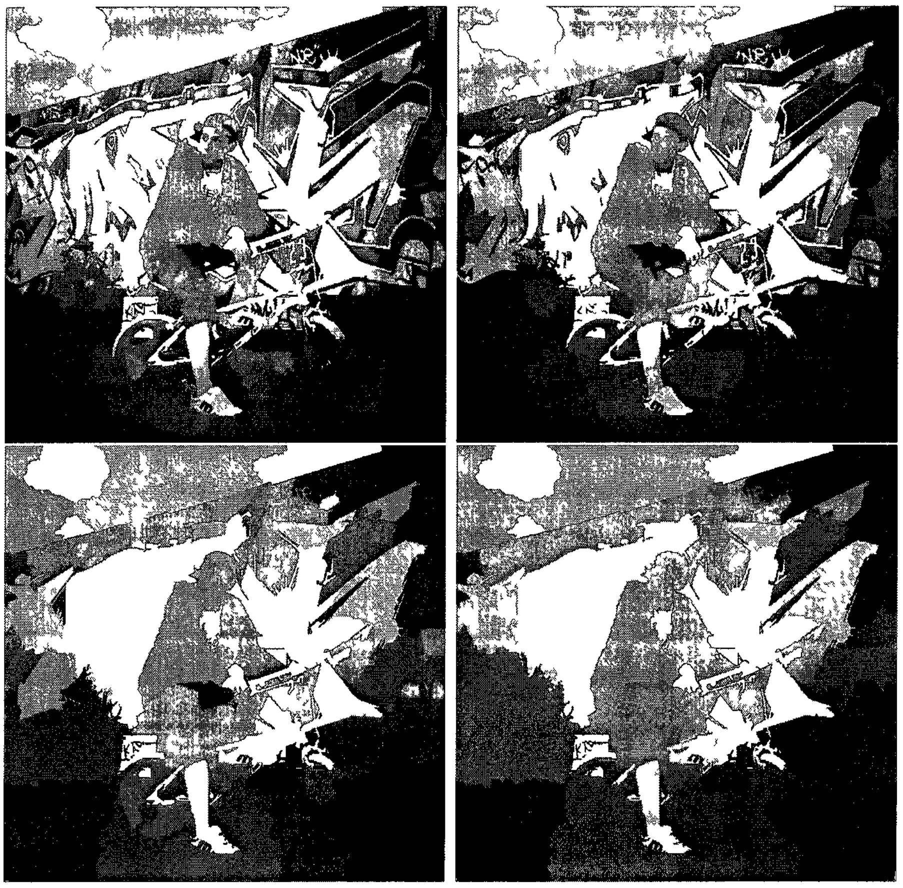


Figure A 1 Sample quick shift results. Increasing σ smoothes the underlying estimate of the density, providing fewer modes Increasing τ increases the average size of a region as well as the error in the distance estimate The top row of images have $\sigma = 2$, the bottom row $\sigma = 10$ The left column has $\tau = 10$ and the right $\tau = 20$

```
function computeDensity()
for x in all pixels
  P[x] = 0
  for n in all pixels less than 3*sigma away
    P[x] += exp(-(f[x]-f[n])^2 / (2*sigma*sigma))

function linkNeighbors()
for x in all pixels
  for n in all pixels less than tau away
    if P[n] > P[x] and distance(x,n) is smallest among all n
      d[x] = distance(x,n)
      parent[x] = n
```

Figure A.2: **Quick shift image segmentation in pseudo-code.** The algorithm proceeds in two steps. First it iterates over the image creating a Parzen estimate of the density at each pixel. Then, it links each pixel to the nearest pixel (in the feature space) which increases the estimate of the density.

Figure A.3: **Evaluation images.** Four images from PASCAL-2007 used to evaluate the speed of the proposed algorithm.

GPUs

Instead, we map the image and the estimate of the density to a 3D and 2D texture, respectively. We have good locality of access because each thread accesses a block of pixels around it. The results based of this texture cached approach are labeled with a "Tex" suffix in the next section.

A.5 Evaluation

There are two aspects of the algorithm to evaluate: the correctness and the time required. To confirm the correctness of the GPU implementation, we compare the energy and segmentation to the one returned by the publicly available implementation of quick shift in VLFeat [VF08].

To measure the speed of the algorithm, we pick a few random images from the PASCAL-2007 dataset (shown in Figure A.3). The images are cropped and up-sampled to 1024x1024. All reported performance numbers are obtained by averaging the results from all of the images.

We explore the effect of each parameter which changes the runtime of the algorithm. First, in Figure A.4 we show the performance of the algorithms as the resolution of the image is increased while keeping σ and τ fixed. Next, in Figure A.5 we keep the resolution fixed at 512x512, fix τ, and adjust σ, showing how it affects the runtime of just the density computation part of the algorithm. Finally, Figure A.6 keeps both the resolution and σ fixed and instead adjusts τ, showing the time required to link the neighbors.

Hardware. The CPU ground truth version is evaluated on a 2.4Ghz Core 2 Duo. We show results for two GPUs: a laptop board (GeForce 8600M GT), and a mid-range desktop card (GeForce 9800 GT). The 8600M GT has 4 multiprocessors, 32 cores, and a core clock speed of 475MHz. The 9800 GT has 14 multiprocessors, 112 cores, a 550MHz core clock speed. Due to limits on the runtime of CUDA kernels on the 8600M, in Figures A.5 and A.6 results are not reported for the slowest running case because the kernel was stopped before completion. We note that while newer hardware (such as cards based on the recently released FERMI architecture) would undoubtedly be faster, we want to show

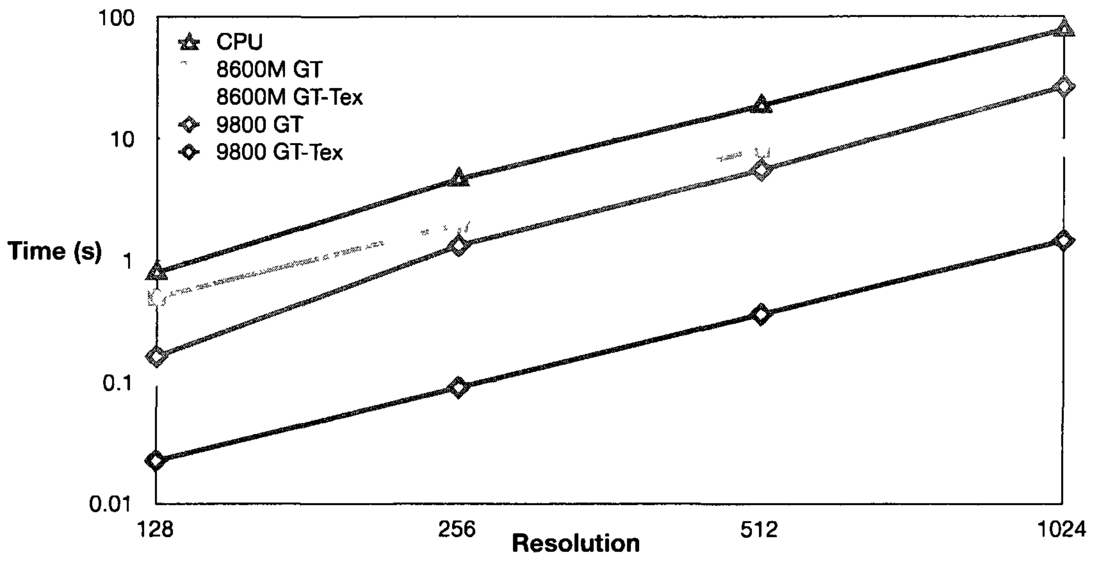


Figure A.4: **Quick shift CPU vs GPU.** The graph shows the time required on two different GPUs as the resolution of the image is increased. Results are averaged over the four images from PASCAL-2007 shown in Figure A.3. For this data, $\sigma = 6$ and $\tau = 10$. At 1024x1024, the speedup compared to the CPU version is 54X.

what is possible with only limited hardware investment.

For both GPUs evaluated we use a block size of 16x16, even though it has been shown that tuning the block size for a particular GPU can provide a boost in performance.

Our complete source code as well as precompiled binaries for major architectures are available on our website at `http://vision.ucla.edu/~brian/qsgpu`.

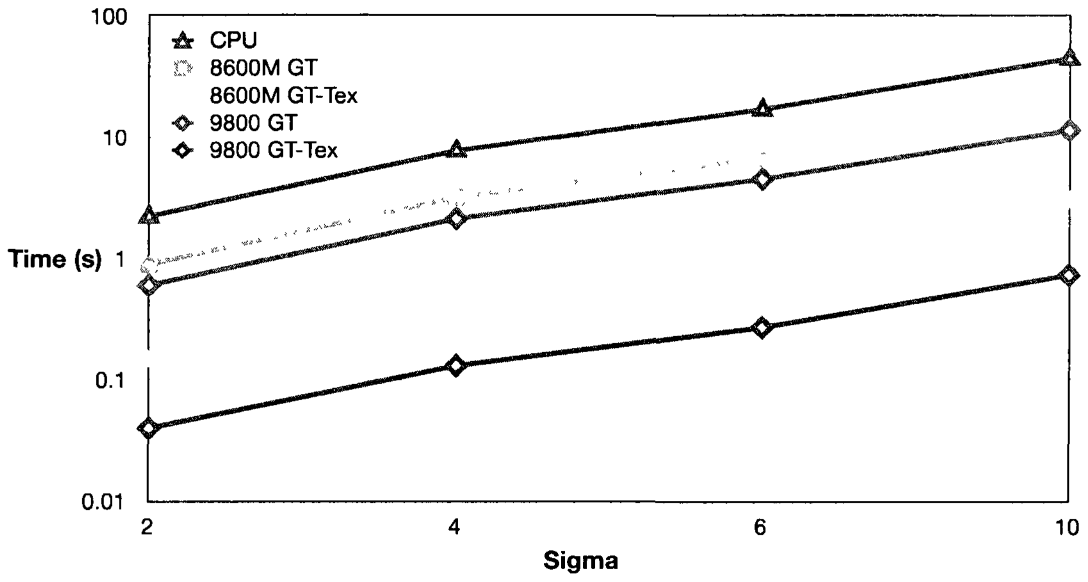


Figure A.5: **Effect of σ on density computation time.** As in Figure A.4, we show that as σ is increased, processing time is increased and the texture memory-backed GPU version remains the most efficient option. Here we fix $\tau = 10$ and the image resolution to 512x512. Results are averaged over the same four images as before.

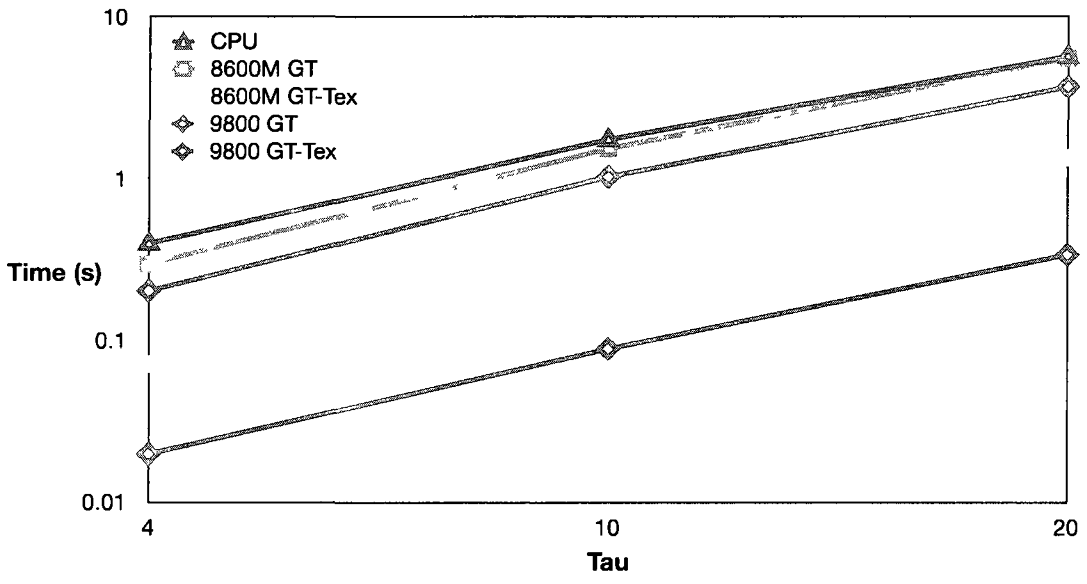


Figure A.6: **Effect of τ on neighbor linking time.** We show that as τ is increased, the amount of time required for finding the nearest neighbor which increases the density estimate is naturally increased. Here we fix $\sigma = 6$ and the image resolution to 512x512. Results are averaged over the same four images as before.

A.6 Conclusion

We have shown a GPU implementation of quick shift which provides a 10 to 50 times speedup over the CPU implementation, resulting in a superpixelization algorithm which can run at 10Hz on 256x256 images. The implementation is an exact copy of quick shift, and could be further speeded up by approximating the density, via subsampling or other methods. It is likely that the implementation would also present similar speedups for exact mean shift.

APPENDIX B

Blocks - An Open Source Experiment Framework

B.1 Introduction

In 1995 when the web was still in its infancy, Buckheit and Donoho [BD95] introduced a MATLAB toolbox called WaveLab. This key insight of the paper associated with this toolbox is:

> An article about computational science in a scientific publication is not the scholarship itself, it is merely advertising of the scholarship. The actual scholarship is the complete software development environment and the complete set of instructions which generated the figures.
> - Buckheit and Donoho [BD95]

In other words: The software which reproduces the results is more important than the paper which reports them. In computer vision research, reproducibility is crucial. Algorithms developed for the field are often evaluated on one or more datasets and judged by their performance. Such an algorithm usually has a number of adjustable parameters, and the datasets are large, sometimes taking days or even weeks to process. Furthermore, as new datasets emerge, without implementations of previous work it becomes impossible to compare with algorithms that already exist on new data.

We have constructed a MATLAB experiment framework called Blocks [FV09] which follows this philosophy. Blocks is a BSD licensed open source framework which makes it easier for researchers to produce experimental results that are reproducible. It does this by compartmentalizing reusable pieces of an algorithm into "blocks" and facilitating the evaluation of these blocks on a dataset.

B.1.1 Related work

Peng *et al.* [PDZ06] surveyed papers in epidemiology published in the first half of 2005 and found that the vast majority of these papers were not readily reproducible, either because the data was not released or the methods for processing the data were not described.

Gentleman *et al.* [GT07] propose using a compendium as a container for the text of the research as well as the software, and advocate using "literate data analysis" [Lei02] to include pieces of code which generate figures directly in the document which presents them.

Stodden [Sto09] explore how the issue of reproducibility should interact with open source licenses, and proposed the Reproducible Research Standard (RRS). The standard proposes that code should be released under a permissive license which requires attribution (e.g. the modified BSD license) and data and figures should use the creative commons BY license.

Sharing code has become even more important as systems (especially in computer vision) are often constructed by assembling many components which are non-trivial to implement. The machine learning community has in recent years recognized this and a journal and companion website have been dedicated to facilitating the release of open source machine learning packages by its members [SBO07].

We have released a software package called VLFeat [VF10] which complements this work, providing the underlying pieces that are used to do fundamental vision tasks such as feature detection and clustering.

B.2 Basic Structure

A block is a piece of MATLAB code which has inputs (both parameters and pointers to other blocks), outputs which can be retrieved by name, and a function body to execute when the inputs have changed. Blocks are identified by their *tag* and *type* and stored in a folder with the same name as their tag located in a directory specified by the `prefix` member of the global `wrd`.

Code involving blocks requires (at minimum) three functions: `bkinit`, `bkbegin`, and `bkend`. In the hello world example in Figure B.2, we create a block of type helloworld with `bkinit`, then begin the block with `bkbegin`. At this point, if the block is already up to date, we do nothing else. However, if it isn't, we perform some computation (say hello) and close the block, saving a data structure to the folder at `hwtest/helloworld@default`.

By convention, we typically wrap `bkbegin` and `bkinit` in a block function. These functions start with `block_`, and usually perform one task in a pipeline (for example, extracting features). A very simple block function is shown in Figure B.2.

B.3 Building with Blocks

In Section B.2, we showed a basic complete block. Here we will explore block functionality in more detail. A basic block consists of a structure with members

```
% Define the location where results should be saved
global wrd;
wrd.prefix = 'hwtest';

% Initialize and run a block of type helloworld.
bk = bkinit('helloworld');
bk.tag = 'helloworld@default';
[bk dirty] = bkbegin(bk);
if dirty
  % Do some work
  fprintf('Hello world!\n')
  bk = bkend(bk);
end
```

Figure B.1: **Hello world in block form.** This code listing shows a minimal block which prints “Hello world!” the first time it is executed.

```
function bk = block_test(bk, varargin)

if nargin == 0
  bk = bkinit('test') ; bk.imsize = 50; bk.fetch = @fetch__ ;
  return ;
end
global wrd;
[bk, dirty] = bkbegin(bk) ;
if ~ dirty, return ; end
output = vl_dsift(single(rand(bk.imsize)));
save(fullfile(wrd.prefix, bk.tag, 'test.mat'), 'output', '-MAT');
bk = bkend(bk) ;

% Define a function for retrieval of data from the block
function varargout = fetch__(bk, what, varargin)
global wrd;
switch lower(what)
case 'output'
  path = fullfile(wrd.prefix, bk.tag, 'test.mat') ;
  data = load(path, '-MAT') ;
  varargout{1} = data.output ;
otherwise
  error('block_test: Attempted to fetch unknown type');
end
```

Figure B.2: **A simple feature extraction block.** This block extracts SIFT features on a random image of size `bk.imsize`.

corresponding to parameters, pointers to inputs, and the block's tag and type.

B.3.1 Parameters

Since each block is implemented as a function, it would be natural to assume that parameters might be passed directly to the function. However, one of the main goals of the framework is to intelligently check when parameters have changed, and in order to do that we must keep a record of them. This record is more naturally stored in the structure containing the block. So instead of:

```
result = block_test(parameter)
```

We have:

```
block = block_test();
block.parameter = parameter;
block = block_test(block);
```

The first call to `block_test` in this example initializes the block with its default parameters. When we call `block_test` we pass the already initialized block, which causes the function to determine if there is a need to execute the function. The function will be evaluated if any of the parameters have changed since its last execution, or if any of its inputs have been updated since its last run. Notice that in the above example, we do not return a result, but instead the updated block. Results can be retrieved by querying the block, which we describe in the next section.

B.3.2 Inputs and Outputs

In addition to parameters, blocks may have inputs which are pointers to other blocks on which they depend. This concept of dependency allows the blocks to intelligently update themselves when an input they depend on has been updated. For example, in a bag of features application, new histograms must be computed when new features are extracted.

Inputs are managed with `bkplug` and `bkfetch`. `bkplug` connects one block to another as in:

```
ex.classify = bkplug(ex.classify, 'hist', ex.hist.tag);
```

This connects a block structure representing the histograms to a block which handles classification. `bkplug` returns the modified block structure, which now contains a field called "hist" containing a pointer to the input tag. We can also ensure that required inputs are present using `bkinit` inside the definition of the block. To ensure that a block of type "feat" has an input named "db", we specify:

```
bk = bkinit('feat', 'db') ;
```

Once a block has finished executing, its results are stored to disk. To retrieve them, we use `bkfetch`. `bkfetch` takes an already initialized block or tag specifying a block and optional arguments and returns the relevant data. Because it internally calls a "fetch" function associated with the block, the actual means of retrieving the data can be as simple or complex as required. For example, in returning features in a block, we can either internally store the descriptors for each image to disk and load them when requested, or we can load the image and calculate the features each time we need them. This process is opaque to the user of the block.

Blocks are intended to be easily interchangeable. Blocks are not objects, but as long as two blocks implement the same interface, they can be interchanged transparently. The compressed dictionary block which implements the agglomerative information bottleneck dictionary discussed in Chapter 4 has the same fetch-able outputs as a k-means dictionary, so we do not have to modify any of the subsequent steps.

B.4 Sample Applications

While the architecture used is applicable to any problem which needs to avoid repeated computation and keep track of parameters, we include blocks which combine to form reference implementations of a bag of features classifier as well as the work described in Chapters 4 and 5.

In Figure B.3 we show the blocks required for a simple bag of features classifier. `block_db` builds a database from a particular dataset, and abstracts the retrieval of images from the dataset away from the user. `block_dbpart` implements the interface of `block_db` and additionally adds a partition of the dataset. In some datasets, this partition is fixed, and for others it is generated randomly. `block_feat` extracts SIFT descriptors and `block_dictionary` constructs a dictionary based on them. This dictionary is used to create histograms in `block_hist` and then a kernel is constructed between these histograms with `block_ker`. SVM training and testing are handled in separate blocks (`block_train_svm` and `block_test_svm` respectively). Finally, `block_vis` visualizes the results of a simple classification problem.

Figure B.4 shows blocks required to construct the class segmentation system described in Chapter 5. Blocks with the same name have the same functionality

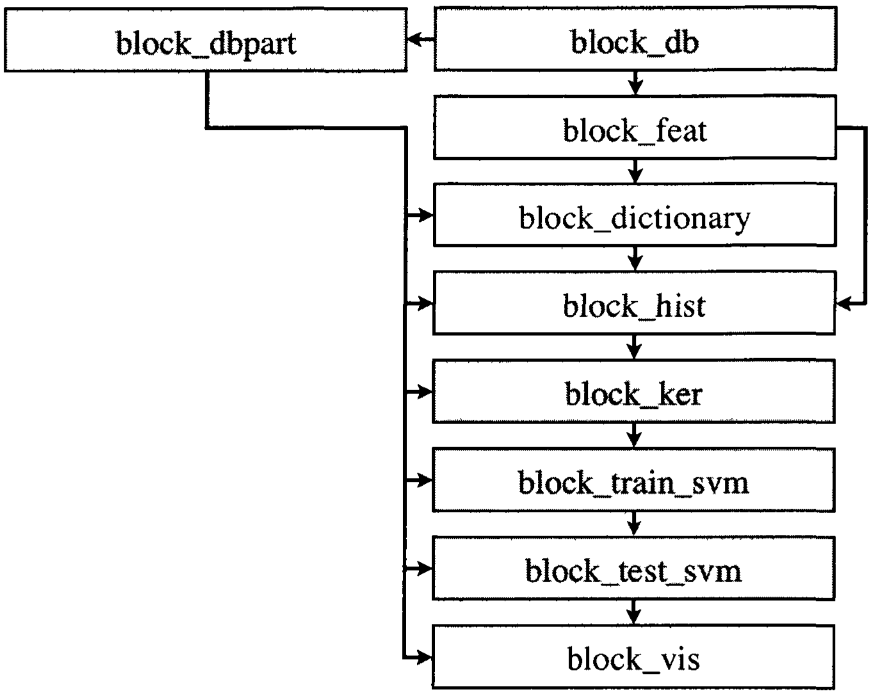


Figure B.3: **Bag of features in blocks.** The blocks used to execute a traditional bag of features pipeline. Blocks are color coded as in Figure 1.1. Blue colored boxes deal with datasets, tan compute features, yellow involve dictionary construction, green make histograms, and orange denote the classifier. Arrows indicate the direction of data flow, and the dependencies between blocks.

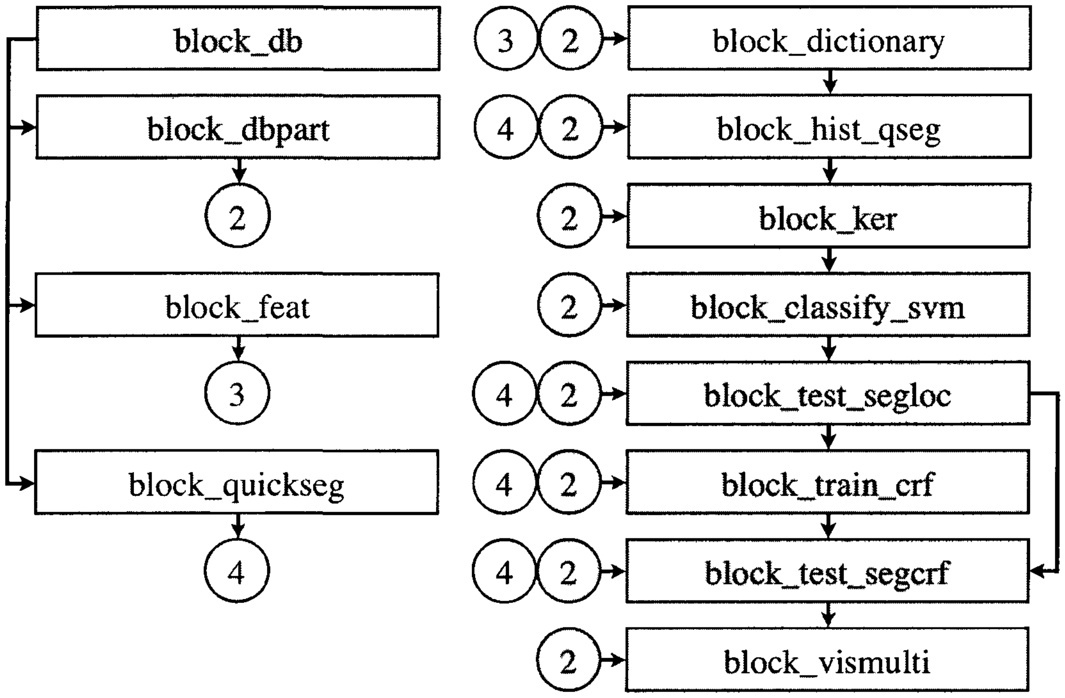


Figure B.4: **Superpixel neighborhoods in blocks.** The blocks used to create the system described in Chapter 5. The color coding is the same as Figure B.3. Notice that many of the same blocks are shared between the two examples, only when there is something specific to the application (blocks that deal with superpixels) are new blocks introduced.

as they do in the bag of features example (Figure B.3). `block_quickseg` builds a quick shift superpixelization of each image in the database. `block_hist_qseg` constructs histograms of superpixel neighborhoods. `block_classify_svm` builds a complete classifier on superpixels which `block_test_segloc` uses to score images from the database. `block_train_crf` trains a Conditional Random Field (CRF) which is applied to the data in `block_test_segcrf`. The results are reported by `block_vismulti`, which generates a report of the classification performance using various metrics.

B.5 Conclusion

We have shown a simple open source framework which allows us to easily create reproducible experiments in MATLAB. It does this by breaking computation into modular pieces and storing the intermediate results and the parameters which generated them. It intelligently recomputes blocks when the input to the block has changed. The modular components are easily interchanged, allowing evaluations which isolate and evaluate one component (e.g. the dictionary in a bag of features pipeline).

References

[AAB07] Emanuel Aldea, Jamal Atif, and Isabelle Bloch. "Image Classification Using Marginalized Kernels for Graphs." In *Proc. CVPR*, 2007.

[AMF09] P. Arbeláez, M. Maire, C. Fowlkes, and J. Malik. "From contours to regions: An empirical evaluation." In *Proc. CVPR*, 2009.

[AT05] A. Agarwal and B. Triggs. "Hyperfeatures - Multilevel Local Coding for Visual Recognition." Technical report, INRIA, 2005.

[BD95] J.B. Buckheit and D.L. Donoho. "Wavelab and reproducible research." *Wavelets and statistics*, p. 55, 1995.

[BDG03] Kobus Barnard, Pinar Duygulu, Raghavendra Guru, Prasad Gabbur, and David Forsyth. "The effects of segmentation and feature choice in a translation model of object recognition." In *Proc. CVPR*, 2003.

[BJ01] Y. Y. Boykov and M.-P. Jolly. "Interactive Graph Cuts for Optimality Boundary & Region Segmentation of Objects in N-D Images." In *Proc. ICCV*, 2001.

[BK04] Yuri Boykov and Vladimir Kolmogorov. "An Experimental Comparison of Min-Cut/Max-Flow Algorithms for Energy Minimization in Vision." In *PAMI*, 2004.

[BL08] M. Blaschko and C. Lampert. "Learning to Localize Objects with Structured Output Regression." In *Proc. ECCV*, 2008.

[BTG06] H. Bay, T. Tuytelaars, and L. Van Gool. "Surf: Speeded up robust features." In *Proc. ECCV*, 2006.

[BVZ01] Yuri Boykov, Olga Veksler, and Ramin Zabih. "Efficient Approximate Energy Minimization via Graph Cuts." In *PAMI*, 2001.

[Car06] MA Carreira-Perpinán. "Acceleration strategies for Gaussian mean-shift image segmentation." In *Proc. CVPR*, 2006.

[CDD04] G. Csurka, C. R. Dance, L. Dan, J. Willamowski, and C. Bray. "Visual Categorization with Bags of Keypoints." In *Proc. ECCV*, 2004.

[CF07] L. Cao and L. Fei-Fei. "Spatially coherent latent topic model for concurrent object segmentation and classification." In *Proc. ICCV*, 2007.

[CL01] Chih-Chung Chang and Chih-Jen Lin. *LIBSVM: a library for support vector machines*, 2001. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

[CLW04] J.E. Cates, A.E. Lefohn, and R.T. Whitaker. "GIST: an interactive, GPU-based level set segmentation tool for 3D medical images." *Medical Image Analysis*, **8**(3):217–231, 2004.

[CM02] D. Comaniciu and P. Meer. "Mean Shift: A Robust Approach Toward Feature Space Analysis." *PAMI*, **24**(5), 2002.

[CS10] J. Carreira and C. Sminchisescu. "Constrained Parametric Min-Cuts for Automatic Object Segmentation." In *Proc. CVPR*, 2010.

[CSK08] B. Catanzaro, N. Sundaram, and K. Keutzer. "Fast support vector machine training and classification on graphics processors." In *Proceedings of the 25th international conference on Machine learning*, pp. 104–111. ACM, 2008.

[CSS09] B. Catanzaro, B.Y. Su, N. Sundaram, Y. Lee, M. Murphy, and K. Keutzer. "Efficient, high-quality image contour detection." In *Proc. ICCV*, 2009.

[CV01] Tony Chan and Vese. "Active contours without edges." *IEEE Transactions on Image Processing*, **10**(2):266–277, feb 2001.

[DT05] N. Dalal and B. Triggs. "Histograms of Oriented Gradients for Human Detection." In *Proc. CVPR*, 2005.

[EHD00] Ahmed Elgammal, David Harwood, and Larry Davis. "Non-parametric Model for Background Subtraction." In *Proc. ECCV*, pp. 751–767, 2000.

[ELZ02] H. Edelsbrunner, D. Letscher, and A. Zomorodian. "Topological persistence and simplification." *Discrete & Computational Geometry*, **28**(4):511–533, 2002.

[EVW] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. "The PASCAL Visual Object Classes Challenge 2007 (VOC2007) Results." http://www.pascal-network.org/challenges/VOC/voc2007/workshop/index.html.

[FFP04] L. Fei-Fei, R. Fergus, and P. Perona. "Learning Generative Visual Models from Few Training Examples: An Incremental Bayesian Approach Tested on 101 Object Categories." In *CVPR Workshop*, 2004.

[FH75] K. Fukunaga and L. D. Hostler. "The Estimation of the Gradient of a Density Function, with Applications in Pattern Recognition." *IEEE Trans. on Information Theory*, **21**(1), 1975.

[FH04] P. F. Felzenszwalb and D. P. Huttenlocher. "Efficient Graph-Based Image Segmentation." *IJCV*, **59**(2), 2004.

[FM05] J. Fung and S. Mann. "OpenVIDIA: parallel GPU computer vision." In *Proceedings of the 13th annual ACM international conference on Multimedia*, p. 852, 2005.

[FMR07] Pedro Felzenszwalb, David McAllester, and Deva Ramanan. "A Discriminatively Trained, Multiscale, Deformable Part Model." `http://people.cs.uchicago.edu/~pff/papers/`, 2007.

[FV09] B. Fulkerson and A. Vedaldi. "Blocks: A MATLAB Experiment Framework." `http://www.vlblocks.org/`, 2009.

[FVS08] B. Fulkerson, A. Vedaldi, and S. Soatto. "Localizing Objects With Smart Dictionaries." In *Proc. ECCV*, 2008.

[FVS09] B. Fulkerson, A. Vedaldi, and S. Soatto. "Class Segmentation and Object Localization with Superpixel Neighborhoods." In *Proc. ICCV*, 2009.

[GBR08] C. Galleguillos, B. Babenko, A. Rabinovich, and S. Belongie. "Weakly Supervised Object Localization with Stable Segmentations." In *Proc. ECCV*, 2008.

[GD06] K. Grauman and T. Darrell. "Pyramid Match Kernels: Discriminative Classification with Sets of Image Features." Technical Report MIT-CSAIL-TR-2006-020, MIT, 2006.

[GDB08] V. Garcia, E. Debreuve, and M. Barlaud. "Fast k nearest neighbor search using gpu." In *Workshop on Computer Vision using GPUs*, 2008.

[GP74] V. Guillemin and A. Pollack. *Differential Topology.* Prentice-Hall, 1974.

[GRC08] Stephen Gould, Jim Rodgers, David Cohen, Gal Elidan, and Daphne Koller. "Multi-Class Segmentation with Relative Location Prior." In *IJCV*, 2008.

[GT07] R. Gentleman and D. Temple Lang. "Statistical analyses and reproducible research." *Journal of Computational and Graphical Statistics*, **16**(1):1–23, 2007.

[HB07] Zaïd Harchaoui and Francis Bach. "Image Classification with Segmentation Graph Kernels." In *Proc. CVPR*, 2007.

[HMS07] S. Heymann, K. Maller, A. Smolic, B. Froehlich, and T. Wiegand. "SIFT implementation and optimization for general-purpose GPU." In *Proc. WSCG*, 2007.

[HZC04] X. He, R. Zemel, and M. Carreira-Perpinán. "Multiscale Conditional Random Fields for Image Labeling." In *Proc. CVPR*, 2004.

[HZR06] X. He, R. Zemel, and D. Ray. "Learning and Incorporating Top-Down Cues in Image Segmentation." In *Proc. ECCV*, 2006.

[KH05] Sanjiv Kumar and Martial Hebert. "A Hierachical Field Framework for Unified Context-Based Classification." In *Proc. ICCV*, 2005.

[Kir96] A. Kirsch. "An Introduction to the Mathematical Theory of Inverse Problems." *Springer-Verlag, New York*, 1996.

[KZ04] Vladimir Kolmogorov and Ramin Zabih. "What Energy Functions can be Minimized via Graph Cuts?" In *PAMI*, 2004.

[LBH08] C. Lampert, M. Blaschko, and T. Hofmann. "Beyond Sliding Windows: Object Localization by Efficient Subwindow Search." In *Proc. CVPR*, 2008.

[LCS10] F. Li, J. Carreira, and C. Sminchisescu. "Object Recognition as Ranking Holistic Figure-Ground Hypotheses." In *Proc. CVPR*, 2010.

[LCW03] A. Lefohn, J. Cates, and R. Whitaker. "Interactive, gpu-based level sets for 3d segmentation." *Medical Image Computing and Computer-Assisted Intervention-MICCAI 2003*, pp. 564–572, 2003.

[Lei02] F. Leisch. "Sweave: Dynamic generation of statistical reports using literate data analysis." In *Compstat*, pp. 575–580, 2002.

[LHS07] M. Leordeanu, M. Hebert, and R. Sukthankar. "Beyond Local Appearance: Category Recognition from Pairwise Interactions of Simple Features." In *Proc. CVPR*, 2007.

[LLS04] B. Leibe, A. Leonardis, and B. Schiele. "Combined Object Categorization and Segmentation with Implicit Shape Model." In *ECCV Workshop on Statistical Learning in Comp. Vision*, 2004.

[LM08] Yuping Lin and Gerard Medioni. "Mutual Information Computation and Maximization Using GPU." In *Workshop on Computer Vision using GPUs*, 2008.

[LMP01] John Lafferty, Andrew McCallum, and Fernando Pereira. "Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data." In *Proc. ICML*, 2001.

[LMS06] B. Leibe, K. Micolajckzyk, and B. Schiele. "Efficient clustering and matching for object class recognition." In *Proc. BMVC*, 2006.

[Low99] D. G. Lowe. "Object Recognition from Local Scale-Invariant Features." In *Proc. ICCV*, 1999.

[Low04] D. G. Lowe. "Distinctive Image Features from Scale-Invariant Keypoints." *IJCV*, **2**(60):91–110, 2004.

[LR07] S. Lazebnik and M. Raginsky. "Learning Nearest-Neighbor Quantizers from Labeled Data by Information Loss Minimization." In *Proc. Conf. on Artificial Intellligence and Statistics*, 2007.

[LS07a] H. Ling and S. Soatto. "Proximity Distribution Kernels for Geometric Context in Category Recognition." In *Proc. CVPR*, 2007.

[LS07b] J. Liu and M. Shah. "Scene Modeling Using Co-Clustering." In *Proc. ICCV*, 2007.

[LSP06] S. Lazebnik, C. Schmid, and J. Ponce. "Beyond Bag of Features: Spatial Pyramid Matching for Recognizing Natural Scene Categories." In *Proc. CVPR*, 2006.

[LSY06] C. Lei, J. Selzer, and Y.H. Yang. "Region-tree based stereo using dynamic programming optimization." In *Proc. CVPR*, 2006.

[MAF08] M. Maire, P. Arbelaez, C. Fowlkes, and J. Malik. "Using Contours to Detect and Localize Junctions in Natural Images." In *Proc. CVPR*, 2008.

[MCU02] J. Matas, O. Chum, M. Urban, and T. Pajdla. "Robust Wide Baseline Stereo from Maximally Stable Extremal Regions." In *Proc. BMVC*, 2002.

[Mil69] J. Milnor. *Morse Theory.* Annals of Mathematics Studies no. 51. Princeton University Press, 1969.

[ML06] J. Mutch and D. G. Lowe. "Multiclass Object Recognition with Sparse, Localized Features." In *Proc. CVPR*, 2006.

[MPW08] A.P. Moore, S. Prince, J. Warrell, U. Mohammed, and G. Jones. "Superpixel Lattices." In *Proc. CVPR*, 2008.

[MS04] K. Mikolajczyk and C. Schmid. "Scale & affine invariant interest point detectors." *IJCV*, **11**(60):63–86, 2004.

[MS06] M. Marszałek and C. Schmid. "Spatial Weighting for Bag-of-Features." In *Proc. CVPR*, 2006.

[MS07] M. Marszałek and C. Schmid. "Accurate Object Localization with Shape Masks." In *Proc. CVPR*, 2007.

[MS08] J. Meltzer and S. Soatto. "Edge Descriptors for Robust Wide-Baseline Correspondence." In *Proc. ICCV*, 2008.

[MT08] Erik Murphy-Chutorian and Mohan M. Trivedi. "Particle Filtering with Rendered Models: A Two Pass Approach to Multi-object 3D Tracking with the GPU." In *Workshop on Computer Vision using GPUs*, 2008.

[MTJ06] Frank Moosmann, Bill Triggs, and Frederic Jurie. "Fast Discriminative Visual Codebooks using Randomized Clustering Forests." In *Proc. NIPS*, 2006.

[NS06] D. Nistér and H. Stewénius. "Scalable Recognition with a Vocabulary Tree." In *Proc. CVPR*, 2006.

[NTU07] Sebastian Nowozin, Koji Tsuda, Takeaki Uno, Taku Kudo, and Gökhan BakIr. "Weighted Substructure Mining for Image Analysis." In *Proc. CVPR*, 2007.

[OP05] A. Opelt and A. Pinz. "Object Localization with Boosting and Weak Supervision for Generic Object Recognition." In *Proc. SCIA*, 2005.

[PCI07] J. Philbin, O. Chum, M. Isard, J. Sivic, and A. Zisserman. "Object retrieval with large vocabularies and fast spatial matching." In *Proc. CVPR*, 2007.

[PDZ06] R.D. Peng, F. Dominici, and S.L. Zeger. "Reproducible epidemiologic research." *American Journal of Epidemiology*, **163**(9):783, 2006.

[Por05] F. Porikli. "Integral Histogram: A Fast Way to Extract Histograms in Cartesian Spaces." In *Proc. CVPR*, 2005.

[PS78] T. Poston and I. Stewart. *Catastrophe theory and its applications.* Pitman, London, 1978.

[PSH08] C. Pantofaru, C. Schmid, and M. Hebert. "Object Recognition by Integrating Multiple Image Segmentations." In *Proc. ECCV*, 2008.

[RES06] B. C. Russell, A. A. Efros, J. Sivic, W. T. Freeman, and A. Zisserman. "Using Multiple Segmentations to Discover Objects and their Extent in Image Collections." In *Proc. CVPR*, 2006.

[RM03] X. Ren and J. Malik. "Learning a Classification Model for Segmentation." In *Proc. ICCV*, 2003.

[Rob01] C. P. Robert. *The Bayesian Choice.* Springer Verlag, New York, 2001.

[Rud73] W. Rudin. *Functional Analysis.* Mc.Graw-Hill, 1973.

[RVG07] A. Rabinovich, A. Vedaldi, C. Galleguillos, E. Wiewiora, and S. Belongie. "Objects in Context." In *Proc. ICCV*, 2007.

[SBC05] J. Shotton, A. Blake, and R. Cipolla. "Contour-Based Learning for Object Detection." In *Proc. ICCV*, 2005.

[SBO07] S. Sonnenburg, M.L. Braun, C.S. Ong, S. Bengio, L. Bottou, G. Holmes, Y. LeCun, K.R. Müller, F.Pereira, C.E. Rasmussen, G. Rätsch, B. Schölkopf, A. Smola, P. Vincent, J. Weston, and R. Williamson. "The need for open source software in machine learning." *Journal of Machine Learning Research*, **8**:2443–2466, 2007.

[SFP06] S.N. Sinha, J.M. Frahm, M. Pollefeys, and Y. Genc. "GPU-based video feature tracking and matching." In *EDGE, Workshop on Edge Computing Using New Commodity Architectures*, volume 278. Citeseer, 2006.

[Sha98] J. Shao. *Mathematical Statistics.* Springer Verlag, 1998.

[SHN03] A. Sherbondy, M. Houston, and S. Napel. "Fast volume segmentation with simultaneous visualization using programmable graphics hardware." In *Proceedings of the 14th IEEE Visualization 2003 (VIS'03)*, p. 23. IEEE Computer Society, 2003.

[SJC08] J. Shotton, M. Johnson, and R. Cipolla. "Semantic Texton Forests for Image Categorization and Segmentation." In *Proc. CVPR*, 2008

[SKK07] Y. A. Sheikh, E. A. Khan, and T. Kanade. "Mode-seeking by medoid-shifts." In *Proc. CVPR*, 2007.

[Slo03] N. Slonim. "IBA_1.0 Matlab Code for Information Bottleneck Clustering Algorithms." `http://www.princeton.edu/~nslonim/`, 2003.

[SM00] J. Shi and J. Malik. "Normalized Cuts and Image Segmentation." *PAMI*, **22**(8):888, 2000.

[Soa09] S. Soatto. "Actionable Information in Vision." In *Proceedings of the International Conference on Computer Vision*, October 2009.

[SPV09] G. Sundaramoorthi, P. Petersen, V. S. Varadarajan, and S. Soatto. "On the set of images modulo viewpoint and contrast changes." In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, June 2009.

[ST99] N. Slonim and N. Tishby. "Agglomerative Information Bottleneck." In *Proc. NIPS*, 1999.

[Sto09] V. Stodden. "Enabling reproducible research: licensing for scientific innovation." *International Journal of Communications Law & Policy*, **13**:1, 2009.

[SWR06] J. Shotton, J. Winn, C. Rother, and A. Criminisi. "*TextonBoost*: Joint Appearance, Shape and Context Modeling for Multi-Class Object Recognition and Segmentation." In *Proc. ECCV*, 2006.

[TLF08] E. Tola, V. Lepetit, and P. Fua. "A Fast Local Descriptor for Dense Matching." In *Proc. CVPR*, 2008.

[TS07] T. Tuytelaars and C. Schmid. "Vector Quantizing Feature Space with a Regular Lattice." In *Proc. ICCV*, 2007.

[VF08] A. Vedaldi and B. Fulkerson. "VLFeat: An Open and Portable Library of Computer Vision Algorithms." `http://www.vlfeat.org/`, 2008.

[VF10] A. Vedaldi and B. Fulkerson. "VLFeat - An open and portable library of computer vision algorithms." In *Proceedings of the 18th annual ACM international conference on Multimedia*, October 2010.

[VJ01] P. Viola and M. Jones. "Robust Real-time Object Detection." In *Second International Workshop on Statistical and Computational Theories of Vision*, Vancouver, Canada, 2001.

[VS05] A. Vedaldi and S. Soatto. "Features for Recognition: Viewpoint Invariance for Non-Planar Scenes." In *Proc. ICCV*, 2005.

[VS08] A. Vedaldi and S. Soatto. "Quick Shift and Kernel Methods for Mode Seeking." In *Proc. ECCV*, 2008.

[VT07] J. Verbeek and B. Triggs. "Region Classification with Markov Field Aspect Models." In *Proc. CVPR*, 2007.

[VZ10] A. Vedaldi and A. Zisserman. "Efficient additive kernels via explicit feature maps." In *Proc. CVPR*, 2010.

[WCM05] J. Winn, A. Criminisi, and T. Minka. "Object Categorization by Learned Universal Visual Dictionary." In *Proc. ICCV*, 2005.

[WDS07] X. Wang, G. Doretto, T. Sebastian, J. Rittscher, and P. Tu. "Shape and Appearance Context Modeling." In *Proc. ICCV*, 2007.

[WDS08] C. Wojek, G. Dorkó, A. Schulz, and B. Schiele. "Sliding-windows for rapid object class localization: A parallel technique." *Pattern Recognition*, pp. 71–81, 2008.

[WOG06] H. Winnemoller, S.C. Olsen, and B. Gooch. "Real-time video abstraction." *ACM Transactions on Graphics (TOG)*, **25**(3):1226, 2006.

[YM08] Qian Yu and Gerard Medioni. "A GPU-based implementation of Motion Detection from a Moving Platform." In *Workshop on Computer Vision using GPUs*, 2008.

[ZBM06] H. Zhang, A. C. Berg, M. Maire, and J. Malik. "SVM-KNN: Discriminative Nearest Neighbor Classification for Visual Category Recognition." In *Proc. CVPR*, 2006.

[ZML06] J. Zhang, M. Marszalek, S. Lazebnik, and C. Schmid. "Local features and kernels for classification of texture and object categories: A comprehensive study." *IJCV*, 2006.