THE UNIVERSITY OF CHICAGO

FROM RIGID TEMPLATES TO GRAMMARS: OBJECT DETECTION WITH STRUCTURED MODELS

A DISSERTATION SUBMITTED TO
THE FACULTY OF THE DIVISION OF THE PHYSICAL SCIENCES
IN CANDIDACY FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

DEPARTMENT OF COMPUTER SCIENCE

BY
ROSS BROOK GIRSHICK

CHICAGO, ILLINOIS
JUNE 2012

To my parents.

In the computer field, the moment of truth is a running program; all else is prophecy.

– Herbert Simon

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

Foremost, I would like to thank my advisor Pedro Felzenszwalb. Pedro has shaped my perspective as a researcher in so many fundamental ways: the value I place on elegant models, crisp mathematical formulations, and on the important role of algorithms in vision research. Pedro cares about the big picture and the small details, offering rarely found guidance at both levels, while still allowing me freedom to explore on my own. It has been tremendous fun working together over the last five years! I will always remember hacking together in the early days of the PASCAL VOC 2008 Challenge.

I would also like to thank my committee members, David McAllester and Yali Amit. Yali's class on statistical computer vision was one of the most influential I've taken. I consider myself very fortunate to have been able to meet with David on a regular basis throughout this work, even in the midst of his always-hectic schedule. Learning from David's intuitive grasp on central issues in machine learning has been truly inspiring.

More broadly, the vision reading group at TTI-Chicago has been a constant source of stimulating discussions, and I've learned far more there about computer vision research "in the wild" than I could have from any single class. I appreciate the humor and insight of its rotating cast of participants.

Like many Ph.D.s, this one was fueled by a steady stream of caffeine. The excellent coffee roasters and shops in Chicago taught me what good coffee is and provided an ideal backdrop for so many hours of reading, reviewing, writing of papers and code, and — the true luxury of being a graduate student — just thinking.

And, for Meera, who has always helped me pursue my dreams and given me confidence. I so much enjoy her curiosity in my work, and her eagerness to learn about computer science in general.

This research has been supported by NSF grant IIS-0746569 and a generous scholarship from the ARCS Foundation.

ABSTRACT

We develop models for localizing instances of a generic object category, such as cars or people, in images. We define these models using a grammar formalism. In this formalism compositional rules are used to encode models that can range in complexity from simple rigid templates to rich deformable part models with variable structure. A central contribution of this dissertation is an exploration along this axis, wherein we gradually enrich our object category representations. We demonstrate that these richer models lead to improved object detection performance on challenging datasets such as the PASCAL VOC Challenges.

While building richer models, we would like to make use of existing training data and annotations. These annotations typically specify labels, such as object bounding boxes, that are "weak" compared to the derivation trees produced by detection with a grammar model. We propose a new discriminative training framework that directly supports learning models from weakly-labeled examples. We show how to apply this framework to the problem of learning the parameters of a grammar model. This approach results in a top-performing method for detecting people in images.

In order to achieve widespread use in research and applications, an object detection system must not only be accurate, but also fast. Along the line of efficient computation, we develop a technique for "compiling" one of our object models into a much faster detector that implements a cascade architecture. We show how to select the cascade thresholds in a way that is both safe and effective. We demonstrate that the cascaded detector produces detections 15x faster than the non-cascade approach with no loss in precision or recall.

CHAPTER 1

INTRODUCTION

1.1 Objective

A car that automatically stops before striking an unnoticed pedestrian; a camera that finds a face and brings it into focus without manual operation; a video search engine that sifts through surveillance footage for the moment when a motorcycle sped across a lonely intersection. These are only a few examples of the numerous applications that all require the algorithmic extraction of high-level information from images. The objective of this dissertation is to develop computational models and techniques that provide one of the most basic pieces of information needed by these applications: *What objects are where?*

To make this question crisp, we formulate it as the *category-level object detection task*. Following the prevailing convention [28]: the input to this task is a static image; the output is a list of (object category, location) pairs; and, given a predefined set of categories, the goal is to return a list that correctly categorizes all objects of interest in the input image and localizes each one with a tight bounding rectangle around its visible extent. Figure 1.1 shows some example input images and their corresponding ground-truth detections.

In some specific domains, where the category is restricted and the imaging conditions are somewhat controlled, object detection is largely solved. Frontal-face detection for autofocus in consumer digital cameras and pedestrian detection from a car are two applications where domain-specific methods have matured into robust technologies. In the general case, however, object detection remains largely unsolved.

As a proxy for "the general case" — which one might consider to be the task of recognizing an estimated 3,000 basic-level object categories [6] in any physically realizable image — we focus on the more modest PASCAL Visual Object Classes (VOC) Challenge: Detect objects from a diverse set of 20 visual[1] object categories in images sampled from flickr [28].

1. As opposed to functional object categories, where membership is defined by functional use rather than

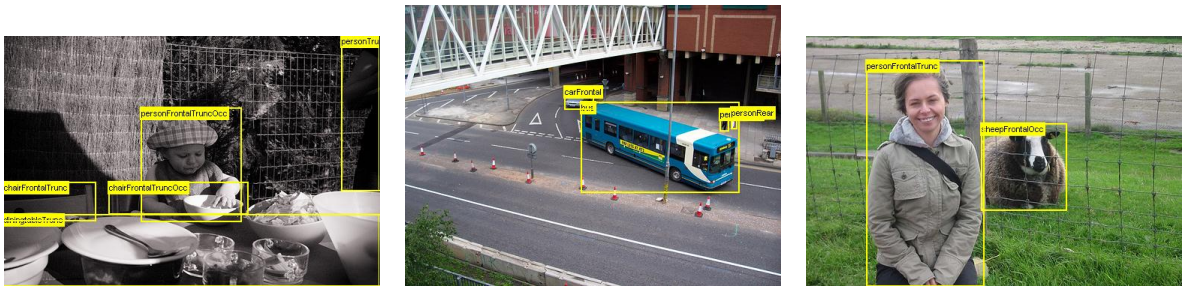


Figure 1.1: Example images, from the PASCAL VOC 2011 detection dataset [27], with overlaid ground-truth bounding boxes.

Even this "modest" proxy task is far from solved. In the 2011 Challenge, the maximum mean average precision (AP) achieved by a single system, averaged over all 20 categories, was about 40%. The maximum per-category AP score achieved, over all systems, ranged from a high of 58% in the *motorbike* category to a low of 16% for the *potted plant* category [27]. These results leave dramatic room for improvement. Why is object detection so challenging?

1.2 Challenges

The challenges faced by an object detection system are daunting. Image formation is an inherently lossy process in which a three-dimensional scene is projected on to a two-dimensional image plane. From this incomplete, and often noisy data, the detection system must make accurate inferences about object category membership. For the most part, however, categories are not precisely defined. If asked, people will offer differing opinions about what belongs to a given category. When is an object with four wheels a car? How much can it change before it switches to some other type of vehicle? This semantic fuzziness leads to a large degree of intraclass variation. Yet worse, even in a narrowly defined category, deformation, viewpoint, and lighting can dramatically change the shape and appearance of an object in an image. Together, these factors produce a wide gamut of variation.

visual similarity.

Photometric variation. The exact same scene, imaged under different lighting conditions, can lead to very different results. We build on low-level image features, such as histogram of oriented gradients (HOG) [18], that are (largely) invariant to several illumination transformations. Bias is removed by working with gradients computed from differences of pixel intensities. Changes in gain, both globally and locally, are removed through normalization. The effects of nonlinear transformations are reduced by working with gradient orientations and clipping strong gradient magnitudes.

In this work, we largely ignore photometric variation and instead focuses on higher-level sources of variation. These higher-level sources can be separated into two classes: *single-instance variation* includes the ways in which a single object instance (i.e., the exact same object) can vary from image to image; *intraclass variation* covers the ways in which different instances from the same category differ from each other.

Deformation. At a conceptual level, many objects are composed of a large number of rigid parts that are linked together through nonrigid connections. An object "deforms" when its parts change their relative positions or orientations. The deformation of a 3D object can cause a dramatic change in the projected 2D shape and appearance of the object. For example, self-occlusion makes it impossible to map the visible portion of an object in one image to the visible portion in another image of the deformed object. Other objects, such as cats, are better described as "soft bodies" and their surfaces exhibit smooth, nonrigid deformations. A successful object detection system must be able to recognize objects across a wide range of deformations.

Viewpoint. The 2D shape and appearance of an object may change substantially when the camera's viewpoint is altered. Some small changes in viewpoint can be approximated as deformations. For example, a small amount of foreshortening is visually similar to a deformation that contracts an object along a direction. Larger viewpoint changes, however, often reveal structures that were previously occluded. At the extreme, one view of an object

may share almost no shape similarity with another view of the object.

Subcategories. The notion of an object category is inherently fuzzy. Most categories that we would like to detect (for practical applications) are not defined purely by visual similarity. Take, for instance, the category *airplane*. This category is largely defined by function (airplanes are designed to fly) and composition (airplanes have engines, wings, a fuselage, *etc.*). However, the compositional elements may be visually very *dissimilar* (propeller *vs.* jet engines), they may be arranged in different spatial configurations, and occur with differing counts. Several reasonable subcategories, each of which is visually more consistent than the supercategory, might be: fighter jet, jumbo passenger jet, single-engine propeller plane, and biplane.

Composition. Most object classes are compositional in nature. For example, people are often composed with other objects, such as a marching band member wrapped in a tuba. Other objects may be composed of a variable number of subobjects, such as the repeated carriages in a train. The compositional nature of objects leads to a combinatorial problem: an object detection system needs to cope with the exponentially large space of combinations.

1.3 Contributions

We present several models for category-level object detection. Each model in this sequence builds on the structures and methods employed by the previous models, while staying within the framework of discriminatively trained grammar models. Along the way, we increase representational capacity, develop new machine learning techniques, and focus on efficient computation.

In the development of our models, we make extensive use of large image datasets for quantitative evaluation. In particular, we use recent releases of the PASCAL VOC Challenge datasets [23, 26, 27]. These datasets are critical to the development of our methods

because they are challenging enough to reveal the performance benefits of richer models. This is in contrast to other standard datasets, such as the INRIA Person Dataset [17], for which detection performance has neared a saturation point, and there is very little to gain by increasing model sophistication.

1.3.1 The context circa 2011

In the 2007 PASCAL VOC Challenge, the top-performing system achieved a mean AP of 21% [34]. By the 2011 Challenge, the winning system had pushed this metric to 41% [27]. The methods described in this dissertation account for two-thirds of the performance gain between 2007 and 2011. The remaining one-third of the gap is due to adding extensions (*e.g.*, more image features, more contextual features, and richer deformation models) to the framework and software developed here [78].

Our point of departure — the system that won the 2007 PASCAL VOC detection task — is the discriminatively trained deformable parts model first presented by Felzenszwalb, McAllester, and Ramanan in [34]. Their work builds on the successful combination of histogram of oriented gradients (HOG) features with linear SVM training, proposed by Dalal and Triggs in [18]. The Dalal and Triggs model is a sliding-window detector implemented as a single HOG filter. This filter can be thought of as a template that models the global appearance of an object category. We refer to this type of global appearance filter as a "root filter."

The part-based model in [34] extends the Dalal and Triggs detector by adding higher resolution local part filters that can translate relative to the lower resolution root filter. Additionally, because their model is trained with weak supervision — it is assumed that the training data does not include labels for *what the parts are* or *where they are located* — they extend SVM to handle latent variables. In their part-based model, the latent variables describe the spatial configuration of the parts. The resulting discriminative training formalism, named *latent SVM* (LSVM), is mathematically equivalent to the MI-SVM formulation of

an SVM for multiple instance learning that was introduced in [4].

Building on this work, we now present a brief description of the main models and methods introduced in this dissertation.

1.3.2 Image representation

A strong set of low-level features is essential for good performance. We show how to transform histogram of oriented gradients features into a lower dimensional feature set that includes more information than the original formulation of HOG. We also introduce two extensions: the first allows our system to detect objects that are partially truncated by the image boundary with greater confidence; while the second allows our system to detect smaller objects with greater confidence.

1.3.3 Learning from weakly-labeled data

Training data for vision is often assigned weak labels such as bounding boxes or just the names of objects occurring in the image. In contrast, an image analysis system will often produce strong predictions such as a segmentation or a pose. Existing structured prediction methods, such as structural SVM [65, 67] and latent structural SVM [75], do not directly support weak labels together with strong predictions. We develop the notion of a *weak-label structural SVM* (WL-SSVM) that generalizes structural SVM and latent structural SVM. The key idea is to introduce a loss $L(y, s)$ for making a strong prediction s when the weak training label is y.

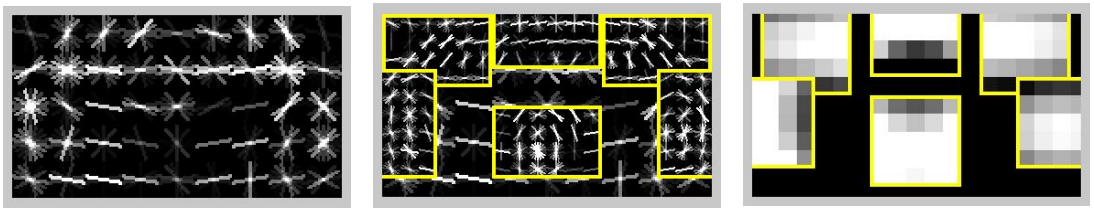
The WL-SSVM learning formalism subsumes LSVM as a special case and allows us to train models with loss functions that depend on structural predictions, rather than binary predictions as is the case with LSVM. We show how to use a WL-SSVM to learn the parameters of a grammar model from bounding box annotations, and then demonstrate that using a structural loss enables us to learn models that perform better on benchmark datasets.

1.3.4 Mixture models

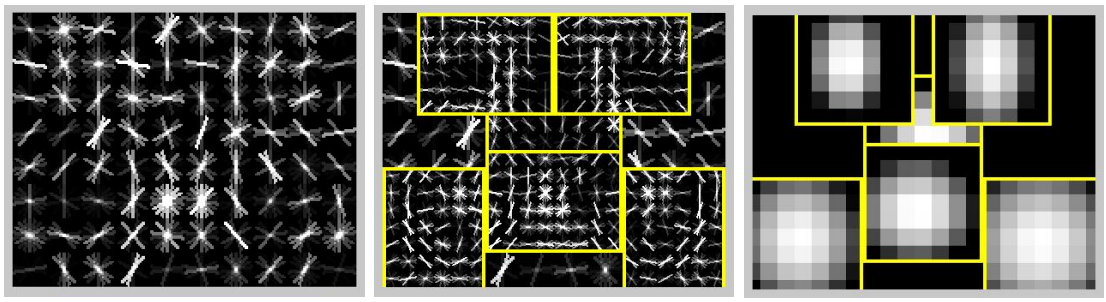
Both the Dalal and Triggs rigid HOG detector, and its part-based extension, use the size and shape of a single root filter to hypothesize detection windows in images. Consider, for a moment, the space of detection windows produced by these detectors. This space only includes detections that have the same aspect ratio as the root filter. This limitation is problematic since there may be a large proportion of object instances from the target category that have very different bounding box aspect ratios. When using an evaluation metric such as the intersection-over-union overlap measure employed by the PASCAL evaluation protocol, outputting detections with a single aspect ratio can severely limit a detector's recall.

Aside from limited recall, the single-root-filter approach is also problematic from the viewpoint of how best to model the appearance and geometry of an object category. Using a single root filter (either with or without parts) requires that a single model must capture all aspects, poses, and structural variation present within a category. This is likely an unreasonable requirement. For example, a bicycle viewed from the front has a substantially different appearance as compared to a bicycle viewed from the side. These two aspects are so visually dissimilar that they do not share any (nonprimitive) parts. The model visualizations in [34], and reproduced in Figure 1.2, reveal that the learned filters appear to be superpositions of multiple aspects. Intuitively, modeling their appearance jointly produces an averaging effect in the filter weights, leading to less specific filters.

Mixture models. The first model that we present builds on the deformable parts models of [34] in a way that helps overcome the aforementioned issues. As in previous work on mixture models [62, 5], the key idea is to decompose a visually complex object category into a set of simpler subcategories. Our goal is to automatically learn subcategory models that span variation in aspect and pose, and where the range of appearance within each subcategory is simple enough that it can be modeled effectively by a single deformable parts model.



(a) car



(b) bicycle

Figure 1.2: Models with a single root filter. Images reproduced from [34].

In our formulation, we treat the subcategory labels as latent information, analogous to the latent cluster labels used when fitting mixture models with expectation maximization (EM). In contrast to existing approaches, such as EM, we use nonprobabilistic, discriminative frameworks — latent SVM and weak-label structural SVM — to learn mixtures. Returning to the bicycle example, our system automatically learns subcategories for: side, 45° angle, and front/rear views.

Latent orientation. For many object categories, photographs taken by people typically capture category instances with significant variation in out-of-plane rotation. Empirically, we find that our method for learning subcategories tends to cluster instances that have similar out-of-plane rotations — *modulo* 180° — together. In the horse category, for example, one of the learned clusters corresponds to side views. Unfortunately this cluster includes both left and right-facing horses, and because their appearance is represented jointly, the result is a model ideally suited to detect two-headed horses (see Figure 1.3 top). To address this problem, we enrich our models by treating orientation within a subcategory as a latent,

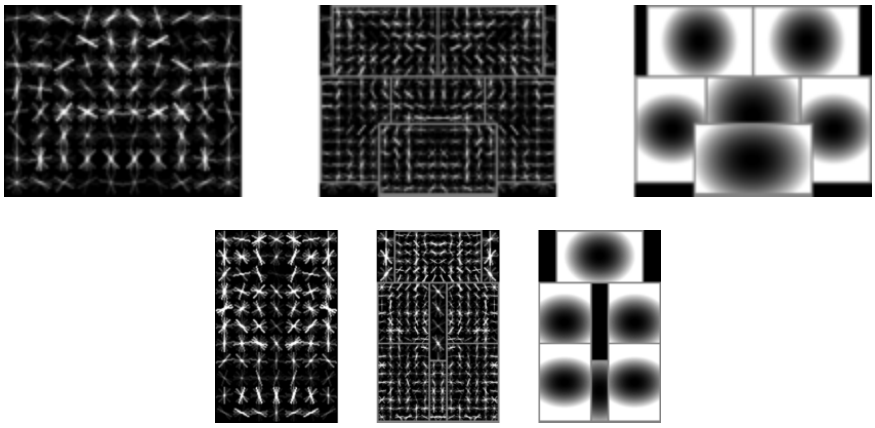
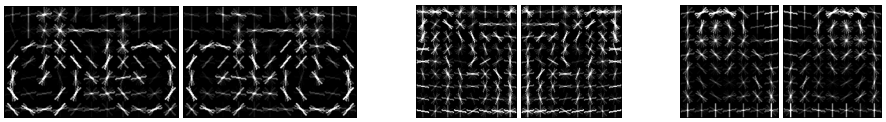


Figure 1.3: Mixture models without latent orientation. A two component horse model. The first component (top) is specialized for detecting side views and the second component (bottom) is specialized for detecting front views.



(a) bicycle (b) horse (c) person

Figure 1.4: Learned latent orientations. For each of three sample classes, we visualize the root filter for one aspect-ratio subcategory trained with latent orientation. Mirrored copies of each root filter are displayed. Compare the horse root filter (b) to the top row in Figure 1.3.

binary choice between a left or right-facing object instantiation (see Figure 1.4).

Both of these modeling extensions fit into the latent SVM and weak-label structural SVM frameworks without modification. However, successfully training these models is quite challenging in practice. Learning meaningful parameters with latent orientation — that is, *not* learning a model where both orientations collapse and become identical — requires careful initialization. Furthermore, we have found that it is essential to remove the classical SVM-based training approximation made in the latent SVM implementation described in [34]. We discuss common learning instabilities that lead to bad local optima and offer general suggestions for avoiding these instabilities in practice.

1.3.5 Object detection grammars

Our mixture models are trained to produce a small number of subcategories, each of which is modeled with independent parameters. These subcategories are initialized by clustering the training examples by aspect ratio. For many categories, aspect ratio is a good proxy for viewpoint (imagine how the aspect ratio of a tight bounding box around the image of a car changes as a photographer circles the car). However, changes in aspect ratio also result from several other factors including articulated pose deformations, occlusion by other objects in the image, or truncation by the image boundary. Visual inspection of the mixture model learned for the person category from the PASCAL data (any year after 2006), reveals that the three subcategories learned for the person class differ in how much of the person is taken to be visible — just the head and shoulders, the head and shoulders together with the upper body, or a whole standing person (see Figure 5.1). Each of the three components has independently trained parts with disjoint sets of parameters. In particular, each component has a head part trained independently from the head part of the other components.

A grammar for detecting people. We build on this observation by constructing a single grammar model that allows more flexibility in describing the amount of the person that is visible. The grammar model avoids dividing the training data between different components and thus uses the training data more efficiently. The parts in the model, such as the head part, are shared across all degrees of visibility in which they appear. The grammar model also includes subtype choice at the part level to accommodate greater appearance variability across object instances. We use parts with subparts to benefit from high-resolution image data, while also allowing for deformations. Unlike in our mixture models, here we explicitly model the source of occlusion for partially visible objects.

1.3.6 Cascaded detection

In order to achieve widespread use in research and applications, an object detection system must not only be accurate, but also fast. All of the models that we consider have a tree graphical structure. The computational implication of this fact is that we can perform exact inference efficiently with dynamic programming. Although there may be advantages in allowing for a graphical structure that contains loops (for example, [66, 73] argue for modeling more relations), we maintain our focus on pushing the detection performance limits of tree-structured models.

Along the line of efficient computation, we develop a technique for "compiling" a fully trained mixture of deformable part models into a much faster detector that implements a cascade architecture [70]. We show how to select the cascade thresholds in a way that is both safe and effective. With no loss in precision over the full range of the original detector's recall, our cascade algorithm computes detections about 15x faster than the baseline dynamic programming algorithm with generalized distance transforms.

1.3.7 Software

Throughout this work, we have made releases of our object detection system available to the public. This dissertation will be accompanied by a new release, `voc-release5`, that supports weak-label structural SVM training.

1.4 Outline

This dissertation is organized in two parts. The first part is about general frameworks for object and image representation (Chapter 2) and for learning (Chapter 3). We review object detection grammars, detail our enhanced histogram of oriented gradients features, and describe an approach to discriminative learning from weakly-labeled data.

The second part is an exploration around various points in the vast landscape of object

detection grammars. We enrich star-structured deformable part models into mixture models with latent orientation (Chapter 4), and then show some promising results with richer detection grammars (Chapter 5). We chronicle the advancement made in object detection performance during the course of this work with several empirical evaluations in Chapter 6. The last technical contribution — cascaded detection — is presented in Chapter 7. This dissertation is then concluded with perspectives on this and future work in Chapter 8.

CHAPTER 2

OBJECT AND IMAGE REPRESENTATIONS

This chapter describes the representations that we use for building object detectors and encoding image data. We start by reviewing object detection grammars in Section 2.1 and Section 2.2. Then, we introduce a reformulation of the histogram of oriented gradients features in Section 2.3. We also present two extensions that help calibrate detection scores between small *vs.* large objects, and between objects that are truncated by the image boundary *vs.* non-truncated objects.

2.1 Object detection grammars

We define all of our object detection models using the *object detection grammar* framework. Felzenszwalb and McAllester describe object detection grammars in full detail in [33], and we review them here for completeness.

2.1.1 Framework overview

From a high-level view, an object detection grammar represents an object category recursively in terms of other objects using a system of symbols and production rules. The framework can be thought of as a modeling language for defining object detectors.

Let $\mathcal{N}$ be a set of nonterminal symbols and $\mathcal{T}$ be a set of terminal symbols. We can think of the terminals as the atomic building blocks of an object. They are never decomposed into simpler constituents. For example, we might choose to declare an "eye" terminal when specifying a person detection grammar or a "wheel" terminal when specifying a car detection grammar. The appearance of a terminal is represented directly, possibly by some type of template.

The nonterminals represent objects that are composed of other objects (often called *parts*), or that have multiple appearance subtypes. Returning to the person detection gram-

mar example, we could declare a "face" nonterminal that is a composition of two "eye" terminals, a "nose" terminal, and a "mouth" nonterminal. The mouth nonterminal could have two appearance subtypes: "smiling mouth" and "frowning mouth."

To represent composition, nonterminals are expanded into bags of terminals and nonterminals according to production rules of the form

$$X \xrightarrow{\beta} \{\ Y_1, \ldots, Y_n\ \}, \tag{2.1}$$

where $X \in \mathcal{N}$, $Y_i \in \mathcal{N} \cup \mathcal{T}$ and $\beta \in \mathbb{R}$ is a bias. The rule in Eq. 2.1 represents an object X as a composition of the objects $Y_1, \ldots, Y_n$. The appearance of a nonterminal is represented indirectly by way of its expansion into terminals.

To represent objects that have a variety of appearance subtypes, we can declare multiple rules with the same symbol on the left-hand side.

$$X \xrightarrow{\beta_1} \{\ W_1, \ldots, W_n\ \} \tag{2.2}$$

$$X \xrightarrow{\beta_2} \{\ Y_1, \ldots, Y_m\ \} \tag{2.3}$$

$$\vdots$$

For example, if X represents a generic vehicle category, then one rule might represent minivans and the other might represent sedans. The biases (β_i) express an *a priori* preference for choosing one particular expansion of X over the alternative expansions.

A grammar has a *cycle* if it is possible for a nonterminal to generate itself through the application of one or more productions. We restrict our attention to *acyclic grammars*, *i.e.*, grammars without cycles.

To describe how an object is arranged in an image, we need to model the instantiation of symbols in images. Let Ω be a set of instantiation values, such as a discrete set of image locations and scales. We denoted by $Y(\omega)$ that symbol $Y \in \mathcal{N} \cup \mathcal{T}$ has been instantiated at $\omega \in \Omega$. In general, Ω may be more expressive than location and scale. It might also include

foreshortening and orientation, for example.

A specific object detection grammar — which we will often call a *grammar model* as shorthand — is defined by a set of productions $\mathcal{R}$ involving instantiated symbols. These productions have the form

$$X(\omega_0) \xrightarrow{\beta} \{\ Y_1(\omega_1), \ldots, Y_n(\omega_n)\ \}, \tag{2.4}$$

where $X \in \mathcal{N}$, $Y_i \in \mathcal{N} \cup \mathcal{T}$, $\omega_i \in \Omega$ and $\beta \in \mathbb{R}$ is a bias.

The size of $\mathcal{R}$ depends on Ω and is typically very large (possibly infinite). Instead of explicitly enumerating all rules in $\mathcal{R}$, we can implicitly represent them with a small set of *production schemas*. Informally, a production schema is a template parameterized by set of free variables. Binding these variables to values converts the template into a single, concrete production. The entire set $\mathcal{R}$ is generated by enumerating all possible variable-value bindings.

More formally, a production schema parameterized by z has the form

$$\forall z \in \mathcal{Z} : X(\omega_0(z)) \xrightarrow{\beta(z)} \{\ Y_1(\omega_1(z)), \ldots, Y_n(\omega_n(z))\ \}. \tag{2.5}$$

The parameter z varies over a set $\mathcal{Z}$ that defines the range of productions that the schema can generate. The functions $\omega_i(z) : \mathcal{Z} \to \Omega$ map parameter values to instantiations and the schema bias $\beta(z) : \mathcal{Z} \to \mathbb{R}$ maps parameter values to production biases.

We will often make $\mathcal{Z}$ explicit when writing schemas. For example, if $\mathcal{Z} = \Omega^{n+1}$ we have

$$\forall(\omega_0, \ldots, \omega_n) \in \Omega^{n+1} : X(\omega_0) \xrightarrow{\beta(\omega_0,\omega_1,\ldots,\omega_n)} \{\ Y_1(\omega_1), \ldots, Y_n(\omega_n)\ \}. \tag{2.6}$$

In this example, the schema bias is a function $\beta : \Omega^{n+1} \to \mathbb{R}$ that takes a set of instantiation values and outputs a rule bias. Note that β can encode complex relationships between the joint spatial arrangement of all symbols participating in the production schema. More generally, the set of schema bias functions defined over schemas sharing the same left-hand

side symbol can express a rich prior over the joint space of which symbols are instantiated and how they are instantiated.

2.1.2 Detection with grammar models

We can expand an instantiated nonterminal into a bag of instantiated terminals by repeatedly applying productions until there are no remaining nonterminals. An expansion of $X(\omega)$ leads to a derivation tree T rooted at $X(\omega)$. Each leaf node of T is labeled with an instantiated terminal $A(\omega)$. Each internal node of T is labeled with the production schema r used to generate the node's children, and with the schema's parameter value z. We assign a score to a derivation tree that combines the production biases used in the tree with a measure of how well the terminals in the leaves match the image data.

Let $\mathcal{X}$ be the space of images[1] and $I \in \mathcal{X}$ be a particular image. To measure how well terminals match image data, we define appearance models using a function $\text{score}(I, A, \omega)$ that computes a real-valued compatibility score for instantiating the terminal A at ω in input I. The score of a derivation tree T instantiated in I is the sum of the biases of the productions used to generate T, plus the score of instantiating the terminals at T's leaves,

$$\text{score}(I, T) = \sum_{(r,z) \in \text{int}(T)} \beta_r(z) + \sum_{A(\omega) \in \text{leaf}(T)} \text{score}(I, A, \omega). \tag{2.7}$$

Given a grammar and an image, we model object detection as the task of finding high scoring derivation trees. This inference problem can be phrased as maximizing $\text{score}(I, T)$ over all derivation trees rooted at a particular instantiated symbol $X(\omega)$. Let $\mathcal{T}_{X(\omega)}$ be the set of all derivation trees rooted at $X(\omega)$. This leads to the problem

$$S^*_{X(\omega)} = \max_{T \in \mathcal{T}_{X(\omega)}} \text{score}(I, T). \tag{2.8}$$

For acyclic grammars, this problem can be solved using a dynamic programming algorithm.

1. In general, we might consider other types of input spaces. For example, $\mathcal{X}$ might be videos or signals captured by some other kind of sensor.

The algorithm begins by sorting the symbols of the grammar so that each symbol Y appears in the sorted list before any symbol X for which Y appears on the right-hand side of X in $\mathcal{R}$. This is exactly the topological ordering of the graph whose vertices are symbols and whose directed edges go from X to Y if and only if there is a rule in $\mathcal{R}$ where X appears on the left-hand side of Y. Given this ordering, score tables S_X are computed for each nonterminal symbol X using the recursion

$$S_X[\omega] = \max_{X(\omega) \xrightarrow{\beta} \{\ Y_1(\omega_1),\ldots,Y_n(\omega_n)\ \}} \beta + \sum_{i=1}^{n} S_{Y_i}[\omega_i], \tag{2.9}$$

and for each terminal symbol A using

$$S_A[\omega] = \text{score}(I, A, \omega). \tag{2.10}$$

The ordering ensures that each score table S_{Y_i} on the right-hand side of Eq. 2.9 has been computed before we attempt to compute the score table S_X on the left-hand side. When all symbols have been processed, the solution to Eq. 2.8 is $S^*_{X(\omega)} = S_X[\omega]$ for all $X \in \mathcal{N} \cup \mathcal{T}$ and $\omega \in \Omega$.

The maximization in Eq. 2.9 is over all rules in $\mathcal{R}$ with $X(\omega)$ on the left-hand side. In general, each production from $X(\omega)$ with a bag of n distinct symbols on the right-hand side may have $|\Omega|^n$ rules in $\mathcal{R}$ containing those symbols. Furthermore, there may be many different production schemas with $X(\omega)$ on the left-hand side. Since $|\Omega|$ is typically very large, even for a small number n this maximization is intractable unless there is special structure in $\mathcal{R}$ that allows Eq. 2.9 to be solved efficiently.

We retain all derivations that score above a threshold, which we set low enough to ensure high recall. Most object detection benchmarks provide ground-truth annotations in the form of bounding boxes and require an object detection algorithm to output predicted bounding boxes, as well. Object detection grammars naturally output derivation trees, and so we need a mechanism to map derivations to bounding boxes. We use $\text{box}(T)$ to denote a detection

window associated with a derivation T. Finally, given a set of candidate detections, we apply nonmaximal suppression to produce a final set of detections.

In general, box(T) may be any function from derivation trees to bounding boxes. It could, for example, predict a bounding box from how the terminals are instantiated by fitting a regression model. For simplicity, we define box(T) by assigning a detection window size, in coordinates that are meaningful with respect to Ω, to each structure schema that can be applied to the grammar's start symbol.

2.1.3 Isolated deformation grammars

We now consider object detection grammars that are specified by two kinds of schemas. These schemas limit the expressive power of the resulting models, but in exchange they introduce structure into $\mathcal{R}$ that makes exact inference efficient.

Let Δ be a set of instantiation displacements, such as changes in position, scale, and rotation. The binary operator $\oplus : \Omega \times \Delta \to \Omega$ maps an instantiation and a displacement to a displaced instantiation.

A *structural schema* generates one production for each instantiation $\omega \in \Omega$,

$$\forall \omega \in \Omega : \; X(\omega) \xrightarrow{\beta} \{ \; Y_1(\omega \oplus \delta_1), \ldots, Y_n(\omega \oplus \delta_n) \; \}. \tag{2.11}$$

Here the $\delta_i \in \Delta$ are *constant* displacements (*i.e.*, the δ_i are *not* template parameters). Structural schemas can be used to define decompositions of objects into other objects, including decompositions that result in variable structure. The number of productions that share $X(\omega)$ on the left-hand is usually small for the models we consider (*e.g.*, 6).

A *deformation schema* generates one production for each instantiation $\omega \in \Omega$ and displacement $\delta \in \Delta$,

$$\forall (\omega, \delta) \in \Omega \times \Delta : \; X(\omega) \xrightarrow{\beta(\delta)} \{ \; Y(\omega \oplus \delta) \; \}. \tag{2.12}$$

Deformation schemas can be used to define deformable models where the deformation cost,

encoded in $\beta(\delta)$, depends only on the displacement between X and Y. Brute force maximization over displacements requires $O(|\Delta|)$ time per $\omega \in \Omega$. For certain forms of $\beta(\delta)$ this maximization can be performed simultaneously for all $\omega \in \Omega$ in $O(|\Omega|)$ time using generalized distance transforms [29, 30]. Alternatively, if Δ only allows for *local* displacements, then $|\Delta|$ might be bounded by a small constant making brute force maximization very fast in practice.

2.2 Linear parameterization

We would like to choose production biases and terminal appearance models that lead to good performance on object detection tasks. In this section, we present a linear parameterization that allows us to set model parameters from training data by applying efficient machine learning tools.

2.2.1 Filters and feature pyramids

Our appearance models are linear filters that are applied to dense feature maps. A feature map is a two-dimensional array whose entries are h-dimensional feature vectors computed on a dense grid of image locations (*e.g.*, every 8×8 pixels). Intuitively, each feature vector describes a small image patch. In practice we use a low-dimensional variation of the histogram of oriented gradients (HOG) features from [18], described in Section 2.3 and [32], but the methods described in this dissertation are independent of the choice of features.

A filter is a two-dimensional array whose entries are h-dimensional weight vectors. Filters are tuned to respond to a specific spatial arrangement of some particular feature vectors near a fixed scale. Because objects appear at a wide range of scales, we apply the same filter to multiple feature maps, each computed from a smoothed and subsampled version of the original image.

To fix notation, let $\mathcal{H} = \{G_1, \ldots, G_L\}$ be an L-level feature pyramid, consisting of feature

maps G_1 through G_L, and let $\omega = (x, y, l)$ specify a position and level (scale) within $\mathcal{H}$. The score, or response, of a $U \times V$ filter F instantiated at ω in $\mathcal{H}$ is the summed dot product of the filter and the subarray of the feature map G_l with top-left corner at (x, y),

$$\mathbf{f} \cdot \boldsymbol{\phi}_{\text{app}}(\mathcal{H}, F, \omega) = \sum_{u=0}^{U-1} \sum_{v=0}^{V-1} F[u, v] \cdot G_l[x + u, y + v]. \tag{2.13}$$

$F[x, y]$ and $G_l[x, y]$ are the h-dimensional vectors located at index (x, y) in arrays F and G_l. The summed dot product can be expressed more concisely as a single dot product, where $\mathbf{f}$ is a vectorized representation of F, formed by concatenating the h-dimensional vectors that comprise the filter in a particular order (*e.g.*, row-major order). The vector-valued function $\boldsymbol{\phi}_{\text{app}}(\mathcal{H}, F, \omega)$ extracts the subarray with the same extent as F from level l of feature pyramid $\mathcal{H}$ with top-left corner at (x, y), and then vectorizes the feature map subarray in the same order that was used to construct $\mathbf{f}$.[2]

Figure 2.1 shows an example image pyramid, the corresponding feature pyramid, and a configuration of filters at different locations and scales within the pyramid. Figure 4.1 shows some examples of filters, feature maps, and filter responses.

The scale sampling in a feature pyramid is determined by a parameter λ defining the number of levels in an octave. That is, λ is the number of levels we need to go down in the pyramid to get to a feature map computed at twice the resolution of another one. In practice we have used $\lambda \in \{4, 5\}$ in training and $\lambda \in \{8, 10\}$ at test time. Fine sampling of scale space is important for obtaining high performance with our models.

We let Ω be the set of feature pyramid positions and scales. We define the appearance models for terminals by associating a filter F_A with each terminal A. Then $\text{score}(I, A, \omega) = \mathbf{f}_A \cdot \boldsymbol{\phi}_{\text{app}}(\mathcal{H}_I, F_A, \omega)$ is the dot product between the filter weights and the features in a subarray of the feature pyramid $\mathcal{H}_I$ computed from image I.

2. The subscript "app" stands for "appearance."

2.2.2 Production biases

All productions generated by the same structural schema share a common bias, so we only need to learn a single bias parameter for each structural schema.

To specify deformation schemas, we let the space of displacements be shifts within *a single level* of a feature pyramid: $\Delta_{\delta_l=0} = \{(\delta_x, \delta_y, \delta_l) \in \Delta \mid \delta_l = 0\}$. All productions generated by the same deformation schema r share a common bias function $\beta_r(\delta)$. We typically define this function to be a separable quadratic in δ_x and δ_y, though other options are possible provided that a linear parameterization is used. We define the deformation feature function $\boldsymbol{\phi}_\Delta : \Delta \to \mathbb{R}^4$ as

$$\boldsymbol{\phi}_\Delta(\delta) = (\delta_x^2,\ \delta_x,\ \delta_y^2,\ \delta_y)^\mathsf{T} \tag{2.14}$$

so that $\beta_r(\delta) = \mathbf{w}_r \cdot \boldsymbol{\phi}_\Delta(\delta)$ for some deformation parameters $\mathbf{w}_r$.

2.2.3 Linear score function

The score of a derivation tree can now be written as a linear function in the model parameters. Let $\mathbf{w} \in \mathbb{R}^d$ be the vector of model parameters. Each terminal and schema in a grammar requires a set of parameters, which are organized as contiguous blocks in $\mathbf{w}$:

$$\mathbf{w} = (\mathbf{w}_1^\mathsf{T}, \ldots, \mathbf{w}_{|\mathcal{R}|}^\mathsf{T}, \mathbf{w}_{A_1}^\mathsf{T}, \ldots, \mathbf{w}_{A_{|\mathcal{T}|}}^\mathsf{T})^\mathsf{T}, \tag{2.15}$$

where the grammar is defined by schemas[3] $\mathcal{R}$ and terminals $\mathcal{T} = \{A_1, \ldots, A_{|\mathcal{T}|}\}$.

We define sparse feature vectors $\boldsymbol{\phi}_r \in \mathbb{R}^d$ and $\boldsymbol{\phi}_A(I, \omega) \in \mathbb{R}^d$ for each production schema r and terminal A, so that we can write biases and terminal appearance scores as a dot product between $\mathbf{w}$ and a (sparse) feature vector.

For a structural schema r, we set r's block of the sparse feature vector $\boldsymbol{\phi}_r$ to a constant (such as 1). For a deformation schema r, we set r's block of $\boldsymbol{\phi}_r$ to $\boldsymbol{\phi}_\Delta(\delta)$, where δ comes from a schema parameter assignment $z = (\omega, \delta)$ in a derivation tree. For a terminal A, we

3. Previously we used $\mathcal{R}$ to denote a set of instantiated productions. Here, the elements of $\mathcal{R}$ are schemas.

set A's block in $\boldsymbol{\phi}_A(I,\omega)$ to $\boldsymbol{\phi}_{\text{app}}(\mathcal{H}_I, F_A, \omega)$.

Given this construction, we can write the score of a derivation tree T on an image I as a linear function in the model parameters.

$$\begin{aligned}
\text{score}(I,T) &= \sum_{(r,z)\in\text{int}(T)} \beta_r(z) + \sum_{A(\omega)\in\text{leaf}(T)} \text{score}(I,A,\omega) \\
&= \sum_{(r,z)\in\text{int}(T)} \mathbf{w}\cdot\boldsymbol{\phi}_r(z) + \sum_{A(\omega)\in\text{leaf}(T)} \mathbf{w}\cdot\boldsymbol{\phi}_A(I,\omega) \\
&= \mathbf{w}\cdot\left[\sum_{(r,z)\in\text{int}(T)} \boldsymbol{\phi}_r(z) + \sum_{A(\omega)\in\text{leaf}(T)} \boldsymbol{\phi}_A(I,\omega)\right] \\
&= \mathbf{w}\cdot\boldsymbol{\psi}(I,T). \qquad (2.16)
\end{aligned}$$

The final equality shows that the score can be written as the dot production between the model parameters and a cumulative feature vector, $\boldsymbol{\psi}(I,T)$, that depends on the derivation tree and image.

Lastly, we note that it is possible to tie parameters so that, for example, several deformation schemas share the same deformation parameters, or an appearance model is defined as a function of the parameters of another appearance model (as is done with the analytically "flipped" HOG features described in Section 4.3).

2.3 Image features

Here we describe the 36-dimensional histogram of oriented gradients (HOG) features from [18] and introduce an alternative 13-dimensional feature set that captures essentially the same information.[4] We have found that augmenting this low-dimensional feature set to include both contrast sensitive and contrast insensitive features, leading to a 31-dimensional feature vector, improves performance for most classes of the PASCAL datasets.

4. There are some small differences between the 36-dimensional features defined here and the ones in [18], but we have found that these differences did not have any significant effect on the performance of our system.

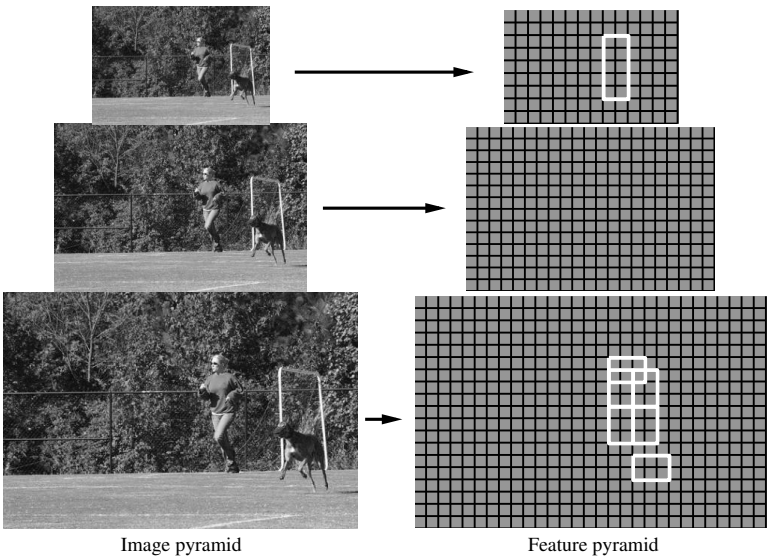


Figure 2.1: A feature pyramid and an instantiation of a person model within that pyramid. The part filters are placed at twice the spatial resolution of the placement of the root.

We also describe two complementary features that (1) allow our models to learn some robustness to truncation at the image boundary, and (2) to better calibrate the scores of small objects with the scores of larger objects.

2.3.1 HOG features and enhanced HOG features

Pixel-level feature maps

Let $\theta(x, y)$ and $r(x, y)$ be the orientation and magnitude of the intensity gradient at a pixel (x, y) in an image. As in [18], we compute gradients using finite difference filters, $[-1, 0, +1]$ and its transpose. For color images we use the color channel with the largest gradient magnitude to define θ and r at each pixel.

The gradient orientation at each pixel is discretized into one of p values using either a contrast sensitive (B_1), or insensitive (B_2), definition,

$$B_1(x, y) = \text{round}\left(\frac{p\theta(x, y)}{2\pi}\right) \bmod p \tag{2.17}$$

$$B_2(x, y) = \text{round}\left(\frac{p\theta(x, y)}{\pi}\right) \bmod p \tag{2.18}$$

Below we use B to denote either B_1 or B_2.

We define a pixel-level feature map that specifies a sparse histogram of gradient magnitudes at each pixel. Let $b \in \{0, \ldots, p-1\}$ range over orientation bins. The feature vector at (x, y) is

$$F(x,y)_b = \begin{cases} r(x,y) & \text{if } b = B(x,y) \\ 0 & \text{otherwise} \end{cases} \tag{2.19}$$

We can think of F as an oriented edge map with p orientation channels. For each pixel we select a channel by discretizing the gradient orientation. The gradient magnitude can be seen as a measure of edge strength.

Spatial aggregation

Let F be a pixel-level feature map for a $w \times h$ image. Let $k > 0$ be a parameter specifying the side length of a square image region. We define a dense grid of rectangular "cells" and aggregate pixel-level features to obtain a cell-based feature map C, with feature vectors $C(i, j)$ for $0 \leq i \leq \lfloor (w-1)/k \rfloor$ and $0 \leq j \leq \lfloor (h-1)/k \rfloor$. This aggregation provides some invariance to small deformations and reduces the size of a feature map.

The simplest approach for aggregating features is to map each pixel (x, y) into a cell $(\lfloor x/k \rfloor, \lfloor y/k \rfloor)$ and define the feature vector at a cell to be the sum (or average) of the pixel-level features in that cell.

Rather than mapping each pixel to a unique cell we follow [18] and use a "soft binning" approach where each pixel contributes to the feature vectors in the four cells around it using bilinear interpolation.

Normalization and truncation

Gradients are invariant to changes in bias. Invariance to gain can be achieved via normalization. Dalal and Triggs [18] used four different normalization factors for the feature vector

$C(i,j)$. We can write these factors as $N_{\delta,\gamma}(i,j)$ with $\delta,\gamma \in \{-1,1\}$,

$$N_{\delta,\gamma}(i,j) = \left(\|C(i,j)\|^2 + \|C(i+\delta,j)\|^2 + \|C(i,j+\gamma)\|^2 + \|C(i+\delta,j+\gamma)\|^2\right)^{\frac{1}{2}}. \quad (2.20)$$

Each factor measures the gradient "energy" in a square block of four cells containing (i,j).

Let $T_\alpha(v)$ denote the component-wise truncation of a vector v by α (the i-th entry in $T_\alpha(v)$ is the minimum of the i-th entry of v and α). The HOG feature map is obtained by concatenating the result of normalizing the cell-based feature map C with respect to each normalization factor followed by truncation,

$$H(i,j) = \begin{pmatrix} T_\alpha(C(i,j)/N_{-1,-1}(i,j)) \\ T_\alpha(C(i,j)/N_{+1,-1}(i,j)) \\ T_\alpha(C(i,j)/N_{+1,+1}(i,j)) \\ T_\alpha(C(i,j)/N_{-1,+1}(i,j)) \end{pmatrix} \quad (2.21)$$

Commonly used HOG features are defined using $p = 9$ contrast insensitive gradient orientations (discretized with B_2), a cell size of $k = 8$ and truncation $\alpha = 0.2$. This leads to a 36-dimensional feature vector. We used these parameters in the analysis described below.

PCA and analytic dimensionality reduction

We collected a large number of 36-dimensional HOG features from different resolutions of a large number of images and performed PCA on these vectors. The principal components are shown in Figure 2.2. The results lead to a number of interesting discoveries.

The eigenvalues indicate that the linear subspace spanned by the top 11 eigenvectors captures essentially all the information in a HOG feature. In fact we obtain the same detection performance in all categories of the PASCAL 2007 dataset using the original 36-dimensional features or 11-dimensional features defined by projection to the top eigenvectors. Using lower dimensional features leads to models with fewer parameters and speeds up the detection and learning algorithms. We note however that some of the gain is lost because

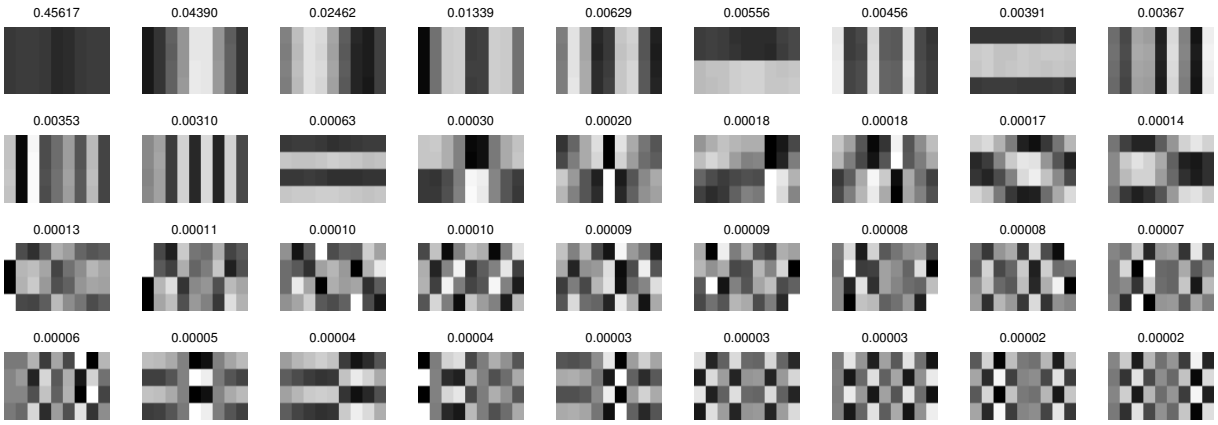


Figure 2.2: PCA of HOG features. Each eigenvector is displayed as a 4 by 9 matrix so that each row corresponds to one normalization factor and each column to one orientation bin. The eigenvalues are displayed on top of the eigenvectors. The linear subspace spanned by the top 11 eigenvectors captures essentially all of the information in a feature vector. Note how all of the top eigenvectors are either constant along each column or row of the matrix representation.

we need to perform a relatively costly projection step when computing feature pyramids.

Recall that a 36-dimensional HOG feature is defined using 4 different normalizations of a 9 dimensional histogram over orientations. Thus a 36-dimensional HOG feature is naturally viewed as a 4×9 matrix. The top eigenvectors in Figure 2.2 have a very special structure: they are each (approximately) constant along each row or column of their matrix representation. Thus the top eigenvectors lie (approximately) in a linear subspace defined by sparse vectors that have ones along a single row or column of their matrix representation.

Let $V = \{u_1, \ldots, u_9\} \cup \{v_1, \ldots, v_4\}$ with

$$u_k(i,j) = \begin{cases} 1 & \text{if } j = k \\ 0 & \text{otherwise} \end{cases} \qquad v_k(i,j) = \begin{cases} 1 & \text{if } i = k \\ 0 & \text{otherwise} \end{cases}$$

We can define a 13-dimensional feature by taking the dot product of a 36-dimensional HOG feature with each u_k and v_k. Projection into each u_k is computed by summing over the 4 normalizations for a fixed orientation. Projection into each v_k is computed by summing over 9 orientations for a fixed normalization.[5]

5. The 13-dimensional feature is not a linear projection of the 36-dimensional feature into V because the

As in the case of 11-dimensional PCA features we obtain the same performance using the 36-dimensional HOG features or the 13-dimensional features defined by V. However, the computation of the 13-dimensional features is much less costly than performing projections to the top eigenvectors obtained via PCA since the u_k and v_k are sparse. Moreover, the 13-dimensional features have a simple interpretation as 9 orientation features and 4 features that reflect the overall gradient energy in different areas around a cell.

We can also define low-dimensional features that are contrast sensitive. We have found that performance on some object categories improves using contrast sensitive features, while some categories benefit from contrast insensitive features. Thus in practice we use feature vectors that include both contrast sensitive and insensitive information.

Let C be a cell-based feature map computed by aggregating a pixel-level feature map with 9 contrast insensitive orientations. Let D be a similar cell-based feature map computed using 18 contrast sensitive orientations. We define 4 normalization factors for the (i, j) cell of C and D using C as in Eq. 2.20. We can normalize and truncate $C(i, j)$ and $D(i, j)$ using these factors to obtain $4 \times (9 + 18) = 108$ dimensional feature vectors, $F(i, j)$. In practice we use an analytic projection of these 108-dimensional vectors, defined by 27 sums over different normalizations, one for each orientation channel of F, and 4 sums over the 9 contrast insensitive orientations, one for each normalization factor. We use a cell size of $k = 8$ and truncation value of $\alpha = 0.2$. The final feature map has 31-dimensional vectors $G(i, j)$, with 27 dimensions corresponding to different orientation channels (9 contrast insensitive and 18 contrast sensitive), and 4 dimensions capturing the overall gradient energy in square blocks of four cells around (i, j).

Finally, we note that the top eigenvectors in Figure 2.2 can be roughly interpreted as a two-dimensional separable Fourier basis. Each eigenvector can be roughly seen as a sine or cosine function of one variable. This observation could be used to define features using a finite number of Fourier basis functions instead of a finite number of discrete orientations.

---

u_k and v_k are not orthogonal. In fact the linear subspace spanned by V has dimension 12.

The appearance of Fourier basis in Figure 2.2 is an interesting empirical result. The eigenvectors of a $d \times d$ covariance matrix Σ form a Fourier basis when Σ is circulant, *i.e.*, $\Sigma_{i,j} = k(i - j \bmod d)$ for some function k. Circulant covariance matrices arise from probability distributions on vectors that are invariant to rotation of the vector coordinates. The appearance of a two-dimensional Fourier basis in Figure 2.2 is evidence that the distribution of HOG feature vectors on natural images have (approximately) a two-dimensional rotational invariance. We can rotate the orientation bins and independently rotate the four normalizations blocks.

2.3.2 Image boundary truncation features

The PASCAL dataset contains many examples of objects that are truncated by the image boundary. To make detection more robust to truncations caused by an image boundary we pad each image's feature map with a boundary region.

In [34, 32] the HOG features in the boundary region are set to the zero vector. This leads to a fixed score of zero for any filter placed entirely outside of the image. Scoring zero may be inappropriately calibrated with respect to filter responses on background patches inside the image. To compensate for this effect, we augment our HOG feature vectors with an additional feature that takes the value 0 if the feature is inside the image and 1 if the feature is in the boundary region. This "boundary truncation feature" enables the learning of a bias parameter for each filter cell that is added to the filter response if that filter cell is placed in the boundary region.

The truncation feature we use is the same as the one proposed in [69], however there are two important differences in our implementation. The implementation described in [69] uses a single truncation feature *per filter* (not per filter cell) that counts the number of filter cells that are placed in the boundary region. Our per-cell features allow us to learn a spatially varying model. Intuitively, as more of a filter is placed outside of the image, the filter should become less confident. The second difference is in our training data requirements.

The training procedure in [69] requires manually extending the PASCAL bounding boxes to indicate how far each bounding box, that is truncated by the image boundary, ought to extend into the boundary region. Our approach does not require any change to the PASCAL annotations. Instead, during the latent variable completion stage of our training procedure, we measure overlap of the putative detection window with the ground-truth bounding box after first clipping the detection window to the image boundary.

2.3.3 Detecting small objects

When computing a HOG feature pyramid we typically use 8×8 pixel HOG cells. In order to detect small objects, we augment the feature pyramid with one or two additional octaves: one computed using 4×4 pixel cells, and, optionally, the other using 2×2 pixel cells. The local contrast normalization used in HOG feature computation helps make features computed over patches of different sizes comparable. However, we have found empirically that HOG features computed on smaller patches tend to have smaller ℓ_2 norms. This difference in the statistics of HOG features can make the score of a small object incompatible with the score of a larger object. Small objects will produce scores with a smaller dynamic range than large objects.

To help make scores more compatible, we can learn a constant that is added to the score of a small object detection. The most direct way to do this is by augmenting the form of our production schema biases. Let $\omega = (x, y, l) \in \Omega$. We define a new location feature function, $\boldsymbol{\phi}_{\text{loc}}(\omega) = (\mathbf{1}_{\{0<l\leq\lambda\}}, \mathbf{1}_{\{\lambda<l\leq 2\lambda\}}, \mathbf{1}_{\{2\lambda<l\}})^{\mathsf{T}}$, where $\mathbf{1}_{\{P\}}$ is 1 if P is true and 0 if P is false. Recall that λ is the number of levels that separate an octave in the feature pyramid. This feature indicates if the location ω is in the first octave of the pyramid, the second octave, or any of the octaves above the second. For a structural schema, the bias becomes $\beta_r(\omega) = \mathbf{w}_r \cdot (1, \boldsymbol{\phi}_{\text{loc}}(\omega)^{\mathsf{T}})^{\mathsf{T}}$, instead of a constant. For a deformation schema, the bias becomes the sum of the deformation cost and the location score: $\beta_r(\omega, \delta) = \mathbf{w}_r \cdot (\boldsymbol{\phi}_{\Delta}(\delta)^{\mathsf{T}}, \boldsymbol{\phi}_{\text{loc}}(\omega)^{\mathsf{T}})^{\mathsf{T}}$.

Alternatively, we could define $\boldsymbol{\phi}_{\text{loc}}(\omega)$ to depend on the (x, y) components of ω as well.

This would enable us to learn an absolute spatial prior on where objects appear in images.

CHAPTER 3

LEARNING FROM WEAKLY-LABELED DATA

There are two learning problems associated with grammar models: grammar *structure* learning; and grammar *parameter* learning.

In structure learning, the units that form the grammar — the terminals, nonterminals, and production rules — are learned from training data. This problem is widely regarded as the harder of the two, and is the subject of future work. In this thesis we rely on manually declared rules and heuristics to "learn" grammar structure.

The second learning problem is: given an object detection grammar *with a fixed structure*, select values for the grammar's parameters. There are various ways of framing this problem. We choose a task-oriented, discriminative approach whereby we seek parameters that make our models effective in practice. We define "effectiveness" through a quantitative goal, such as increasing average precision scores on a benchmark dataset.

In this chapter, we introduce a new discriminative training formalism for learning models from weakly-labeled examples. We call our approach *weak-label structural* SVM (WL-SSVM). It generalizes structural SVM [65, 67], latent structural SVM [75], and structural ramp loss machines [13, 20, 52]. It also includes latent SVM, which we used to train models in [32], as a special case.

After defining our learning framework, we give an algorithm that solves a WL-SSVM on a generic dataset. Then we discuss critical specializations of the algorithm that yield greater efficiency when learning from large object detection datasets.

Solving for the parameters of a WL-SSVM requires optimizing a nonconvex function. As with other methods that use latent information and nonconvex objectives (*e.g.*, EM for mixture models [19] or hidden CRFs [42, 59]), we are only able to find a local optimum of our objective function. Parameter learning therefore is sensitive to initialization. Since we do not yet have a generic approach for initializing our optimization problem (or learning grammar structure), we discuss solutions to these important problems in the context of the

specific models developed in the subsequent chapters.

3.1 Weak-label structural SVM

Here we define a general formalism for learning functions from weakly-labeled data. Let $\mathcal{X}$ be an input space, $\mathcal{Y}$ be a label space, and $\mathcal{S}$ be an output space. We are interested in learning functions $f : \mathcal{X} \to \mathcal{S}$ from a set of training examples $\{(x_1, y_1), \ldots, (x_n, y_n)\}$, where $(x_i, y_i) \in \mathcal{X} \times \mathcal{Y}$. In contrast to the usual supervised learning setting, we do not assume that the label space and the output space are the same. In particular there may be many output values that are compatible with a label, and we can think of each example as being only weakly labeled. It will also be useful to associate a subset of possible outputs, $\mathcal{S}(x) \subseteq \mathcal{S}$, with an example x. In this case $f(x) \in \mathcal{S}(x)$. This restriction makes it easy to constrain predictions based on *a priori* knowledge of which outputs are valid for a given input.

We draw a connection between labels and outputs with a loss function $L : \mathcal{Y} \times \mathcal{S} \to \mathbb{R}_{\geq 0}$. The function $L(y, s)$ computes the cost for predicting output $s \in \mathcal{S}$ when an example is labeled $y \in \mathcal{Y}$. Let $\mathcal{D}$ be a distribution over $\mathcal{X} \times \mathcal{Y}$. Then a natural goal is to find a function f with low expected loss $\mathbb{E}_{\mathcal{D}}[L(y, f(x))]$.

A simple example of a weakly-labeled training problem comes from learning sliding window detectors in the PASCAL object detection dataset. The training data specifies pixel-accurate bounding boxes for the target objects while a sliding window detector reports boxes with a fixed aspect ratio and at a finite number of scales. The output space is, therefore, a subset of the label space. It is unlikely that any given label y is in $\mathcal{S}$, and therefore *the model cannot predict the training labels*. This discrepancy creates ambiguity: Which output should we train the detector to predict for each example? Naturally, we want the detector to predict an output with low loss given each example's label.

As usual, we assume f is parameterized by a vector of model parameters $\mathbf{w}$ and generates

predictions by maximizing a linear function of a feature map $\boldsymbol{\psi}(x, s)$.

$$f_{\mathbf{w}}(x) = \operatorname*{argmax}_{s \in \mathcal{S}(x)} \mathbf{w} \cdot \boldsymbol{\psi}(x, s) \tag{3.1}$$

To make the parameterization explicit, we write f as $f_{\mathbf{w}}$.

The true distribution $\mathcal{D}$ is unknown, so we train $\mathbf{w}$ by minimizing a regularized risk on the training set. A *weak-label structural* SVM is defined by the training equation

$$E(\mathbf{w}) = \frac{1}{2}\|\mathbf{w}\|^2 + C \sum_{i=1}^{n} L_{\text{surr}}(\mathbf{w}, x_i, y_i). \tag{3.2}$$

The surrogate training loss L_{surr} is defined in terms of two different loss augmented predictions.

$$\begin{aligned} &L_{\text{surr}}(\mathbf{w}, x, y) \\ &\quad = \underbrace{\max_{s \in \mathcal{S}(x)} \left[\mathbf{w} \cdot \boldsymbol{\psi}(x, s) + L_{\text{margin}}(y, s)\right]}_{\text{(3.3a)}} - \underbrace{\max_{s \in \mathcal{S}(x)} \left[\mathbf{w} \cdot \boldsymbol{\psi}(x, s) - L_{\text{output}}(y, s)\right]}_{\text{(3.3b)}} \end{aligned} \tag{3.3}$$

L_{margin} encourages high-loss outputs to "pop out" of (3.3a), so that their scores get pushed down. L_{output} suppresses high-loss outputs in (3.3b), so the score of a low-loss prediction gets pushed up.

One option is to take $L_{\text{margin}} = L_{\text{output}} = L$. In this case L_{surr} becomes a type of ramp loss [13, 20, 52]. Alternatively, several popular learning frameworks can be derived as special cases of WL-SSVM. For the examples below, let $\bar{\delta}(a, b) = 0$ when $a = b$, and $\bar{\delta}(a, b) = \infty$ when $a \neq b$. We will see that the choice of L_{output} can have a significant effect on the computational difficulty of the training problem.

Structural SVM. Let $\mathcal{S} = \mathcal{Y}$, $L_{\text{margin}} = L$ and $L_{\text{output}}(y, \hat{y}) = \bar{\delta}(y, \hat{y})$. Then $L_{\text{surr}}(\mathbf{w}, x, y)$ is the hinge loss used in a structural SVM [67, 65]. In this case L_{surr} is convex in $\mathbf{w}$ because the maximization in (3.3b) disappears. We note, however, that this choice of L_{output} may be problematic and lead to inconsistent training problems. Consider the following situation.

A training example (x, y) may be compatible with a different label $\hat{y} \neq y$, in the sense that $L(y, \hat{y}) = 0$. But even in this case a structural SVM pushes the score $\mathbf{w} \cdot \boldsymbol{\psi}(x, y)$ to be above $\mathbf{w} \cdot \boldsymbol{\psi}(x, \hat{y})$. A structural SVM is too strict because it punishes models that give high scores to low-loss predictions that are not exactly the same as the training labels. This issue can be addressed by relaxing L_{output} to allow for a maximization over labels in (3.3b).

Latent structural SVM. Now let $\mathcal{Z}$ be a space of latent values, $\mathcal{S} = \mathcal{Y} \times \mathcal{Z}$, $L_{\text{margin}} = L$ and $L_{\text{output}}(y, (\hat{y}, \hat{z})) = \bar{\delta}(y, \hat{y})$. Then $L_{\text{surr}}(\mathbf{w}, x, y)$ is the hinge loss used in a latent structural SVM [75]. In this case L_{surr} is not convex in $\mathbf{w}$ due to the maximization over latent values in (3.3b). As in the previous example, this choice of L_{output} can be problematic because it "requires" that the training labels be predicted exactly. This can be addressed by relaxing L_{output}, as in the previous example.

Ramp loss. If we let $\mathcal{Y} = \mathcal{S}$ and take $L_{\text{margin}} = L$ and $L_{\text{output}} = 0$, then we produce the ramp loss that was shown to be consistent in [52]. Ramp loss allows the model to effectively change the ground-truth training labels to alternative labels, at a bounded cost, making it more robust to labeling errors, output-space/label-space mismatch, and outliers.

3.1.1 Discussion

The choice of L_{output} has a strong effect on the computational and statistical properties of the resulting WL-SSVM. In one case (structural SVM), the objective becomes convex, but is not consistent. In another case (ramp loss), the objective becomes consistent, but is not convex. From a statistical point of view, it is appealing to choose $L_{\text{output}} = 0$ — the ramp loss formulation. However, this choice may not be practical from a computational viewpoint.

We cannot optimize the WL-SSVM objective globally. In practice we find a local optimum using an iterative method such as stochastic subgradient descent (SGD) or the convex-concave procedure. Our empirical experience suggests that the computational problems that

arise from ramp loss cannot be ignored: current optimization techniques tend to find low-performing, locally optimal solutions.

Consider the update rule used in SGD (ignoring regularization):

$$\mathbf{w} \leftarrow \mathbf{w} + \Delta\mathbf{w} \tag{3.4}$$

$$\Delta\mathbf{w} \propto \boldsymbol{\psi}(x, \hat{s}^-) - \boldsymbol{\psi}(x, \hat{s}^+) \tag{3.5}$$

$$\hat{s}^+ = \operatorname*{argmax}_{s \in \mathcal{S}(x)} \mathbf{w} \cdot \boldsymbol{\psi}(x, s) + L_{\text{margin}}(y, s) \tag{3.6}$$

$$\hat{s}^- = \operatorname*{argmax}_{s \in \mathcal{S}(x)} \mathbf{w} \cdot \boldsymbol{\psi}(x, s) - L_{\text{output}}(y, s). \tag{3.7}$$

In each step, the parameters are updated so the score $\mathbf{w} \cdot \boldsymbol{\psi}(x, \hat{s}^+)$ is lower and the score $\mathbf{w} \cdot \boldsymbol{\psi}(x, \hat{s}^-)$ is higher. Intuitively, we want $\hat{s}^+$ to be a "bad" (high-loss) output for x and $\hat{s}^-$ to be a "good" (low-loss) output for x.[1] Then, each step would move the weight vector away from a bad prediction and towards a good prediction.

Crucially, $\hat{s}^+$ and $\hat{s}^-$ are *predictions made by the current model* $\mathbf{w}$. What happens if $\mathbf{w}$ is a poor model, *i.e.*, it makes bad predictions? We might expect this to be the case right after initialization, or if the learning rate is not small enough. A bad model will predict a high-loss output for $\hat{s}^+$, and so the first half of the update is safe. However, if $L_{\text{output}} = 0$, the model will likely also predict a high-loss output for $\hat{s}^-$. In total, the update will move $\mathbf{w}$ away from one high-loss prediction, but towards another high-loss prediction. Learning will not make progress.

Our experience confirms that setting $L_{\text{output}} = 0$ makes optimization too difficult for our particular models and datasets. Note how this difficulty is avoided in the case of a structural SVM where the choice of L_{output} converts the optimization problem from a nonconvex one to convex one. That approach requires full supervision, and is problematic as noted above. It suggests, however, that we can use L_{output} as a general mechanism for making our objective function more amenable to optimization. We can encode our *a priori* knowledge of which

1. Or, we want the two outputs to be equal, in which case the update cancels.

outputs we think are good directly in L_{output}. If successful, the loss adjustment will ensure that $\hat{s}^-$ is a good prediction, and thus will help guide the optimization procedure away from bad local optima.

In Chapters 4 and 5 we use a particularly strong $0/\infty$ form of L_{output}. This function encodes our knowledge that reasonable outputs for a training example must have high overlap with the ground-truth bounding box label for that example.

3.1.2 Optimization with the convex-concave procedure

Since L_{surr} is not convex, minimizing the WL-SSVM objective (Eq. 3.2) is a nonconvex optimization problem. We follow [75] in which the concave-convex procedure (CCCP) [77] was used to find a local optima of a similar objective.

The convex-concave procedure for WL-SSVM

CCCP is an iterative algorithm that uses a decomposition of the objective into a sum of convex and concave parts: $E(\mathbf{w}) = E_{\text{convex}}(\mathbf{w}) + E_{\text{concave}}(\mathbf{w})$. We have the natural decomposition

$$E_{\text{convex}}(\mathbf{w}) = \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{n} \max_{s\in\mathcal{S}(x_i)} \left[\mathbf{w}\cdot\boldsymbol{\psi}(x_i,s) + L_{\text{margin}}(y_i,s)\right] \tag{3.8}$$

$$E_{\text{concave}}(\mathbf{w}) = -C\sum_{i=1}^{n} \max_{s\in\mathcal{S}(x_i)} \left[\mathbf{w}\cdot\boldsymbol{\psi}(x_i,s) - L_{\text{output}}(y_i,s)\right]. \tag{3.9}$$

In each iteration, CCCP computes a linear upper bound to E_{concave} based on a current weight vector $\mathbf{w}_t$. The bound depends on subgradients of the summands in Eq. 3.9. For each summand, we choose the subgradient $\boldsymbol{\psi}(x_i, s_i(\mathbf{w}_t))$, where

$$s_i(\mathbf{w}) = \operatorname*{argmax}_{s\in\mathcal{S}(x_i)} \left[\mathbf{w}\cdot\boldsymbol{\psi}(x_i,s) - L_{\text{output}}(y_i,s)\right] \tag{3.10}$$

is a loss augmented prediction.

After computing $s_i(\mathbf{w}_t)$ and $\boldsymbol{\psi}(x_i, s_i(\mathbf{w}_t))$ for all examples, the weight vector is updated

by minimizing a convex upper bound[2] $U(\mathbf{w}, \mathbf{w}_t)$ of the objective $E(\mathbf{w})$:

$$\begin{aligned}
&\mathbf{w}_{t+1} \\
&\quad = \operatorname*{argmin}_{\mathbf{w}} \frac{1}{2}\|\mathbf{w}\|^2 + C \sum_{i=1}^{n} \left[\max_{s \in \mathcal{S}(x_i)} \left[\mathbf{w} \cdot \boldsymbol{\psi}(x_i, s) + L_{\text{margin}}(y_i, s) \right] - \mathbf{w} \cdot \boldsymbol{\psi}(x_i, s_i(\mathbf{w}_t)) \right] \\
&\quad = \operatorname*{argmin}_{\mathbf{w}} U(\mathbf{w}, \mathbf{w}_t). \qquad (3.11)
\end{aligned}$$

It is easy to show that $U(\mathbf{w}, \mathbf{w}_t)$ is convex in $\mathbf{w}$. First, the regularization term $(1/2)\|\mathbf{w}\|^2$ is convex. The maximization inside the sum is over linear functions of $\mathbf{w}$, and thus is convex in $\mathbf{w}$. Finally, the term $-\mathbf{w} \cdot \boldsymbol{\psi}(x_i, s_i(\mathbf{w}_t))$ is convex because it is simply a linear function of $\mathbf{w}$. Putting all of these pieces together, we have a sum of convex functions, which itself must be convex. We call optimization problem Eq. 3.11 the *slave problem*.

To execute CCCP, we start from an initial weight vector $\mathbf{w}_0$ (or an initial choice of the $\boldsymbol{\psi}(x_i, s_i)$), and repeatedly solve the slave problem until a local optimum is reached [77].

Solving the slave problem with data mining

Our slave problem is similar to a structural SVM, and it can be solved using any of the techniques applicable to structural SVM optimization. Given the size of our training dataset and the nature of our inference algorithm, we opt to use a modified form of the data-mining procedure from [32] (the need for this will become clear in Section 3.1.3).

Data mining is an iterative procedure. In each data-mining iteration, we minimize an objective function that is restricted to a *cache*. If the cache is properly maintained, the sequence of solutions converges to the globally optimal solution for the *full* training set. We can thus use the data-mining procedure to compute an exact solution to a single instance of the slave problem.

2. The function U is written without a term that depends only on $\mathbf{w}_t$. Since the optimization is over $\mathbf{w}$, with $\mathbf{w}_t$ fixed, this term is constant and does not change the solution to the optimization problem in Eq. 3.11. Strictly speaking, without this constant, $U(\mathbf{w}, \mathbf{w}_t)$ is not necessarily an upper bound on E.

The cache-restricted slave problem. Recall that wish to compute

$$\begin{aligned} &\operatorname*{argmin}_{\mathbf{w}} U(\mathbf{w}, \mathbf{w}_t) \\ &= \operatorname*{argmin}_{\mathbf{w}} \frac{1}{2}\|\mathbf{w}\|^2 + C \sum_{i=1}^{n} \left[\max_{s \in \mathcal{S}(x_i)} \left[\mathbf{w} \cdot \boldsymbol{\psi}(x_i, s) + L_{\text{margin}}(y_i, s) \right] - \mathbf{w} \cdot \boldsymbol{\psi}(x_i, s_i(\mathbf{w}_t)) \right]. \end{aligned} \tag{3.12}$$

By analogy to a structural SVM, $s_i(\mathbf{w}_t)$ plays the role of the ground-truth label. However, unlike in a structural SVM, $s_i(\mathbf{w}_t)$ was not specified as ground truth in the training data. Rather, it is the *prediction* made by $\mathbf{w}_t$, for example i, in the presence of some loss adjustment. Due to this connection, we say that $s_i(\mathbf{w}_t)$ is "$\mathbf{w}_t$'s *belief* for example i" — it is what model $\mathbf{w}_t$ "believes" the output should be. The beliefs play an important role in the definition of the data-mining cache.

We define a *cache* $\mathcal{V}$ to be a set of (example, output) pairs. Each entry $(i, s) \in \mathcal{V}$ specifies an example index $i \in [1, n]$ and an output $s \in \mathcal{S}(x_i)$. Note that there may be many entries $(i, s) \in \mathcal{V}$ that share the same example i, but have different outputs s. Let $I(\mathcal{V}) \subseteq [1, n]$ be the set of example indices in $\mathcal{V}$, and let $\mathcal{S}(i, \mathcal{V}) \subseteq \mathcal{S}(x_i)$ be the set of outputs in $\mathcal{V}$ for example i.

We can write Eq. 3.12 so that it is *restricted to a cache* $\mathcal{V}$, provided that the cache satisfies one condition: If $i \in I(\mathcal{V})$, then $s_i(\mathbf{w}_t) \in \mathcal{S}(i, \mathcal{V})$. That is, if an example is in the cache, then the cache must contain that example's belief.

$$\begin{aligned} &\operatorname*{argmin}_{\mathbf{w}} U_{\mathcal{V}}(\mathbf{w}, \mathbf{w}_t) \\ &= \operatorname*{argmin}_{\mathbf{w}} \frac{1}{2}\|\mathbf{w}\|^2 + C \sum_{i \in I(\mathcal{V})} \left[\max_{s \in \mathcal{S}(i, \mathcal{V})} \left[\mathbf{w} \cdot \boldsymbol{\psi}(x_i, s) + L_{\text{margin}}(y_i, s) \right] - \mathbf{w} \cdot \boldsymbol{\psi}(x_i, s_i(\mathbf{w}_t)) \right] \end{aligned} \tag{3.13}$$

We can optimize the cache-restricted objective in a variety of ways. In practice, we've experimented with stochastic subgradient descent (SGD) [60, 63] and the quasi-Newton method LBFGS [50, 61]. Recently we've found that LBFGS consistently minimizes the objective to a lower value faster than our previous approach that used a highly-tuned SGD

implementation.[3] We speed up the gradient computations needed by LBFGS using the "margin-bound pruning" method described in Appendix B. This method reduces the amount of time LBFGS spends evaluating our objective function by about a factor of two.

Cache maintenance and the data-mining algorithm. To describe how the cache is maintained, we need to define a special cache that is equivalent to the full training dataset and then define the notions of *easy* and *hard* outputs.

Let $\mathcal{P} = \{(i, s) \mid i \in [1, n] \text{ and } s \in \mathcal{S}(x_i)\}$ be the set of all example indices and all of the possible outputs for each example. This cache corresponds to the full training set, and therefore $U_{\mathcal{P}}(\mathbf{w}, \mathbf{w}_t) = U(\mathbf{w}, \mathbf{w}_t)$. For any cache $\mathcal{V}$, let $\mathbf{w}^*(\mathcal{V}) = \text{argmin}_{\mathbf{w}} U_{\mathcal{V}}(\mathbf{w}, \mathbf{w}_t)$. We would like to find a small cache such that $\mathbf{w}^*(\mathcal{V}) = \mathbf{w}^*(\mathcal{P}) = \text{argmin}_{\mathbf{w}} U(\mathbf{w}, \mathbf{w}_t)$.

We define the *hard* outputs relative to the full training set cache $\mathcal{P}$, and a fixed model $\mathbf{w}_t$, as

$$\mathcal{H}(\mathbf{w}, \mathbf{w}_t) = \Big\{(i, s) \in \mathcal{P} \;\Big|\; s = \operatorname*{argmax}_{s' \in \mathcal{S}(i,\mathcal{P}) \setminus \{s_i(\mathbf{w}_t)\}} \mathbf{w} \cdot \boldsymbol{\psi}(x_i, s') + L_{\text{margin}}(y_i, s') \\ \text{and } \mathbf{w} \cdot \boldsymbol{\psi}(x_i, s_i(\mathbf{w}_t)) < \mathbf{w} \cdot \boldsymbol{\psi}(x_i, s) + L_{\text{margin}}(y_i, s)\Big\}. \quad (3.14)$$

In words, an output s is hard for example i, if the score of $\mathbf{w}_t$'s belief for i does not exceed s's score by the required margin $L_{\text{margin}}(y_i, s)$. Note that beliefs are never classified as hard by construction.

We define the *easy* outputs relative to a cache $\mathcal{V}$, and a fixed model $\mathbf{w}_t$, as

$$\mathcal{E}(\mathbf{w}, \mathbf{w}_t, \mathcal{V}) = \Big\{(i, s) \in \mathcal{V} \;\Big|\; \mathbf{w} \cdot \boldsymbol{\psi}(x_i, s_i(\mathbf{w}_t)) > \mathbf{w} \cdot \boldsymbol{\psi}(x_i, s) + L_{\text{margin}}(y_i, s)\Big\}. \quad (3.15)$$

An output s is easy for example i, if the score of $\mathbf{w}_t$'s belief for i is larger than s's by the required margin $L_{\text{margin}}(y_i, s)$. Note that since $L_{\text{margin}}(y, s) \geq 0$, beliefs are never classified as easy.

3. Although LBFGS is designed to optimize smooth functions, we have found, as have others, that it works well when applied to nonsmooth objectives involving hinge loss.

Now we describe the data-mining algorithm for computing $\mathbf{w}^*(\mathcal{P})$ given beliefs defined by a fixed model $\mathbf{w}_t$. The algorithm alternates between training a model and updating the cache. Let $\mathcal{V}_1$ be an initial cache. The only requirement is that $\mathcal{V}_1$ contains the belief outputs $(i, s_i(\mathbf{w}_t))$ for each example.

1. Let $\mathbf{w}' := \mathbf{w}^*(\mathcal{V}_j)$ [train a model using, *e.g.*, SGD or LBFGS]
2. If $\mathcal{H}(\mathbf{w}', \mathbf{w}_t) \subseteq \mathcal{V}_j$ stop and return $\mathbf{w}'$
3. Let $\mathcal{V}'_j := \mathcal{V}_j \setminus \mathcal{G}$ for any $\mathcal{G}$ such that $\mathcal{G} \subseteq \mathcal{E}(\mathbf{w}', \mathbf{w}_t, \mathcal{V}_j)$ [shrink the cache]
4. Let $\mathcal{V}_{j+1} := \mathcal{V}'_j \cup \mathcal{G}$ for any $\mathcal{G}$ such that $\mathcal{G} \cap (\mathcal{H}(\mathbf{w}', \mathbf{w}_t) \setminus \mathcal{V}_j) \neq \emptyset$ [grow the cache]

Following the proof in [32], it can be shown that this procedure terminates after a finite number of steps, and that when it terminates, it returns $\mathbf{w}^*(\mathcal{P})$.

Interleaving data mining with CCCP

The cache used to solve one instance of the slave problem can be *reused* to solve the next instance posed by CCCP more quickly than if starting from an empty cache. We maintain a single cache over the duration of CCCP. This leads to procedure `train-wl-ssvm`.

In each data-mining iteration we can only add a single output per example. The final cache might require several outputs for some examples (when multiple output scores tie). If the cache is emptied between each CCCP iteration, then it may be necessary to perform multiple data-mining passes over the dataset in order to extract all support vectors that participate in a tie. Maintaining a cache can keep tied support vectors across CCCP iterations, allowing data mining to converge faster.

A very basic cache maintenance policy is used on line 22 of `train-wl-ssvm`. This policy only keeps outputs in the cache that are either inside of, or within distance ϵ to, the margin. For data mining to converge, it is essential that all support vectors remain in the cache. The addition of ϵ increases robustness to numerical issues.

Data:
Training examples $\{(x_i, y_i), \ldots, (x_n, y_n)\}$
Initial model $\mathbf{w}$

Result: New model $\mathbf{w}$

$\mathcal{V} := \emptyset$
$\mathbf{w}_0 := \mathbf{w}_1 := \mathbf{w}$
$t := 1$
$\epsilon > 0$ `// a small positive constant, such as` 0.0001
repeat `// Outer CCCP iterations`
  for $i := 1$ **to** n **do**
    `Remove old belief` $(i, s_i(\mathbf{w}_{t-1}))$ `from` $\mathcal{V}$
    `Compute new belief` $s_i^* := \operatorname{argmax}_{s \in \mathcal{S}(x_i)} \left[\mathbf{w}_t \cdot \boldsymbol{\psi}(x_i, s) - L_{\text{output}}(y_i, s)\right]$
    `Add` (i, s_i^*) `to` $\mathcal{V}$
  end
  $\mathbf{w}' := \operatorname{argmin}_{\mathbf{w}} U_{\mathcal{V}}(\mathbf{w}, \mathbf{w}_t)$
  repeat `// Data mining to solve CCCP subproblem`
    for $i := 1$ **to** n **do**
      if $|\mathcal{V}| \geq$ *memory-limit* **then** `break`
      $s := \operatorname{argmax}_{s' \in \mathcal{S}(x_i)} \left[\mathbf{w}' \cdot \boldsymbol{\psi}(x_i, s') + L_{\text{margin}}(y_i, s')\right]$
      if $s \neq s_i^*$ **and** $\mathbf{w}' \cdot \boldsymbol{\psi}(x_i, s_i^*) - \mathbf{w}' \cdot \boldsymbol{\psi}(x_i, s) < L_{\text{margin}}(y_i, s) + \epsilon$ **then**
        `Add margin violator` (i, s) `to` $\mathcal{V}$
      end
    end
    $\mathbf{w}' := \operatorname{argmin}_{\mathbf{w}} U_{\mathcal{V}}(\mathbf{w}, \mathbf{w}_t)$
    `Remove easy outputs` (i, s) `with` $s \neq s_i^*$
      `and` $\mathbf{w}' \cdot \boldsymbol{\psi}(x_i, s_i^*) - \mathbf{w}' \cdot \boldsymbol{\psi}(x_i, s) > L_{\text{margin}}(y_i, s) + \epsilon$ `from` $\mathcal{V}$
  until *data mining converges*
  $w_{t+1} := \mathbf{w}'$
  $t := t + 1$
until *beliefs do not change*

Procedure train-wl-ssvm

The other advantage from reusing the cache comes when more sophisticated cache policies are used. Instead of keeping only those entries that are (nearly) support vectors, we can keep those entries, plus the top-K scoring entries found so far in data mining, up to some memory limit. At the start of each CCCP iteration, the current set of beliefs is updated. This can result in a large move in parameter space. Even though the additional top-K outputs were not support vectors with respect to the previous weight vector (which was induced by the previous set of beliefs), many of them may be support vectors with respect to the new set of beliefs. Keeping these "hot" outputs in the cache can reduce the number of data-mining iterations required for convergence.

3.1.3 Optimization specializations for object detection

The training problems that arise from object detection datasets have a particular structure that must be exploited to make the optimization procedure `train-wl-ssvm` efficient. An object detection dataset $\mathcal{P} = \{(x_i, y_i), \ldots, (x_n, y_n)\}$ is usually composed of two distinct types of examples: *foreground examples* and *background examples*. We write this decomposition as $\mathcal{P} = \mathcal{F} \cup \mathcal{B}$. Each foreground example comes from an image and a bounding box labeled with the foreground class. The background examples (for simplicity) come from images that do not contain the foreground class. Typically, each background image gives rise to a large number of background examples, and we can think of $\mathcal{B}$ as being further decomposed into $\mathcal{B} = \mathcal{B}(I_1) \cup \cdots \cup \mathcal{B}(I_B)$, where $\mathcal{B}(I_b)$ is the subset of background examples that come from background image I_b. Each background image may give rise to a large number (typically more than 250,000) of background examples. Our training problems often use over 2,000 background images, leading to *more half a billion background examples*. We outline four strategies that allow for efficient training using CCCP and data mining when faced with such large scale training sets.

Caching feature vectors. Rather than representing the cache as (image, output) pairs, we can instead maintain a cache of feature vectors and metadata about the feature vectors. This allows the convex optimization algorithm to work directly on feature vectors without having to first reconstruct them from a stored output, which would be too slow. For each feature vector, we store the example index that it came from and the L_{margin} loss associated with the output that generated it. We also store a flag that indicates if a feature vector is the belief for its example.

Image-level data mining. For background images, we can dramatically reduce the cost of computing the margin violating output, s, on line 15 of `train-wl-ssvm`, by processing whole images rather than individual background examples. This follows from the fact that the computation required during grammar inference can be shared between background examples that are nearby in the same image.

Lazy background caching. Next, we note that by making a few careful choices most background examples do not need to be explicitly represented in the data-mining cache. During training, we augment our output space $\mathcal{S}$ so that in addition to it containing all derivation trees, it also includes a special symbol $\perp$ that means "background" or "no foreground object here." We define the feature vector for the background output as $\boldsymbol{\psi}(x, \perp) = \vec{0}$, for all x. As a result, the score of the background output is $\mathbf{w} \cdot \boldsymbol{\psi}(x, \perp) = 0$ regardless of the example or $\mathbf{w}$. Additionally, we can set L_{output} so that it forces the belief for a background example to be $\perp$ (*i.e.*, we disallow "flipping" the label and treating a background example as foreground). With these two choices, we can avoid explicitly adding each background example to the data-mining cache. We adopt a lazy cache maintenance policy and only add background beliefs $s_i(\mathbf{w}) = \perp$ when there exists a margin violating output for example i. We can think of the other background beliefs as being implicitly represented in the cache (rather than explicitly storing a half-billion zero vectors).

Foreground data-mining approximation. Finally, we approximate the objective U in a way that lowers the cost of data mining on the foreground examples. Data mining for the foreground examples would be slow because it requires performing relatively expensive inference (more than 1 second per image) on thousands of images during each data mining iteration.[4] Instead of applying data mining to the foreground examples, each time we compute $s_i(\mathbf{w}_t)$ for a foreground example (only once per outer CCCP iteration), we also compute the top M scoring outputs $s \in \mathcal{S}(x_i)$ of $\mathbf{w}_t \cdot \boldsymbol{\psi}(x_i, s) + L_{\text{margin}}(y_i, s)$, and place the corresponding feature vectors in the data mining cache. This is efficient since much of the required computation is shared with computation already necessary for computing $s_i(\mathbf{w}_t)$. While this is only a heuristic approximation to true data mining, it leads to an improvement over training with binary LSVM (see Section 5.5). In practice, we find that $M = 1$ is sufficient for improved performance and that increasing M beyond 1 does not improve results.

3.1.4 Relationship to latent SVM

One motivation for using WL-SSVM instead of LSVM is shown in Figures 3.1 and 3.2. At training time, the LSVM objective pushes one derivation (illustrated to the left and right of the training image in Figure 3.1 (top)) to score ≥ 1 on each positive training instance. At test time — as a direct result of this objective — all derivations may produce similar scores regardless of whether a person is partially or fully visible. If the wrong derivation fires, a false positive will likely be generated due to low overlap. There may be many false positives due to the wrong derivation winning, which would diminish the gains we hoped to achieve by using a richer model. This example illustrates a general phenomenon: as models become richer, training their parameters often becomes more challenging. WL-SSVM addresses this deficiency by attempting to achieve a margin between the score of the correct derivation

4. In contrast, we typically use a small number (*e.g.*, 200) of background images during most of training. We do a final round of data mining with the full set of background images

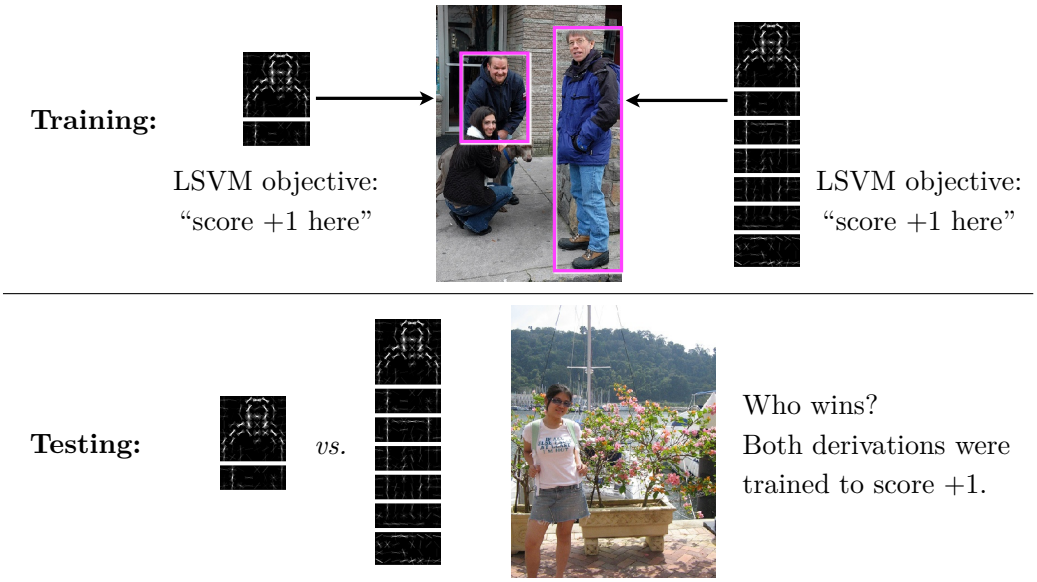


Figure 3.1: An illustration of how latent SVM training can create ambiguity at test time.

(output) and the score of all incorrect derivations (outputs). Figure 3.2 illustrates WL-SSVM training on the same example image.

We can also show that WL-SSVM is strictly more general than LSVM by formulating LSVM as a special case of WL-SSVM. We also note a connection between the coordinate descent optimization algorithm for LSVM given in [32] and the CCCP algorithm given above.

We can construct the latent SVM from [32] using WL-SSVM in the following way. Let the label space be binary, so $\mathcal{Y} = \{\pm 1\}$. Let $\mathcal{Z}$ be the space of latent values used by the latent SVM, and let the output space be $\mathcal{S} = \mathcal{Y} \times \mathcal{Z}$. An output $s = (y, z)$ is a (label, latent value) pair.

The latent SVM objective is

$$E_{\text{LSVM}}(\mathbf{w}) = \frac{1}{2}\|\mathbf{w}\|^2 + C \sum_{i=1}^{n} \max \left[0, 1 - y_i \max_{z \in \mathcal{Z}(x_i)} \mathbf{w} \cdot \boldsymbol{\psi}(x_i, z) \right]. \tag{3.16}$$

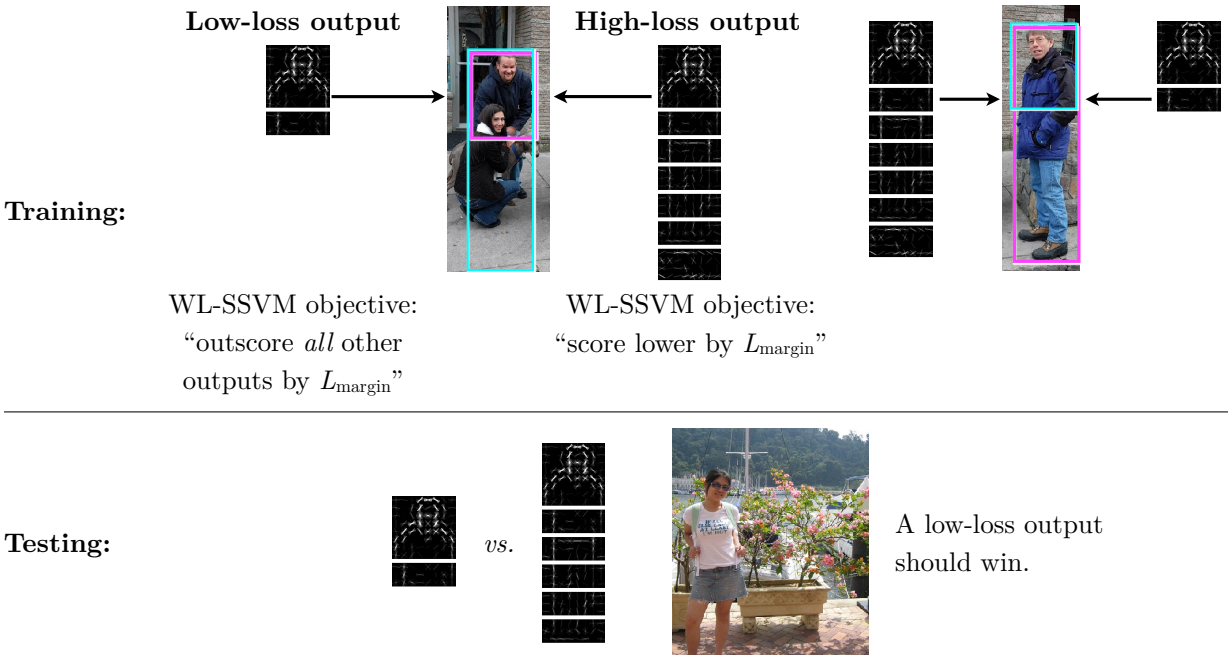


Figure 3.2: An illustration of training and testing with WL-SSVM. The WL-SSVM objective attempts to create a margin between the scores of high-loss outputs and low-loss outputs. If successful, training should result in better predictions.

We can rewrite E_{LSVM} as a WL-SSVM

$$\frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{n}\left[\max_{(y,z)\in\mathcal{S}(x_i)} \mathbf{w}\cdot\boldsymbol{\psi}(x_i,(y,z)) + L_{\text{margin}}(y_i,(y,z))\right] - \left[\max_{(y,z)\in\mathcal{S}(x_i)} \mathbf{w}\cdot\boldsymbol{\psi}(x_i,(y,z)) - L_{\text{output}}(y_i,(y,z))\right], \quad (3.17)$$

with the following definitions.

1. $\boldsymbol{\psi}(x,(-1,z)) = \vec{0}$ for all x and z
2. $L_{\text{output}}(y,(y',z)) = 0$ if $y = y'$ and $z \in \mathcal{Z}(x)$, and $-\infty$ otherwise
3. $L_{\text{margin}}(y,(y',z)) = 0$ if $y = y'$, and 1 otherwise

Note that L_{output} plays the role of enforcing that the latent values z are in the valid set $\mathcal{Z}(x)$, and that the positive/negative training labels are respected. By case analysis we can show that Eq. 3.17 (with the provided definitions) is equivalent to Eq. 3.16.

In [32], we applied a coordinate descent optimization algorithm to the latent SVM objective in Eq. 3.16. By rewriting the LSVM objective in the form of a WL-SSVM the decomposition of E_{LSVM} into a sum of convex and concave functions becomes immediate. This suggests that an alternative optimization approach is to use CCCP to find a local optimum, instead of coordinate descent. The main difference between the two approaches is that when using CCCP, we would need to data mine over the positive examples, not just over the negative examples as in [32].

In the previous section we introduced a heuristic approximation to data mining over the positives that is controlled by reducing the size of the data-mining space for positive examples. When we set the size of this space to $M = 0$, we eliminate data mining over positives altogether. Note that setting $M = 0$ transforms the CCCP optimization algorithm for latent SVM into the coordinate descent optimization algorithm for latent SVM.

3.2 Related work

There has been recent interest in applying structured prediction frameworks to object detection [7, 8, 69]. In [7], Blaschko and Lampert apply structural SVM to learning bag of visual words models for object detection. They search over all possible image rectangles, using a branch-and-bound algorithm, making the label space equal to the output space as is required by structural SVM. However, they assume that each image contains exactly one instance of the target category. In their experiments, they train and test on the PASCAL 2006 dataset, though they do not describe how they address the issue of multiple object instances per image during training.

In [69], Vedaldi and Zisserman apply latent structural SVM to HOG-based models. Following our work, they use mixtures of part-based models (Chapter 4). Each component is composed of a small number of parts (*e.g.*, 4) that are allowed to shift in a tight neighborhood without incurring a deformation penalty. Their models are single resolution and do not use a root filter. In their application of latent structural SVM, they define the *pose* of an object

to be a specification of translation, scale, part placement, and mixture component. They define a space of latent values that allows for small pose adjustments relative to reference poses that are estimated from the training labels. By limiting the latent space to small adjustments, they prevent the latent poses that are inferred during training from being "far away" from the training label. To define a reference pose for each example, they map each training label (bounding box and aspect) to an estimated object pose. The details of this estimation procedure are not given in the paper.

WL-SSVM, in contrast, handles these issues in a more general and flexible way. First, WL-SSVM uses a loss, L_{output}, between outputs and labels to prevent inferring latent configurations that are "far away" from the label. Note that here "far away" is defined explicitly by the loss function, rather than implicitly by limiting the latent space to small adjustments. Our approach enables the use of a larger, more general space of latent values that is adapted to each training instance through the loss function. Secondly, our framework does not require mapping each training label into the latent space in order to define a reference pose (or output, more generally). Instead, it is more natural to define L_{output} by providing a mapping from strong outputs to weak labels. Such a mapping is typically required at test-time, and the loss can be defined intuitively by thinking about the relationship between the model's output and the test time task. The estimation in [69], in contrast — which goes from labels to outputs — requires inferring missing information before training a model to predict that information.

In [8], Blaschko *et al.* apply a ranking generalization of latent structural SVM to object detection with bag of visual words and HOG-based detection models. Their formulation uses a special case of the output-label decoupling in WL-SSVM, where they consider a hard compatibility relation between weak labels and outputs. Our framework is more general because it also allows for soft relations.

CHAPTER 4

MIXTURE MODELS

In this chapter, we develop object detection models that are mixtures of several component models, where each component is a star-structured deformable part model. We start in Section 4.1 by redefining the star models of [34] and [32] in the object detection grammar framework. Next, we extend these simple grammars so that they can represent category subtypes. The resulting models can be thought of as mixtures of several star models. We consider mixtures models that arise from an initial clustering of the foreground training examples by aspect ratio. This simple cue reliably produces effective mixture models where each component model typically corresponds to a canonical view or pose within each object category. We then present a method for automatically learning asymmetric component models with a latent left-*versus*-right orientation. After describing these models, we turn to how their parameters are learned from data.

Learning mixture models involves two high-level tasks. The first task is to determine the model's structure. This structure includes the number of components, M, and for each component the size and aspect ratio of its root filter, the size, aspect ratio, and number of part filters to use, and their anchor positions relative to the root. The second task is to learn the model parameters given the fixed structure. In practice, these two tasks are interwoven because we have found that it is advantageous to learn the structure and parameters of simpler models (*e.g.*, root-filter-only models), before using them to initialize more complex models (*e.g.*, models with parts). We also introduce an alternative form of regularization — that we call "max component regularization" — to the LSVM and WL-SSVM frameworks.

In Section 2.2.3 we described in very general terms how the score function for an object detection grammar can be written as a dot product between a vector of model parameters and an appropriately constructed feature vector. To make this procedure more concrete, we carefully show the process for star-structured grammar models and mixtures thereof.

4.1 Star-structured deformable part models

The Dalal-Triggs detector [18], which won the 2006 PASCAL object detection challenge, used a single filter on histogram of oriented gradients (HOG) features to represent an object category. This detector uses a sliding window approach, where a filter is applied at all positions and scales of an image. We can think of the detector as a classifier which takes as input an image, a position within that image, and a scale. The classifier determines whether or not there is an instance of the target category at the given position and scale. Since the model is a simple filter we can compute a score as $\mathbf{f} \cdot \boldsymbol{\psi}(I, \omega)$ where $\mathbf{f}$ is a vector of the filter weights, I is an image with a specified position and scale, ω, and $\boldsymbol{\psi}(I, \omega)$ is a feature vector. A major innovation of the Dalal-Triggs detector was the construction of particularly effective features.

Our star-structured deformable part models are defined by a coarse root filter that approximately covers an entire object (analogous to the Dalal-Triggs filter) and higher resolution part filters that cover smaller parts of the object. Figure 2.1 illustrates an instantiation of such a model in a feature pyramid. The root filter location defines a detection window (the pixels contributing to the part of the feature map covered by the filter). The part filters are placed λ levels (one octave) below in the pyramid, so the features at that level are computed at twice the resolution of the features in the root filter level.

We have found that using higher resolution features for defining part filters is essential for obtaining high recognition performance. With this approach the part filters capture finer resolution features that are localized to greater accuracy when compared to the features captured by the root filter. Consider building a model for a face. The root filter could capture coarse resolution edges such as the face boundary while the part filters could capture details such as eyes, nose and mouth.

4.1.1 Reformulation as a simple grammar

In the following, we use the object detection grammar notation and definitions from Section 2.1. Our star models can be defined as very simple object detection grammars. We start by declaring a start symbol for the grammar, Q, which can be thought of as representing the target object category (*e.g.*, *car*). To add a root filter, we declare a terminal symbol A and associate it with a filter F_0. For a star model with N parts, we need to declare a nonterminal that represents the canonical (nondeformed) instantiation of each part. Let these canonical part nonterminals be $Y_1, \ldots, Y_N$. Now, we specify a set of productions, using a structural schema, that composes Q out of the root and parts.

$$\forall \omega \in \Omega : Q(\omega) \xrightarrow{\beta} \{\ A(\omega), Y_1(\omega \oplus \delta_1), \ldots, Y_N(\omega \oplus \delta_N)\ \} \tag{4.1}$$

The structural schema has a bias β and canonical "anchor" offsets for each part. Each anchor offset has the form $\delta = (\delta_x, \delta_y, \delta_l)$. The operator $\oplus$ maps $(x, y, l) \times (\delta_x, \delta_y, \delta_l) \in \Omega \times \Delta$ to $(x + 2^{\delta_l}\delta_x, y + 2^{\delta_l}\delta_y, l + \delta_l\lambda)$. We focus on models that constrain the part filters to be one octave below the root filter, and, accordingly, we fix $\delta_l = 1$ for each δ_i.

The appearance model for each part is still undefined. To specify part appearance, we create terminals $B_1, \ldots, B_N$ and associate them with filters $F_1, \ldots, F_N$. To complete the grammar, we connect each part terminal to its part nonterminal through a deformation schema.

$$\begin{aligned} &\forall (\omega, \delta, n) \in \Omega \times \Delta_{\delta_l = 0} \times [1, N] : \\ &\qquad Y_n(\omega) \xrightarrow{-\mathbf{d}_n \cdot \boldsymbol{\phi}_\Delta(\delta)} \{\ B_n(\omega \oplus \delta)\ \} \end{aligned} \tag{4.2}$$

The set of productions generated by these deformation schemas allow for derivation trees that place the part filters at displaced locations relative to the root. We restrict the displacements, δ, to the set $\Delta_{\delta_l=0} = \Big\{(\delta_x, \delta_y, \delta_l) \in \Delta \mid \delta_l = 0\Big\}$. The parameters $\mathbf{d}_n$ define the coefficients of a quadratic deformation cost for displacing part n. Intuitively, this parameterization

assigns a large negative bias to a rule that places a part terminal far away from the part nonterminal's canonical location. Putting these pieces together, we have Grammar 1, which defines star-structured deformable part models.

Grammar 1 star-grammar (with N parts)

Terminals and nonterminals:

$$\mathcal{T} = \{A, B_1, \ldots, B_N\} \tag{4.3}$$

$$\mathcal{N} = \{Q, Y_1, \ldots, Y_N\} \tag{4.4}$$

Structural schemas:

$$\forall \omega \in \Omega : Q(\omega) \xrightarrow{\beta} \{\ A(\omega), Y_1(\omega \oplus \delta_1), \ldots, Y_N(\omega \oplus \delta_N)\ \} \tag{4.5}$$

Deformation schemas:

$$\forall (\omega, \delta, n) \in \Omega \times \Delta_{\delta_l = 0} \times [1, N] :$$

$$Y_n(\omega) \xrightarrow{-\mathbf{d}_n \cdot \boldsymbol{\phi}_\Delta(\delta)} \{\ B_n(\omega \oplus \delta)\ \} \tag{4.6}$$

In this simple grammar, all derivation trees have the same graph structure, their internal nodes are labeled with the same production schemas, and their leaf nodes are labeled with the same terminals. The derivation trees differ only in the schema parameters that label the internal nodes (*i.e.*, the placement of Q and the (placement, displacement) pair of each part nonterminal) and the placement labels in the leaf nodes (*i.e.*, where each terminal/filter is placed). To help elucidate the equivalence between the non-grammar formulation of star models in [32] and the grammar formulation here, we can think of each derivation tree as an object hypothesis that specifies the location of each filter in the model in a feature pyramid, $z = (\omega_0, \ldots, \omega_N)$, where $\omega_n = (x_n, y_n, l_n) \in \Omega$ is the position and level of the n-th filter. Note that the grammar forces the level of each part filter to be at the level that was computed at twice the resolution of the root level, *i.e.*, $l_n = l_0 - \lambda$ for $n > 0$.

The score of a hypothesis (derivation tree) in an image I is given by the bias minus a deformation cost that depends on the relative position of each part with respect to the root,

plus the scores of each filter at their respective locations.

$$\underbrace{\text{score}(I, \omega_0, \ldots, \omega_N)}_{\text{score}(I,T)} = \underbrace{\beta - \sum_{n=1}^{N} \mathbf{d}_n \cdot \boldsymbol{\phi}_\Delta(\delta\omega_n)}_{\sum_{(r,z)\in\text{int}(T)} \beta_r(z)} + \underbrace{\sum_{n=0}^{N} \mathbf{f}_n \cdot \boldsymbol{\phi}_{\text{app}}(H_I, F_n, \omega_n)}_{\sum_{A(\omega)\in\text{leaf}(T)} \text{score}(I,A,\omega)}, \tag{4.7}$$

where

$$\delta\omega_n = \omega_n - 2(\omega_0 \oplus \delta_n) \tag{4.8}$$

gives the displacement of the n-th part relative to its anchor position. The correspondence between the general formulation of the score of a derivation tree, given in Eq. 2.7, and the simplified hypothesis score above is shown in the underbraces.

Note that if $\mathbf{d}_n = (a_x, b_x, a_y, b_y)^\intercal$ with $a_x = a_y = 1$, then deformation cost for the n-th part is the squared distance between its actual position and its anchor position relative to the root (assuming we use the deformation feature function in Eq. 2.14). In practice, we constrain the quadratic coefficients a_x and a_y to be positive. Below we will discuss in more detail why this constraint is necessary to avoid instabilities in the learning algorithm.

As described in Section 2.2.3 in full generality, the score of a hypothesis (derivation tree) z in image I can be expressed in terms of a dot product, $\mathbf{w} \cdot \boldsymbol{\psi}(I, z)$, between a vector of model parameters $\mathbf{w}$ and a feature vector $\boldsymbol{\psi}(I, z)$. Concretely, in the case of star grammars, this construction takes the form

$$\mathbf{w} = (\beta, \mathbf{d}_1^\intercal, \ldots, \mathbf{d}_N^\intercal, \mathbf{f}_0^\intercal, \ldots, \mathbf{f}_N^\intercal)^\intercal. \tag{4.9}$$

$$\begin{aligned}\boldsymbol{\psi}(I, z) = (1, &-\boldsymbol{\phi}_\Delta(\delta\omega_1)^\intercal, \ldots, -\boldsymbol{\phi}_\Delta(\delta\omega_N)^\intercal, \\ &\boldsymbol{\phi}_{\text{app}}(H_I, F_0, \omega_0)^\intercal, \ldots \boldsymbol{\phi}_{\text{app}}(H_I, F_N, \omega_N)^\intercal)^\intercal.\end{aligned} \tag{4.10}$$

4.1.2 Detection with star grammars

In Section 2.1.2 we gave a general dynamic programming algorithm for computing high scoring derivations in an image given an object detection grammar. To illustrate the procedure in more detail, here we shows the steps of the algorithm for the special case of star grammars.

To detect objects in an image we compute an overall score for each root location according to the best possible placement of the parts,

$$\text{score}(I, \omega_0) = \max_{\omega_1,\ldots,\omega_N} \text{score}(I, \omega_0, \omega_1, \ldots, \omega_N). \tag{4.11}$$

High-scoring root locations define detections while the locations of the parts that yield a high-scoring root location define a full object hypothesis. Translating Eq. 4.11 into the dynamic programming algorithm in Section 2.1.2, and using the grammar specified in Grammar 1, we have that $\text{score}(I, \omega_0) = S_Q[\omega_0]$, and so our goal is to compute the score table S_Q.

First, we compute the score tables for the terminals, $S_A, S_{B_1}, \ldots, S_{B_N}$ by convolving their filters with the HOG feature pyramid. Examples of these score tables are visualized as the "response of root filter" and "responses of part filters" in Figure 4.1. Given these score tables, we can then compute the score tables for the part nonterminals S_{Y_n}.

$$\begin{aligned} S_{Y_n}[\omega] &= \max_{Y_n(\omega) \xrightarrow{-\mathbf{d}_n \cdot \boldsymbol{\phi}_\Delta(\delta)} \{\ B_n(\omega \oplus \delta)\ \}} -\mathbf{d}_n \cdot \boldsymbol{\phi}_\Delta(\delta) + S_{B_n}[\omega \oplus \delta] \\ &= \max_{\delta \in \Delta} -\mathbf{d}_n \cdot \boldsymbol{\phi}_\Delta(\delta) + S_{B_n}[\omega \oplus \delta] \end{aligned} \tag{4.12}$$

For each table entry, $S_{Y_n}[\omega]$, the maximization is over all $\delta \in \Delta$. A brute force algorithm would take $O(|\Omega||\Delta|)$ time per symbol Y_n. If $|\Delta| \approx |\Omega|$, then brute force search is too costly and generalized distance transforms (see [29, 30] for details) can be used to compute this maximization over all $\omega \in \Omega$ in $O(|\Omega|)$ time. Alternatively, if $|\Delta|$ is small (*i.e.*, bounded by a constant that is independent of Ω), then brute force maximization is an effective strategy.

In our applications, Δ can be made quite small with no impact on average precision performance. For example, we can restrict Δ to only include shifts of up to ± 4 HOG cells, leading to $|\Delta| = 81$. Furthermore, if separable cost functions are used, then the brute force maximization is separable, which further reduces the number of operations from $|\Delta|$ to $2\sqrt{|\Delta|}$. The part nonterminal score tables, S_{Y_n}, appear as spread out versions of the part terminal score tables. Examples are visualized as "transformed responses" in Figure 4.1.

After computing the part score tables, the start symbol's score table is computed.

$$S_Q[\omega] = \max_{Q(\omega) \xrightarrow{\beta} \{\ A(\omega), Y_1(\omega \oplus \delta_1), \ldots, Y_N(\omega \oplus \delta_N)\ \}} \beta + S_A[\omega] + \sum_{n=1}^{N} S_{Y_n}[\omega \oplus \delta_n]$$
$$= \beta + S_A[\omega] + \sum_{n=1}^{N} S_{Y_n}[\omega \oplus \delta_n] \tag{4.13}$$

Since there is only one schema with $Q(\omega)$ on the left-hand side, the maximization disappears. The resulting operation is simply the summation of the root filter's score table, S_A, with the appropriately shifted part score tables. This operation is illustrated by the "$\oplus$" in Figure 4.1.

4.2 Mixture models

A mixture model with M components is defined as the disjunction of M grammar models. Grammar 2 gives the structural and deformation schemas for a mixture model where the m-th component is a star model with N_m parts.

Grammar 2 mixture-star-grammar (with M components)

Terminals and nonterminals:

$$\mathcal{T} = \{A_1, \ldots, A_M\} \cup \{B_{1,1}, \ldots, B_{1,N_1}\} \cup \cdots \cup \{B_{M,1}, \ldots, B_{M,N_M}\} \tag{4.14}$$
$$\mathcal{N} = \{Q\} \cup \{Y_{1,1}, \ldots, Y_{1,N_1}\} \cup \cdots \cup \{Y_{M,1}, \ldots, Y_{1,N_M}\} \tag{4.15}$$

Structural schemas:

$$\forall (\omega, m) \in \Omega \times [1, M]:$$
$$Q(\omega) \xrightarrow{\beta_m} \{\ A_m(\omega), Y_{m,1}(\omega \oplus \delta_{m,1}), \ldots, Y_{m,N_m}(\omega \oplus \delta_{m,N_m})\ \} \tag{4.16}$$

Deformation schemas:

$$\forall (\omega, \delta, m) \in \Omega \times \Delta_{\lambda=0} \times [1, M]:$$
$$\forall n \in [1, N_m]:$$
$$Y_{m,n}(\omega) \xrightarrow{-\mathbf{d}_{m,n} \cdot \boldsymbol{\phi}_\Delta(\delta)} \{\ B_{m,n}(\omega \oplus \delta)\ \} \tag{4.17}$$

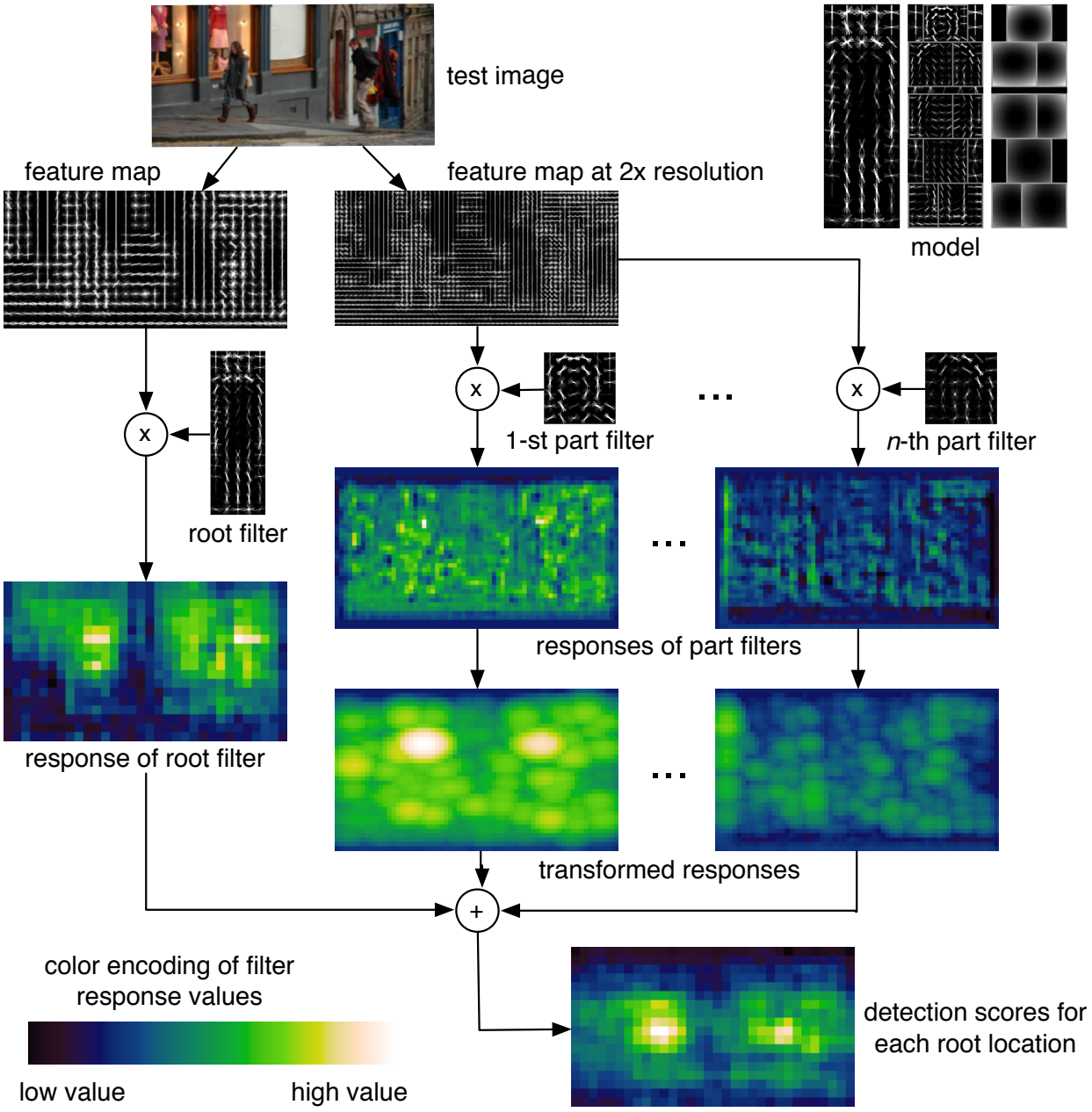


Figure 4.1: The detection process at one scale. Responses from the root and part filters are computed a different resolutions in the feature pyramid. The transformed responses are combined to yield a final score for each root location. We show the responses and transformed responses for the "head" and "right shoulder" parts. Note how the "head" filter is more discriminative. The combined scores clearly show two good hypothesis for the object at this scale.

This grammar has M root filters, $F_{m,0}$, associated with the root terminals A_m, and part filters, $F_{m,n}$ associated with the part terminals $B_{m,n}$. Note that there are now M alternative productions with $Q(\omega)$ on the left-hand side — one for each mixture component. Each choice has a different bias β_m associated with it. These biases provide an important, though limited, mechanism for calibrating the scores of competing derivations. These issues are discussed more in Section 4.5.

A derivation tree generated by one of these mixture grammars has a root node that is labeled with the structural schema in Eq. 4.16 and schema parameters specifying where Q is placed and which mixture component is instantiated. Beyond this difference, the information in the rest of the derivation tree specifies where the selected component's part nonterminals and terminals are instantiated, just as before. We can write the model configuration described by a derivation tree that instantiates component m concisely as $z = (m, \omega_0, \ldots, \omega_{N_m})$. The score of this hypothesis is the score of the hypothesis $z' = (\omega_0, \ldots, \omega_{N_m})$ for the m-th model component.

As in the case of a single component model, the score of a hypothesis for a mixture model can be expressed by a dot product between a vector of model parameters $\mathbf{w}$ and a vector $\boldsymbol{\psi}(I, z)$. For a mixture model the vector $\mathbf{w}$ is the concatenation of the model parameter vectors for each component. The vector $\boldsymbol{\psi}(I, z)$ is sparse, with nonzero entries defined by $\boldsymbol{\psi}(I, z')$ in the block of vector indices matching the block used by $\mathbf{w}_m$ in $\mathbf{w}$,

$$\mathbf{w} = (\mathbf{w}_1^\mathsf{T}, \ldots, \mathbf{w}_m^\mathsf{T}, \ldots, \mathbf{w}_M^\mathsf{T})^\mathsf{T} \tag{4.18}$$

$$\boldsymbol{\psi}(I, z) = (0, \ldots, 0, \underbrace{\boldsymbol{\psi}(I, z')^\mathsf{T}}_{\text{block for } \mathbf{w}_m}, 0, \ldots, 0)^\mathsf{T} \tag{4.19}$$

With this construction $\mathbf{w} \cdot \boldsymbol{\psi}(I, z) = \mathbf{w}_m \cdot \boldsymbol{\psi}(I, z')$.

To detect objects using a mixture model, we apply the same dynamic programming algorithm. Now, when computing the score table for the start symbol, the maximization does not disappear. Instead, the maximization is over the score of each top-level production

from Q:

$$S_Q[\omega] = \max_{m \in [1,M]} \beta_m + S_{A_m}[\omega] + \sum_{n=1}^{N_m} S_{Y_{m,n}}[\omega \oplus \delta_{m,n}]. \tag{4.20}$$

4.2.1 Initializing mixture models

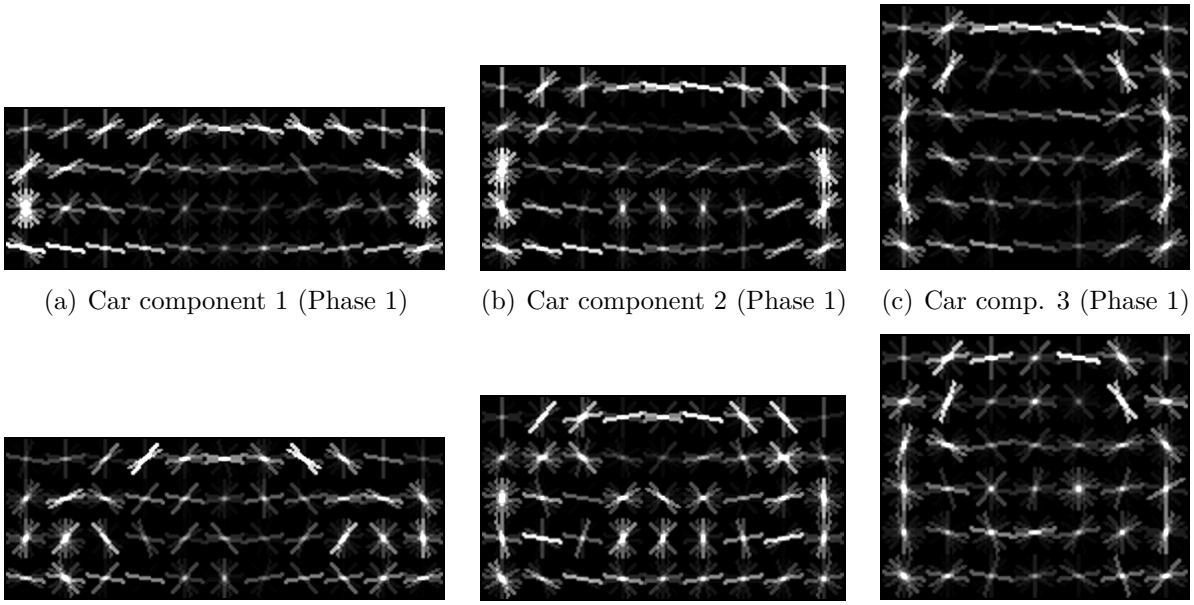
The weak-label structural SVM training algorithm is susceptible to local minima and thus is sensitive to initialization. This is a common limitation of other methods that use latent information as well. We initialize and train mixture models in three phases as follows.

Phase 1: Initializing root filters. Let $\mathcal{P}$ be the set of (image, bounding box) pairs for the foreground examples in the training set. For a mixture model with M components, we sort the bounding boxes in $\mathcal{P}$ by their aspect ratio and split them into M groups of equal size $\mathcal{P}_1, \ldots, \mathcal{P}_M$. Aspect ratio is used as a simple indicator of extreme intraclass variation. We train M different root filters $F_1, \ldots, F_M$, one for each group of positive bounding boxes.

To define the dimensions of F_m we select the mean aspect ratio of the boxes in $\mathcal{P}_m$ and the largest area not larger than 80% of the boxes. This ensures that for most pairs $(I, B) \in \mathcal{P}_m$ we can place F_m in the feature pyramid of I so it significantly overlaps with B.

We train F_m using a standard SVM, with no latent information, as in [18]. For $(I, B) \in \mathcal{P}_m$ we warp the image region under B so its feature map has the same dimensions as F_m. This leads to a positive example. We select random subwindows of appropriate dimension from images that does not contain the target object class to define negative examples. Figure 4.2(a-c) shows the result of this phase when training a three component car model.

Phase 2: Merging components. We combine the initial root filters into a mixture model with no parts and retrain the parameters of the combined model on the full (unsplit and without warping) dataset. In this case the component label and root location are the only latent variables for each example. The WL-SSVM training algorithm can be thought of as a discriminative clustering method that alternates between assigning cluster (mixture) labels



(a) Car component 1 (Phase 1) (b) Car component 2 (Phase 1) (c) Car comp. 3 (Phase 1)

(d) Car component 1 (Phase 2) (e) Car component 2 (Phase 2) (f) Car comp. 3 (Phase 2)

Figure 4.2: Top row: The three root filters of a car mixture model after Phase 1 of the initialization process (classical SVM training with "warped" foreground examples). Bottom row: The three root filters of a car mixture model after Phase 2 of the initialization process (mixture-of-roots training with LSVM or WL-SSVM). Note how the filters have become sharper and contain more shape detail than after Phase 1.

for each foreground example and estimating cluster "means" (root filters). Figure 4.2(d-f) shows the result of this phase. Some structures, such as wheels, are now visible in the filters.

Phase 3: Initializing part filters. We initialize the parts of each component using a simple heuristic. We fix the number of parts per component (*e.g.*, 8), and using parts of fixed extend (*e.g.*, 6×6) we greedily place parts to cover high-energy regions of the root filter.[1] A part is either anchored along the central vertical axis of the root filter, or it is off-center and has a symmetric part on the other side of the root filter. Once a part is placed, the energy of the covered portion of the root filter is set to zero, and we look for the next highest-energy region, until all parts are chosen.

The part filters are initialized by interpolating the root filter subarray covered by each part to twice the root's spatial resolution. After extracting and upsampling a part filter, we normalize it so it has unit ℓ_2 norm and then multiply it by 0.1. By making the filter norms small, the component label and latent position selected by the root filter is typically the same before and after adding parts. We have found that this helps stabilize the training procedure: when parts are added their appearance models are weak and allowing them to dramatically change the inferred latent configurations on the foreground examples is dangerous. The deformation parameters for each part are initialized to $\mathbf{d}_i = (0.01, 0, 0.01, 0)^\intercal$. This choice encourages the parts to be placed fairly close to their anchor positions. The resulting model serves as the initial model for the last round of CCCP parameter learning.

After greedily selecting initial part locations, we use local search to move the parts, one at a time in random order, to maximize the total amount of root filter energy that is covered. When a better covering cannot be found, this phase is restarted from the initial part configuration to see if more energy can be covered by repositioning the parts in a different order. After several restarts the part configuration that covers the most energy is selected.

Figure 4.3(a,c,e) shows the result of initializing the parts of a three component car model.[2]

1. The "energy" of a region is defined by the norm of the positive weights in a subwindow.

2. Recall that we make the initial part filter norms small. For visualization purposes, the part filter

As expected, the initial parts are blurry due to interpolation. Nevertheless, they contain enough structure to lock onto salient image features. The final car model is shown in Figure 4.3(b,d,f). The part filters have become sharper, the deformation models are tighter, and, interestingly, the root filters look quite different from the initial ones. The root filters change for a couple of reasons: 1) they must compromise with where the parts would like to be placed, and 2) regularization pulls energy out of the root filter and into the part filters.

4.3 Latent orientation

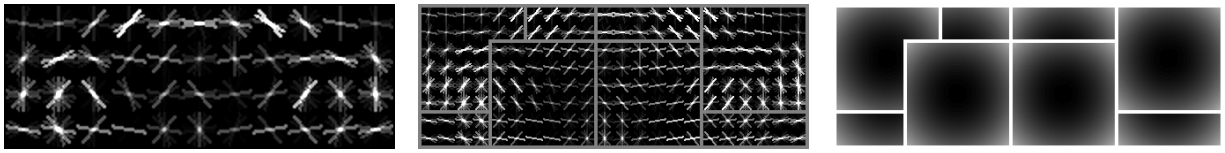
It is potentially difficult to find a semantic interpretation of the component models that result from weakly supervised training. Nevertheless, for many object categories (*e.g.*, bicycle, car, horse, bus, *etc.*) it is evident that the components represent different ranges of out-of-plane rotation. This is natural since aspect ratio is strongly correlated with out-of-plane rotation for nonradially symmetric objects. One notable exception is the PASCAL person category, for which the learned components correspond to the dominate occlusions patterns in the dataset.

One failure mode that results from building mixture models based on aspect ratio is that opposite-facing instances (*e.g.*, a pair of left and right-facing side views of horses) have the same aspect ratio and are typically modeled by the same component. This leads to averaging the appearance model of the two orientations. In the case of horses, this creates a side-view component ideally suited for detecting a horse with two heads (Figure 1.3).

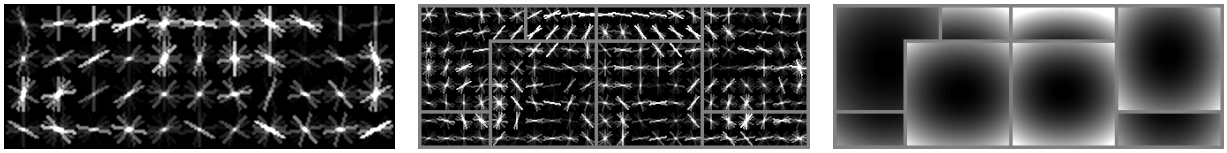
To address this problem, we enrich the mixture models from the previous section so that they explicitly model asymmetric components with latent left-right orientation. To do this, we double the number of top-level structural schemas in the grammar so that for each of the M components, the grammar has a "left-facing" production schema and a "right-facing"

---

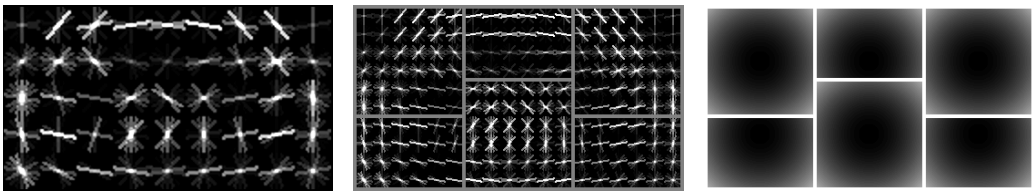
weights have been amplified to reveal structure. This amplification is applied only to the *initial filters*. The trained part filters are shown with their actual weight magnitudes.



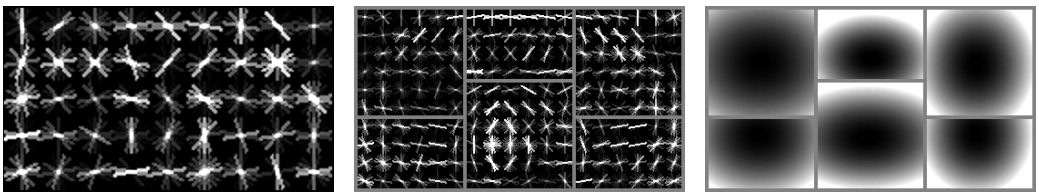
(a) Car component 1 (initial parts)



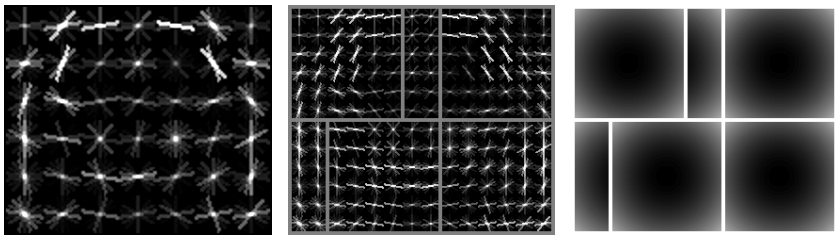
(b) Car component 1 (trained parts)



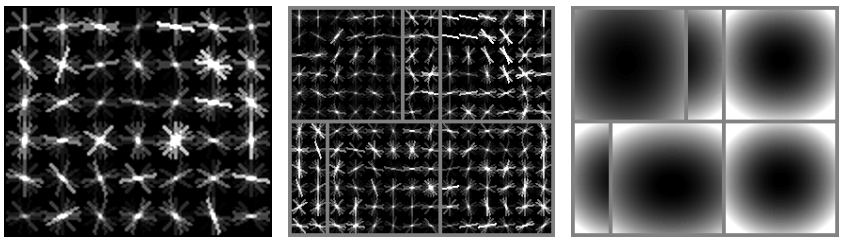
(c) Car component 2 (initial parts)



(d) Car component 2 (trained parts)



(e) Car component 3 (initial parts)



(f) Car component 3 (trained parts)

Figure 4.3: Car components with parts initialized by interpolated the root filter to twice its resolution (a,c,e), and parts after training with LSVM or WL-SSVM (b,d,f).

production schema.[3] To enforce the left-right orientation relationship, we tie all model parameters between each set of paired schemas.

Filter are tied by analytically "flipping" their parameters. Using the fact the HOG features are histograms over gradient orientations, flipping the parameters inside one filter cell amounts to a permutation of the parameters. This permutation swaps each orientation with its vertically mirrored equivalent. To flip an entire filter, first each cell is flipped, and then the cells are swapped across the filter's vertical axis. Quadratic deformation models are flipped by simply changing the sign of the linear coefficient for the horizontal direction.

4.3.1 Orientation clustering

In contrast to the previous section, now we must take special care when selecting the subset of examples used to initialize root filters. Ideally, each subset should contain examples with only one orientation (*e.g.*, all left-facing side views of horses). We use an unsupervised clustering procedure to create an approximate partition of each of the M component subsets, $\mathcal{P}_m$, into two orientation clusters: $\mathcal{P}_m = \mathcal{P}_m^{\mathrm{L}} \cup \mathcal{P}_m^{\mathrm{R}}$.

First, we crop the image region under each bounding box in $\mathcal{P}_m$ and resize it to a fixed width and height (estimated just as before). Each cropped and resized region, along with its vertically flipped counterpart, is mapped to a HOG feature vector. We apply a clustering algorithm to these feature vectors.

The clustering algorithm is a variant of online k-means with the following constraint: no example and its flipped counterpart may be placed into the same cluster. The algorithm begins by selecting at random an example and its flipped counterpart. These feature vectors seed the two clusters. In each iteration, a new example is drawn from the pool of remaining examples. The chosen example is assigned to the cluster with minimum Euclidean distance between the cluster center and the example. The example's flipped counterpart is assigned

3. These mirrored schemas are learned in an unsupervised fashion, so we cannot tell which represents left or which represents right.

to the other cluster.

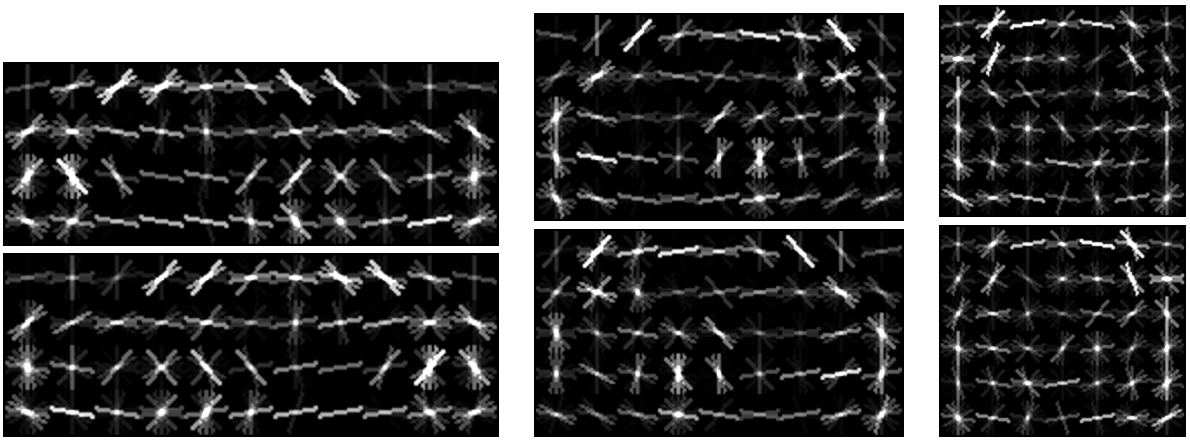
After all examples have been assigned, we use a local search method to improve the clustering. We repeatedly pick an example and its counterpart, and check if swapping their cluster labels reduces the total sum of squared distances (SSD) from examples to their assigned cluster's center. The move is accepted if the SSD decreases and otherwise it is rejected.

The clustering process is repeated several times using different initial seed examples in order to avoid bad local minima. The clustering with the best SSD objective function value, over all random restarts, is selected. Empirically we have found that this process is very effective at grouping most object classes into two clusters corresponding to left and right-facing examples. See Figure 1.4 for some examples. Naturally there are some exceptions (*e.g.*, bottle) where the left-right distinction is not applicable.

After orientation clustering, a root filter for each component is trained by SVM using examples from only one of the orientation clusters. The root filters are then retrained with LSVM or WL-SSVM, on examples from both orientation clusters, but now treating the position, scale, and orientation of each example as latent information. Figure 4.4 the root filters of a three component car model after initializing with latent orientation clustering. These compare directly with Figure 4.2.

4.4 Avoiding learning instabilities

Training star models, mixtures of star models, and richer grammar models can be challenging in practice. Many of the challenges arise from the nonconvexity of the LSVM and WL-SSVM objective functions. The coordinate descent and CCCP algorithms that we use for optimization (Chapter 3) can be thought of as iterative procedures that alternate between two steps: 1) perform inference with the current model to select a fixed "belief" feature vector for each training example, and then 2) optimize a convex function that treats the selected beliefs as though they are the ground truth.



(a) Car component 1 (b) Car component 2 (c) Car component 3

Figure 4.4: The six root filters of a car mixture model after initialization with latent orientation clustering.

If enough of the beliefs selected in step one represent the target concept poorly (*e.g.*, a part that should represent the wheel of a car is placed in random, non-wheel locations), then step 2 will learn a model that tends towards poor inferences the next time the algorithm returns to step one. It is easy to see how this type of procedure can go wrong and lead to a poor local optimum.

Based on our experience with building mixture models, and the richer grammar models described in Chapter 5, we have developed a few strategies that lead to more consistently well-behaved training.

Wait for data mining to converge. Waiting for data mining to truly convergence may take a long time. In [34], to speed up training a small, fixed number (2) of data mining iterations were run. This strategy typically works for star models, but tends to fail for mixture models. With mixture models, or any grammar containing variable substructure, a "seesaw" effect occurs during data mining. In the first data-mining iteration, one model component may generate nearly all of the hard negatives, while the others are starved. This results from taking a max over scores during inference. Then, in the next iteration, another component will take all of the hard negatives. This process oscillates until all model

components are producing a similar number of hard negatives.

If data mining is terminated before the oscillation settles down, then it is possible for one or more of the mixture components to have very poor models (because they have not seen many negative examples). These components will, then, likely score too low to produce many beliefs in the next CCCP (or coordinate descent) iteration. The resulting mixture model may have many degenerate components. Worse, these degenerate components will tend to have very small norms, as a result of ℓ_2 regularization, and biases close to -1. Meanwhile, a good component may produce useful true positive detections below the -1 score threshold. However, just below -1, the degenerate components will produce a large number of false positives (their low norms ensure a small dynamic range around -1). This flood drives precision to zero just below threshold -1.

In practice, we've found that terminating data mining when the relative change in the objective function is less than 5% eliminates this source of instability. We've also found that a small subset (*e.g.*, 200) of background images can be used in all but the last CCCP (or coordinate descent) iterations. This makes data mining fairly fast. In the final CCCP iteration, we expose data mining to all background images (typically ≥ 2000).

Constrain parameters when possible. Even with sufficient data-mining iterations, training simple star models my fail when unconstrained quadratic deformation models are used together with unbounded distance transforms, as was done in [34]. Consider what a deformation model with negative quadratic coefficients means: a part's score will increase (*i.e.*, improve) as the part moves further away from its "ideal" location. This behavior violates our intuitive notion of what a part should be — a local structure of an object category — since such a "part" will try to match non-local image data that is as far away as possible from the object.

To avoid this form of instability, we introduce positivity constraints on the quadratic coefficient parameters. Using bounded distance transforms also helps avoid this problem,

however in practice including the positivity constraints leads to faster data-mining convergence.

Guide belief inference with L_{output}. Unlike latent SVM, the weak-label structural SVM infers beliefs in a loss adjusted fashion by weighing the score of an output against how well the output matches the provided training label (via the loss function $L_{\text{output}}(y, s)$). When the WL-SSVM objective is viewed abstractly, without regard to the specific optimization algorithm that will used, it may make sense to choose L_{output} to be the same as the task loss, or simply to zero because that choice leads to a provably consistent ramp loss objective [52]. However, in the case of a specific optimization algorithm, such as CCCP, the loss L_{output} can be viewed as a general mechanism for imposing constraints on which beliefs may be selected.

Taking this view, L_{output} is a mechanism for taming the nonconvexity of the objective function and guiding a particular optimization algorithm away from bad local optima. In practice, for a foreground example, we choose $L_{\text{output}}(y, s)$ so that it requires box(s) have at least 70% overlap with the bounding box label y. Empirically this leads to better performing detectors than ones trained with a more relaxed form of this loss.

4.5 Max component regularization

We have experimented with replacing the usual ℓ_2 regularizer used in the LSVM and WL-SSVM objectives with a regularizer that only penalizes the mixture model component with the largest norm. Using the decomposition of the model parameters into per-component parameters (Eq. 4.18), we write the "max component" regularizer as

$$\max_{m \in [1,M]} \frac{1}{2} \|\mathbf{w}_m\|^2. \tag{4.21}$$

Max component regularization leads to all components having the same norm, unless reducing the norm of one component also leads to a lower per-example surrogate training

loss. In our setting the training examples are not separable, and the surrogate loss can always be reduced by scaling the weight vector. Empirically we observe that max component regularization yields components with almost exactly equal norms.

The max component regularizer is convex, but it is not differentiable. For optimization with SGD, we use the subgradient $\mathbf{w}_{m^*}$, where $m^* = \text{argmax}_{m \in [1,M]} \|\mathbf{w}_m\|$. We have found that using this subgradient with LBFGS can lead to poor results. Instead, when using LBFGS, we approximate max with a log-sum-exp softmax,

$$\frac{1}{\gamma} \log \sum_{m=1}^{M} \exp\left(\frac{\gamma}{2}\|\mathbf{w}_m\|^2\right), \tag{4.22}$$

where the parameter that controls the approximation quality is set at $\gamma = 1000$.

In Chapter 6 we show that max component regularization leads to a small, but consistent improvement in mean average precision. We hypothesize that the improvement is due to better score calibration between mixture components. When ℓ_2 regularization is used, the per-component biases are the only mechanism for calibrating the scores generated by different components. Bias calibration can be insufficient because the model components may have very different dynamic ranges. In order to get good average precision performance, scores need to be calibrated over the full range of recall, not just at a single point. Otherwise, one component might produce a large number of false positives at a score where the other components are still detecting true positives at high precision. The flood of false positives will cause the precision of the mixture model to be low. Encouraging components to have similar norms equalizes the dynamic ranges of all components.

4.6 Related work

There is a significant body of work on deformable models of various types for object detection, including several kinds of deformable template models (*e.g.* [41, 76, 15, 14]), and a variety of part-based models (*e.g.* [3, 12, 74, 35, 37, 30, 16, 47]).

In the constellation models from [74, 35] parts are constrained to be in a sparse set of locations determined by an interest point operator, and their geometric arrangement is captured by a Gaussian distribution. In contrast, pictorial structure models [37, 30] define a matching problem where parts have an individual match cost in a dense set of locations, and their geometric arrangement is captured by a set of "springs" connecting pairs of parts. The patchwork of parts model from [3] is similar, but it explicitly considers how the appearance model of overlapping parts interact.

Our models are largely based on the pictorial structures framework from [37, 30]. We use a dense set of possible positions and scales in an image, and define a score for placing a filter at each of these locations. The geometric configuration of the filters is captured by a set of deformation costs ("springs") connecting each part filter to the root filter, leading to a star-structured pictorial structure model. Note that we do not model interactions between overlapping parts. While we might benefit from modeling such interactions, this does not appear to be a problem when using models trained with a discriminative procedure, and it significantly simplifies the problem of matching a model to an image.

The introduction of new local and semi-local features has played an important role in advancing the performance of object recognition methods. These features are typically invariant to illumination changes and small deformations. Many recent approaches use wavelet-like features [54, 71] or locally-normalized histograms of gradients [51, 18]. Other methods, such as [5], learn dictionaries of local structures from training images. In our work, we use histogram of oriented gradients (HOG) features from [18] as a starting point, and introduce a variation that reduces the feature size with no loss in performance. As in [44] we used principal component analysis (PCA) to discover low dimensional features, but we note that the eigenvectors we obtain have a clear structure that leads to a new set of "analytic" features. This removes the need to perform a costly projection step when computing dense feature maps.

Significant variations in shape and appearance, such as caused by extreme viewpoint

changes, are not well captured by a 2D deformable model. Aspect graphs [57] are a classical formalism for capturing significant changes that are due to viewpoint variation. Mixture models provide a simpler alternative approach. For example, it is common to use multiple templates to encode frontal and side views of faces and cars [62]. Mixture models have been used to capture other aspects of appearance variation as well, such as when there are multiple natural subclasses in an object category [5].

Matching a deformable model to an image is a difficult optimization problem. Local search methods require initialization near the correct solution [14, 76, 3]. To guarantee a globally optimal match, more aggressive search is needed. One popular approach for part-based models is to restrict part locations to a small set of possible locations returned by an interest point detector [2, 74, 35]. Tree (and star) structured pictorial structure models [30, 36, 16] allow for the use of dynamic programming and generalized distance transforms to efficiently search over all possible object configurations in an image, without restricting the possible locations for each part. We use these techniques for matching our models to images.

Part-based deformable models are parameterized by the appearance of each part and a geometric model capturing spatial relationships among parts. For generative models one can learn model parameters using maximum likelihood estimation. In a fully-supervised setting training images are labeled with part locations and models can often be learned using simple methods [30, 16]. In a weakly-supervised setting training images may not specify locations of parts. In this case one can simultaneously estimate part locations and learn model parameters with EM [3, 74, 35].

Our mixture models are closely related to those of Schneiderman and Kanade [62] and Bernstein and Amit [5]. In [62], mixture models are used to represent different views of faces and cars that result from out-of-plane rotation. Faces are modeled as a mixture of a frontal and a right-facing face detector. Cars are represented as a mixture of eight detectors, each specialized for a different range of right-facing, out-of-plane rotation. Left-facing instances

are detected by running the detector on a right-left flipped copy of the test image. Modeling aspects with mixture models is a simple, though coarse, alternative to classical approaches based on aspect graphs [57].

In contrast to [62], we learn mixtures and orientation without pose supervision. Instead, we use only bounding box annotations and local image features. The way we infer latent orientation at test time also differs from [62]. To evaluate the two orientation choices for each component, we can use the component model on the original input and a flipped copy of the input, or we can simply use an analytically flipped copy of each model component on the original input. Models are typically represented by relatively small vectors compared to feature pyramids, and so flipping each model component is more efficient.

In [5], Bernstein and Amit use expectation maximization (EM) to learn mixture models for MNIST digits, a synthetic dataset composed of deformed LaTeX symbols, and side views of cars. They apply EM to a feature space representation of the training images that is based on a sparse encoding of local edge feature patches. As in our work, they learn mixture models without component label supervision. The specific learning approaches, generative EM versus discriminative LSVM are quite different, however.

Our experience suggests that learning mixture models with LSVM tends to be less stable (*e.g.*, components collapse more easily) and requires more careful initialization than is necessary for EM, which usually works well with random initialization. Despite these difficulties, we use a discriminative training framework for a variety of reasons. From a computational perspective, it is difficult to match generative part-based models to images without over counting evidence when parts overlap. One approach, based on local search, is presented in [3]. From a modeling perspective, it is often easier to specify and implement new ideas in discriminative models. For example, all of our models capture image information at multiple resolutions and are learned from a set of examples with diverse (and latent) scales. In a generative setting, it is not clear how to express multiresolution models, or how to run EM with latent scale, in ways that do not run afoul of correct probabilistic modeling

rules, or make very crude independence assumptions. Finally, from a parameter learning perspective, discriminative training can be seen as a more direct way to learn the parameters of a family of decision functions, while making fewer assumptions than would be made in a generative model that corresponds to the same decision family. This flexibility appears to avoid over-counting issues that can arise when inappropriate independence assumptions are made. Additionally, the more "task oriented" objectives used in discriminative training tend to result in better performance in practice.

CHAPTER 5

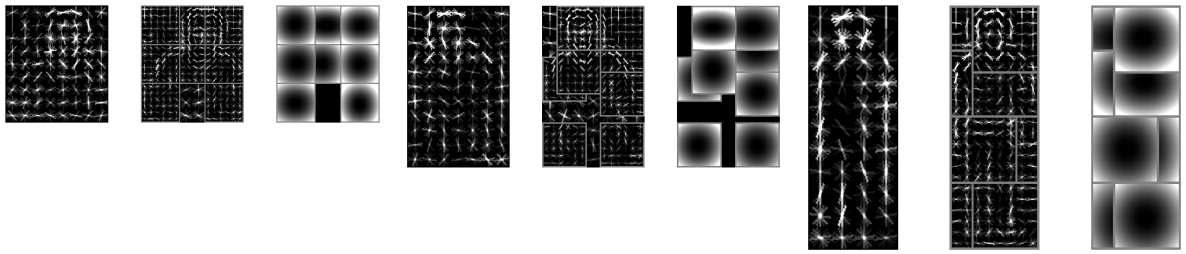
TOWARDS RICHER GRAMMAR MODELS

The idea that images can be hierarchically parsed into objects and their parts has a long history in computer vision, see for example [53]. Image parsing has also been of considerable recent interest [33, 43, 79, 80, 82]. However, it has been difficult to demonstrate that sophisticated grammar models lead to performance advantages on challenging metrics such as the PASCAL object detection benchmark [26]. In this chapter we achieve new levels of performance for person detection using a grammar model that is richer than previous models used in high-performance systems.

Our models are based on the grammar formalism from [33] that we described in Section 2.1. A grammar model represents objects in terms of other objects through compositional rules. Deformation rules allow for parts of an object to move relative to each other, leading to hierarchical deformable part models. Structural variability provides choice between multiple part subtypes — effectively creating mixture models throughout the compositional hierarchy — and also enables optional parts. In this formalism parts may be reused both within an object category and across object categories.

Our baseline and departure point is the UoC-TTI object detector [32, 31]. This system represents a class of objects with three different pictorial structure models. Although these models are learned automatically, making semantic interpretation unclear, it seems that the three components for the person class differ in how much of the person is taken to be visible — just the head and shoulders, the head and shoulders together with the upper body, or a whole standing person (Figure 5.1). Each of the three components has independently trained parts. In particular each component has a head part trained independently from the head part of the other components.

Here we construct a single grammar model that allows more flexibility in describing the amount of the person that is visible. The grammar model avoids dividing the training data between different components and thus uses the training data more efficiently. The parts in



(a) Component 1 (b) Component 2 (c) Component 3

Figure 5.1: The three component mixture model learned for the PASCAL 2007 person category.

the model, such as the head part, are shared across different interpretations of the degree of visibility of the object. The grammar model also includes subtype choice at the part level to accommodate greater appearance variability across object instances. We use parts with subparts to benefit from high-resolution image data, while also allowing for deformations. Unlike previous approaches, we explicitly model the source of occlusion for partially visible objects.

Our approach differs from that of Jin and Geman [43] in that theirs focuses on whole scene interpretation with fully probabilistic models, while we focus on discriminatively trained models of individual objects. We also make Markovian restrictions not made in [43]. Our work is more similar to that of Zhu *et al.* [79] who impose similar Markovian restrictions. However, our training method, image features, and grammar design are substantially different.

The model presented here is designed to accurately capture the visible portion of a person. There has been recent related work on occlusion modeling in pedestrian and person images [22, 72]. In [22], Enzweiler *et al.* assume access to depth and motion information in order to estimate occlusion boundaries. In [72], Wang *et al.* rely on the observation that the scores of individual filter cells (using the Dalal and Triggs detector [18]) can reliably predict occlusion in the INRIA pedestrian data. This does not hold for the harder PASCAL person data.

Pushing towards richer models introduce new learning challenges. By allowing for more fine-grained occlusion reasoning, we also introduce the ability for the model to make new

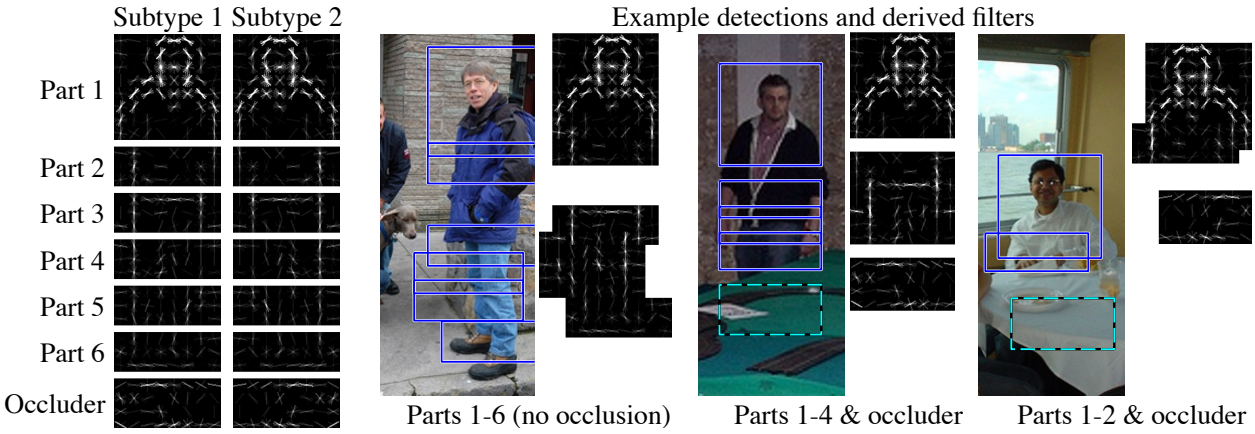


Figure 5.2: Shallow grammar model. This figure illustrates a shallow version of our grammar model (Section 5.1). This model has six person parts and an occlusion model ("occluder"), each of which comes in one of two subtypes. A detection places one subtype of each visible part at a location and scale in the image. If the derivation does not place all parts it must place the occluder. Parts are allowed to move relative to each other, but are constrained by deformation penalties.

kinds of detection mistakes. For example, when a full person is visible, a derivation that only places the head might score higher than a derivation that places the full person. Intuitively, we would like to use a training procedure that calibrates the scores across different derivations so that the correct model output scores highest. The latent SVM framework cannot encode this type of constraint. We show that training with weak-label structural SVM (Section 3.1), which allows for soft margin requirements between different outputs on training examples, improves detection results over training with latent SVM.

5.1 A grammar model for detecting people

Each component in the person model learned by the `voc-release4` system [31] is tuned to detect people under a prototypical visibility pattern. Based on this observation we designed, by hand, the structure of a grammar that models visibility by using structural variability and optional parts. For clarity, we begin by describing a shallow model (Figure 5.2) that places all filters at the same resolution in the feature pyramid. After explaining this model, we describe a deeper model that includes deformable subparts at higher resolutions.

Fine-grained occlusion. Our grammar model has a start symbol Q that can be expanded using one of six possible structural schemas. These choices model different degrees of visibility ranging from heavy occlusion (only the head and shoulders are visible) to no occlusion at all.

Beyond modeling fine-grained occlusion patterns when compared to the mixture models from [22] and [31], our grammar model is also richer in a number of ways. In Section 5.5 we show that each of the following modeling choices improves detection performance.

Occlusion model. If a person is occluded, then there must be some cause of the occlusion — either the edge of the image or an occluding object, such as a desk or dinner table. We use a nontrivial model to capture the appearance of the *stuff* that occludes people.

Part subtypes. The mixture model from [31] has two subtypes for each mixture component. The subtypes are forced to be mirror images of each other and correspond roughly to left-facing versus right-facing people. Our grammar model has two subtypes for each part, which are also forced to be mirror images of each other. But in the case of our grammar model, the decision of which part subtype to instantiate at detection time is independent for each part.

The shallow person grammar model is defined by the following grammar. The indices p (for part), t (for subtype), and k have the following ranges: $p \in \{1, \ldots, 6\}$, $t \in \{\mathrm{L}, \mathrm{R}\}$ and $k \in \{1, \ldots, 5\}$.

$$
\begin{aligned}
Q(\omega) &\xrightarrow{\beta_k} \{\ Y_1(\omega \oplus \delta_1), \ldots, Y_k(\omega \oplus \delta_k), O(\omega \oplus \delta_{k+1})\ \} \\
Q(\omega) &\xrightarrow{\beta_6} \{\ Y_1(\omega \oplus \delta_1), \ldots, Y_6(\omega \oplus \delta_6)\ \}
\end{aligned}
$$

$$
\begin{aligned}
Y_p(\omega) &\xrightarrow{0} \{\ Y_{p,t}(\omega)\ \} \qquad Y_{p,t}(\omega) \xrightarrow{-\mathbf{d}_{p,t} \cdot \boldsymbol{\phi}_\Delta(\delta)} \{\ A_{p,t}(\omega \oplus \delta)\ \} \\
O(\omega) &\xrightarrow{0} \{\ O_t(\omega)\ \} \qquad O_t(\omega) \xrightarrow{-\mathbf{d}_t \cdot \boldsymbol{\phi}_\Delta(\delta)} \{\ A_t(\omega \oplus \delta)\ \}
\end{aligned}
$$

The grammar has a start symbol Q with six alternate choices that derive people under varying degrees of visibility (occlusion). Each part has a corresponding nonterminal Y_p that is placed at some ideal position relative to Q. Derivations with occlusion include the occlusion symbol O. A derivation selects a subtype and displacement for each visible part. The parameters of the grammar (production biases, deformation parameters, and filters) are learned with the discriminative procedure described in Section 5.3. Figure 5.2 illustrates the filters in the resulting model and some example detections.

Deeper model. We extend the shallow model by adding deformable subparts at two scales: (1) the same as, and (2) twice the resolution of the start symbol Q. When detecting large objects, high-resolution subparts capture fine image details. However, when detecting small objects, high-resolution subparts cannot be used because they "fall off the bottom" of the feature pyramid. The model uses derivations with low-resolution subparts when detecting small objects.

We begin by *replacing* the productions from $Y_{p,t}$ in the grammar above, and then adding new productions. Recall that p indexes the top-level parts and t indexes subtypes. In the following schemas, the indices r (for resolution) and u (for subpart) have the ranges: $r \in \{\mathrm{H}, \mathrm{L}\}, u \in \{1, \ldots, N_p\}$, where N_p is the number of subparts in a top-level part Y_p.

$$
\begin{array}{rcl}
Y_{p,t}(\omega) & \xrightarrow{-\mathbf{d}_{p,t} \cdot \boldsymbol{\phi}_\Delta(\delta)} & \{\ Z_{p,t}(\omega \oplus \delta)\ \} \\
Z_{p,t}(\omega) & \xrightarrow{0} & \{A_{p,t}(\omega), W_{p,t,r,1}(\omega \oplus \delta_{p,t,r,1}), \ldots, W_{p,t,r,N_p}(\omega \oplus \delta_{p,t,r,N_p})\} \\
W_{p,t,r,u}(\omega) & \xrightarrow{-\mathbf{d}_{p,t,r,u} \cdot \boldsymbol{\phi}_\Delta(\delta)} & \{A_{p,t,r,u}(\omega \oplus \delta)\}
\end{array}
$$

We note that as in [81] our model has hierarchical deformations. The part terminal $A_{p,t}$ can move relative to Q and the subpart terminal $A_{p,t,r,u}$ can move relative to $A_{p,t}$.

The displacements $\delta_{p,t,\mathrm{H},u}$ place the symbols $W_{p,t,\mathrm{H},u}$ one octave below $Z_{p,t}$ in the feature pyramid. The displacements $\delta_{p,t,\mathrm{L},u}$ place the symbols $W_{p,t,\mathrm{L},u}$ at the same scale as $Z_{p,t}$. We add subparts to the first two top-level parts ($p = 1$ and 2), with the number of subparts set

to $N_1 = 3$ and $N_2 = 2$. We find that adding additional subparts does not improve detection performance.

5.2 Inference and test time detection

Inference involves finding high scoring derivations. At test time, because images may contain multiple instances of an object class, we compute the maximum scoring derivation rooted at $Q(\omega)$, for each $\omega \in \Omega$. This can be done efficiently using a standard dynamic programming algorithm (see Section 2.1.2 or [33]).

We retain all derivations that score above a threshold, which we set low enough to ensure high recall. We use box(T) to denote a detection window associated with a derivation T. Given a set of candidate detections, we apply nonmaximal suppression to produce a final set of detections.

We define box(T) by assigning a detection window size, in feature coordinates, to each structural schema that can be applied to Q. This leads to detections with one of six possible aspect ratios, depending on which production was used in the first step of the derivation. The absolute location and size of a detection depends on the placement of Q. For the first five production schemas, the ideal location of the occlusion part, O, is *outside* of box(T).

5.3 Training grammar models

Now we consider learning the parameters of an object detection grammar using the training data in the PASCAL VOC datasets with the WL-SSVM framework. For two rectangles a and b let $\text{overlap}(a, b) = \text{area}(a \cap b)/\text{area}(a \cup b)$. We will use this measure of overlap in our loss functions.

The training data specifies a bounding box for each instance of an object in a set of training images. We construct a set of weakly-labeled examples $\{(x_1, y_1), \ldots, (x_n, y_n)\}$ as follows. For each training image I, and for each bounding box B in I, we define a *foreground*

example (x, y), where $y = B$, x specifies the image I, and the set of valid predictions $\mathcal{S}(x)$ includes:

1. Derivations T with $\text{overlap}(\text{box}(T), B) \geq 0.1$ *and* $\text{overlap}(\text{box}(T), B') < 0.5$ for all B' in I such that $B' \neq B$.
2. The background output $\perp$.

The overlap requirements in (1) ensure that we consider only predictions that are relevant for a particular object instance, while avoiding interactions with other objects in the image. In (2) we augment our model's output space (the set of all derivation trees) to also include a special "background" output $\perp$. With the loss defined below, this output encourages at least one non-background output to score high for each foreground example. We define $\boldsymbol{\psi}(x, \perp)$ to be the zero vector, as was done in [7]. Thus the score of a background hypothesis is zero independent of the model parameters $\mathbf{w}$.

We also define a very large set of *background* examples. For simplicity, we use images that do not contain any bounding boxes. For each background image I, we define a different example (x, y) for each position and scale ω within I. In this case $y = \perp$, x specifies the image I, and $\mathcal{S}(x)$ includes derivations T rooted at $Q(\omega)$ and the background output $\perp$. The set of background examples is very large because the number of positions and scales within each image is typically around 250,000.

5.3.1 Loss functions

Recall from Chapter 3 that the per-example surrogate training loss used by WL-SSVM is

$$L_{\text{surr}}(\mathbf{w}, x, y) = \underbrace{\max_{s \in \mathcal{S}(x)} \left[\mathbf{w} \cdot \boldsymbol{\psi}(x, s) + L_{\text{margin}}(y, s)\right]}_{(5.1a)} - \underbrace{\max_{s \in \mathcal{S}(x)} \left[\mathbf{w} \cdot \boldsymbol{\psi}(x, s) - L_{\text{output}}(y, s)\right]}_{(5.1b)}. \quad (5.1)$$

To train our model using WL-SSVM we must specify L_{margin} and L_{output}.

The PASCAL benchmark requires a correct detection to have at least 50% overlap with a ground-truth bounding box. We use this rule to define our loss functions. First, define $L_{l,\tau}(y, s)$ as follows

$$L_{l,\tau}(y,s) = \begin{cases} l & \text{if } y = \perp \text{ and } s \neq \perp \\ 0 & \text{if } y = \perp \text{ and } s = \perp \\ l & \text{if } y \neq \perp \text{ and } \text{overlap}(y,s) < \tau \\ 0 & \text{if } y \neq \perp \text{ and } \text{overlap}(y,s) \geq \tau. \end{cases} \tag{5.2}$$

Following the PASCAL VOC protocol we use $L_{\text{margin}} = L_{1,0.5}$. For a foreground example this pushes down the score of detections that don't overlap with the bounding box label by at least 50%.

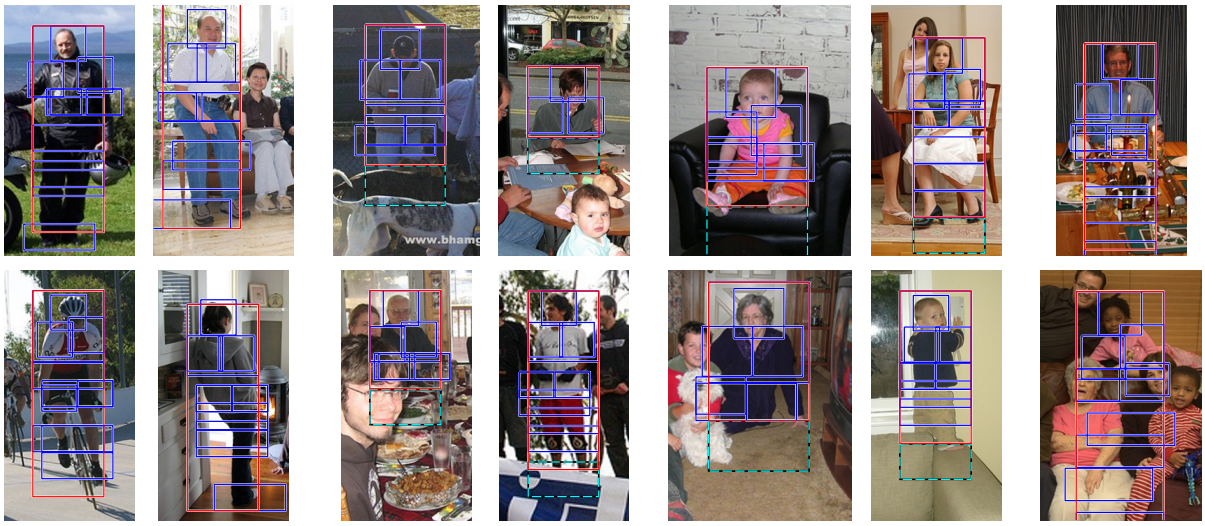
Instead of using $L_{\text{output}} = L_{\text{margin}}$, we let $L_{\text{output}} = L_{\infty,0.7}$. For a foreground example this ensures that the maximizer of (5.1b) is a detection with high overlap with the bounding box label. For a background example, the maximizer of (5.1b) is always $\perp$. This choice allows us to use the "lazy background caching" strategy described in Section 3.1.3. While our choice of L_{output} does not produce a convex objective, it does tightly limit the range of outputs, making our optimization less prone to reaching bad local optima.

5.3.2 Optimization

We optimize the resulting weak-label structural SVM using the concave-convex procedure with data mining given in Section 3.1.2. For our experiments we set M, the number of false positive outputs used for data mining the foreground examples, to 1.

5.4 Initialization

Using CCCP requires an initial model or heuristic for selecting the initial outputs (beliefs) $s_i(\mathbf{w}_0)$. Inspired by the methods in [32, 31], we train a single filter for fully visible people using a standard binary SVM. To define the SVM's training data, we select vertically elon-



(a) Full visibility (b) Occlusion boundaries (c) Early termination (d) Mistakes

Figure 5.3: Example detections. Parts are blue. The occlusion part, if used, is dashed cyan. (a) Detections of fully visible people. (b) Examples where the occlusion part detects an occlusion boundary. (c) Detections where there is no occlusion, but a partial person is appropriate. (d) Mistakes, where the model did not detect occlusion properly.

gated examples. We apply the orientation clustering method from Section 4.3.1 to further divide these examples into two sets that approximately correspond to left-facing versus right-facing orientations. Examples from one of these two sets are then anisotropically rescaled so their HOG feature maps match the dimensions of the filter. These form the positive examples. For negative examples, random patches are extracted from background images. After training the initial filter, we slice it into subfilters (one 8×8 and five 3×8) that form the building blocks of the grammar model. We mirror these six filters to get subtypes, and then add subparts using the energy covering heuristic in [32, 31].

5.5 Experimental results

We evaluated the performance of our person grammar and training framework on the PASCAL VOC 2007 and 2010 datasets [23, 26]. We used the standard PASCAL VOC `comp3` test protocol, which measures detection performance by average precision (AP) over different recall levels. Figure 5.3 shows some qualitative results, including failure cases.

	Grammar	+bbox	+context	UoC-TTI [26]	+bbox	+context	Poselets [26]
AP	47.5	47.6	**49.5**	44.4	45.2	47.5	48.5

Table 5.1: PASCAL 2010 results. UoC-TTI and our method compete in `comp3`. Poselets competes `comp4` due to its use of detailed pose and visibility annotations and non-PASCAL images.

	Grammar LSVM	Grammar WL-SSVM	Mixture LSVM	Mixture WL-SSVM
AP	45.3	**46.7**	42.6	43.2

Table 5.2: Training objective and model structure evaluation on PASCAL 2007.

PASCAL VOC 2010. Our results on the 2010 dataset are in Table 5.1. We present our results in the context of two strong baselines. The first, UoC-TTI, won the person category in the `comp3` track of the 2010 competition [26]. This method extends [31] by adding an extra octave to the HOG feature pyramid, which allows the detector to find smaller objects. We also computed the AP score of the UoC-TTI basic person detector, as well as the scores after applying the bounding box prediction and context rescoring methods described in [32]. Comparing raw detector outputs, prior to post-processing, our grammar model significantly outperforms the mixture model: 47.5 vs. 44.4.

We also applied these two post-processing steps to the grammar model, and found that unlike with the mixture model, the grammar model does not benefit from bounding box prediction. This is likely because our fine-grained occlusion model reduces the number of near misses that are fixed by bounding box prediction. To test context rescoring, we used the UoC-TTI detection data for the other 19 object classes. Context rescoring boosts our final score to 49.5.

The second baseline is the poselets system described in [11]. Their system requires detailed pose and visibility annotations, in contrast to our grammar model which was trained only with bounding box labels. Prior to context rescoring, our model scores one point lower than the poselets model, and after rescoring it scores one point higher.

Structure and training. We evaluated several aspects of our model structure and training objective on the PASCAL VOC 2007 dataset. In Table 5.2 we compare the weak-label structural SVM framework developed in this dissertation with the binary latent SVM framework used in [32]. For our grammar model, WL-SSVM improves performance by 1.4 AP points over binary LSVM training. WL-SSVM also improves the mixture model results, but by a smaller 0.6 points. In all of our experiments we set the regularization constant to $C = 0.006$ and use an extra feature pyramid octave. Chapter 6 has additional experiments that compare LSVM *vs.* WL-SSVM on the PASCAL VOC 2011 dataset.

To investigate model structure, we evaluated the effect of part subtypes and occlusion modeling. Removing subtypes reduces the score from 46.7 to 45.5. Removing the occlusion part also decreases the score from 46.7 to 45.5. The shallow model (no subparts) achieves a score of 40.6.

5.6 Discussion

This chapter establishes grammar-based methods as a high-performance approach to object detection by demonstrating their effectiveness on the challenging task of detecting people in the PASCAL VOC datasets. To do this, we carefully designed a flexible grammar model that can detect people under a wide range of partial occlusion, pose, and appearance variability. Automatically learning the structure of grammar models remains a significant challenge for future work. We hope that our empirical success will provide motivation for pursing this goal, and that the structure of our handcrafted grammar will yield insights into the properties that an automatically learned grammar might require. We also develop a structured training framework, weak-label structural SVM, that naturally handles learning a model with strong outputs, such as derivation trees, from data with weak labels, such as bounding boxes. Our training objective is nonconvex and we use a strong loss function to avoid bad local optima. We plan to explore making this loss softer, in an effort to make learning more robust to outliers.

CHAPTER 6

EXPERIMENTAL EVALUATION

The goal of this chapter is twofold. First, we demonstrate the substantial improvements in object detection performance contributed by this work, compared to the original discriminatively trained part-based model from [34]. Second, we look at detector performance over twenty splits of the PASCAL VOC 2011 `train+val` dataset and offer some recommendations about how to judge if experimental results are yielding significant changes in performance. We begin by briefly reviewing the main contributions to evaluate.

1. [EHOG] **Enhanced HOG features** (Section 2.3.1) — a lower dimensional reformulation of classical HOG features [18] that uses both contrast sensitive and contrast insensitive gradient orientations.

2. [TRUNC] **Image boundary truncation feature** (Section 2.3.2) — an image feature allowing filter cells placed outside of the image to have nonzero scores, leading to better score calibration for partially truncated objects.

3. [SM-OBJ] **Small object feature** (Section 2.3.3) — a combination of adding an extra octave to the HOG feature pyramid, computed using 2×2 pixel HOG cells, and scale-dependent production biases that help models learn how to calibrate the scores of small detections with detections at larger scales.

4. [BBOX] **Bounding box prediction** (Section A.1) — a regression method that predicts a new detection window from an existing detection and its associated filter placements.

5. [CNTXT] **Context rescoring** (Section A.2) — a method that re-ranks detections in a single image based on the scores of other object category detectors in the same image.

6. [OPTIM] **Optimization improvements** (Section 4.4) — we always data mine until (near) convergence; constrain quadratic deformation parameter; and optimize the *true* LSVM or WL-SSVM objective functions, instead a heuristic implemented on top of classical SVM (*cf.* [34]).

7. [MIX] **Mixture models without latent orientation** (Section 4.2). — mixtures of star-structured deformable part models.

8. [ORIENT] **Mixture models with latent orientation** (Section 4.3) — mixture models with latent left-*versus*-right orientation.

9. [MAX-REG] **Max component regularization** (Section 4.5) — an alternative form of regularization for LSVM and WL-SSVM that is applicable to mixture models.

10. [WL-SSVM] **Weak-label structural SVM** (Section 3.1) — A generalization of structural SVM, latent structural SVM, and latent SVM that supports training models from weakly-labeled data using structural loss functions.

6.1 Evaluation overview

Unless otherwise noted, all experiments were conducted with the following setup.

- The `voc-release4.01` version of our object detection system, which we have made publicly available, was used.
- Models were trained with LSVM (as opposed to WL-SSVM).
- Mixture models were built with a fixed number (three) of components.
- We use the PASCAL VOC detection evaluation protocol [28] and train only the provided training images and annotations (*i.e.*, our methods "compete" in the `comp3` track of the challenge).

6.1.1 Computing average precision

To compute an average precision score for any of the PASCAL VOC challenges, the precision-recall (PR) curve is first replaced by its tightest monotonically decreasing upper bound. This transformation removes the characteristic "sawtooth" shape found is many PR curves.

After computing this upper bound (which we will refer to as simply the "PR curve" from here on), the exact method for computing average precision changed in the 2010 challenge. Prior to 2010, average precision was computed by sampling precision values from the PR curve at 11 points ($\{i/10 \mid i \in [0, 10]\}$), and averaging those values. Such coarse sampling can lead to very large changes in AP (*e.g.* ± 10 AP points, *absolute*) when performance is low. Starting in 2010, this coarse sampling method was replaced by the exact area under the (upper-bounded) PR curve. Computing the exact area under the curve leads to less noisy, more meaningful AP scores.

When comparing results on the 2007 dataset with existing methods, we will typically use the (bad) 11-point AP calculation, because existing methods will have already published their results using that AP formula. Otherwise, we will use the better, area-under-the-curve formula for computing AP. To avoid confusion, we will be explicit about the formula used by referring to the 11-point method as 11PT-AP and the area under the curve method as AUC-AP. We caution that when comparing scores in the 0-15% AP range, under the 11PT-AP calculation, differences — even large ones — should be ignored.

6.2 Model structure

6.2.1 Comparison to LSVM CVPR [34]

Table 6.1 shows average precision and mean average precision (mAP) scores on the PASCAL VOC 2007 dataset using 11PT-AP. The first row (LSVM CVPR) is the baseline star-structured deformable parts model from [34]. The next row of results (star model) uses a

	aero	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	mbike	person	plant	sheep	sofa	train	tv	mAP
LSVM CVPR [34]	18.0	41.1	9.2	9.8	24.9	34.9	39.6	11.0	15.5	16.5	11.0	6.2	30.1	33.7	26.7	14.0	14.1	15.6	20.6	33.6	22.3
	voc-release4.01: EHOG + TRUNC + BBOX + OPTIM + MAX-REG																				
	aero	**bike**	**bird**	**boat**	**bottle**	**bus**	**car**	**cat**	**chair**	**cow**	**table**	**dog**	**horse**	**mbike**	**person**	**plant**	**sheep**	**sofa**	**train**	**tv**	**mAP**
star model	21.5	49.7	9.3	9.3	27.3	39.8	53.3	11.4	13.5	17.8	28.6	3.7	42.1	38.3	33.1	13.4	15.1	20.4	28.7	42.7	26.0
MIX	33.2	58.0	2.3	15.7	26.5	49.3	55.5	13.9	18.1	19.1	24.6	12.6	47.3	42.6	40.5	14.4	17.0	27.4	37.9	39.0	29.7
$\cdots$ + **ORIENT**	28.9	59.5	10.0	15.2	25.5	49.6	57.9	19.3	22.4	25.2	23.3	11.1	56.8	48.7	41.9	12.2	17.8	33.6	45.1	41.6	32.3
$\cdots$ + **CNTXT**	31.2	61.5	11.9	17.4	27.0	49.1	59.6	23.1	23.0	26.3	24.9	12.9	60.1	51.0	43.2	13.4	18.8	36.2	49.1	43.0	34.1

Table 6.1: PASCAL VOC 2007 results with AP scores computed using 11PT-AP. All models were trained on `train+val` and tested on `test`.

	voc-release4.01: EHOG + TRUNC + BBOX + OPTIM + MAX-REG																				
	aero	**bike**	**bird**	**boat**	**bottle**	**bus**	**car**	**cat**	**chair**	**cow**	**table**	**dog**	**horse**	**mbike**	**person**	**plant**	**sheep**	**sofa**	**train**	**tv**	**mAP**
star model	18.9	49.9	0.7	1.0	25.2	38.0	54.1	9.0	9.2	15.0	27.2	2.5	41.3	36.8	31.5	8.3	11.3	16.8	26.6	42.6	23.3
MIX	31.6	58.4	1.0	11.5	24.2	48.6	55.7	7.7	15.1	15.0	21.8	6.0	47.2	41.6	39.2	8.9	13.3	24.2	36.5	37.8	27.3
$\cdots$ + **ORIENT**	26.4	60.5	2.7	11.8	22.9	49.8	58.6	15.7	19.9	21.8	20.4	3.3	57.5	48.4	40.8	7.3	15.2	31.8	44.8	41.4	30.1

Table 6.2: PASCAL VOC 2007 results with AP scores computed using AUC-AP. All models were trained on `train+val` and tested on `test`.

star-structured deformable parts model as well. The improved mAP performance, from 22.3 to 26.0, is the combined result of EHOG, TRUNC, BBOX, OPTIM, and MAX-REG.

Next, we can see the change in performance when going from a single star model to a mixture of three star models, *without latent orientation*. These results are presented in the row labeled MIX. The next two rows build on MIX by adding in latent orientation ($\cdots$+ ORIENT) and then context rescoring ($\cdots$+ CNTXT). Overall, the combination of these improvements boosts mAP by approximately 50% relative to the LSVM CVPR baseline.

6.2.2 PASCAL 2007 and 2011 with AUC-AP

Now we look into changes in model structure using more reliable AUC-AP scores on the PASCAL VOC 2007 and 2011 datasets. Table 6.2 shows AUC-AP scores using the same PR curves as were used in Table 6.1. Unfortunately, PR curves for the LSVM CVPR baseline are not available, and so we cannot recompute scores for that method.

Scores under the AUC-AP formula are very different than the scores under the 11PT-AP for some classes. In particular, *bird* scored $\{9.3,\ 2.3,\ 10.0,\ 11.9\}$ under 11PT-AP *vs.*

{0.7, 1.0, 2.7, 5.6} under AUC-AP. The large difference between the 9.3 and 0.7 score is due to the first detection being a single true positive, which contributes ≈ 9.1 points to the overall score under the 11PT-AP formula. The other most effected categories are: *boat*, *dog*, *pottedplant*, and to a lesser degree, *sheep* and *sofa*.

Most classes benefit from enriching the star model to a mixture of star models. Several categories (*aeroplane*, *bicycle*, *bus*, *train*) have very large — nearly 10 AP point — score boosts. A few classes (*bottle*, *cat*, *diningtable*, *tvmonitor*) lose between 1 and 5 points. For most of the categories that have decreased performance (*bottle*, *diningtable*, *tvmonitor*) building mixture components based on aspect ratio is not very meaningful and primarily results in data-impoverished training. Overall, the change in mAP shows an increase of 4 AP points, from 23.3 to 27.3, as a result of using mixture models.

Next, we look at enriching mixture models by adding latent orientation. Most classes improve in performance by 1-2 points. Latent orientation particularly helps *chair*, *cow*, *horse*, *motorbike*, and *train* where the improvement is as much as 10 points. Overall, adding latent orientation to our mixture models boosts mAP performance by slightly less than 3 points, from 27.3 to 30.1.

Table 6.3 shows the same model structures trained and tested on the `train` and `val` splits of the 2011 datasets. Extending star models to mixtures, and then to mixtures with latent orientation, boosts mAP from 21.3 to 23.9, and then to 26.7. The trend is the same as in the 2007 dataset, though the magnitude of improvement is smaller.

	voc-release4.01: EHOG + TRUNC + BBOX + OPTIM + MAX-REG																				
	aero	**bike**	**bird**	**boat**	**bottle**	**bus**	**car**	**cat**	**chair**	**cow**	**table**	**dog**	**horse**	**mbike**	**person**	**plant**	**sheep**	**sofa**	**train**	**tv**	***mAP***
star model	40.5	44.2	0.5	1.3	23.0	49.3	36.1	13.3	8.5	4.6	12.0	9.3	23.8	32.3	33.4	4.6	17.8	10.8	22.6	38.3	21.3
MIX	39.9	44.0	0.7	6.2	22.0	52.2	36.9	17.9	10.5	9.6	11.4	11.5	30.5	24.7	42.0	8.3	24.8	16.8	32.2	35.3	23.9
··· **+ ORIENT**	44.2	51.1	6.2	6.7	20.5	54.6	39.8	25.3	15.1	16.1	7.9	11.1	36.7	34.3	42.5	3.7	27.4	18.9	38.4	34.3	26.7

Table 6.3: PASCAL VOC 2011 results with AP scores computed using AUC-AP. All models were trained on `train` and tested on `val`.

6.3 Small objects (SM-OBJ)

Table 6.4 reports the effect of adding an extra octave to the feature pyramid and including a feature that allows models to learn a bias adjustment for detections in the new pyramid octave. The categories *bottle*, *car*, and *person* benefit the most from SM-OBJ. The other classes are largely unaffected. Training and testing with the extra octave is time consuming, and so we report pre-existing, though slightly old results, rather than re-running the experiment with more recent code.

	setup: EHOG + TRUNC + OPTIM + MIX + ORIENT																				
	aero	**bike**	**bird**	**boat**	**bottle**	**bus**	**car**	**cat**	**chair**	**cow**	**table**	**dog**	**horse**	**mbike**	**person**	**plant**	**sheep**	**sofa**	**train**	**tv**	**mAP**
	43.3	48.6	4.6	5.3	21.1	53.5	35.8	20.2	13.9	16.7	7.1	10.0	35.5	36.1	40.5	4.9	25.7	14.8	32.8	35.5	25.3
SM-OBJ	44.7	48.5	4.6	5.8	**28.5**	53.9	**39.8**	20.5	11.7	15.7	8.7	8.8	36.0	34.1	**43.8**	5.1	26.5	15.3	33.8	35.1	26.0

Table 6.4: Effect of SM-OBJ on the PASCAL VOC 2011 `train/val` split. Note that BBOX and MAX-REG were not used in this experiment. Scores reported using AUC-AP.

6.4 Variance due to train/test splits

Understanding if a proposed change, such as an alternative grammar structure or a new optimization algorithm, results in improved performance is an important question. The idea of testing a hypothesis for statistical significance is used throughout most scientific fields, though it is notably missing in most recent computer vision work. The computer vision community is largely focused on a few benchmark datasets, such as PASCAL, and the idea of what constitutes a "significant" improvement in performance in something of a shared community belief based on lore and the performance deltas reported in conference papers. For PASCAL, the conventional wisdom[1] is that an improvement of 1 AP point, or more, in any category constitutes a meaningful change in performance.

1. Based on conversations with other PASCAL challenge participants and authors at recent CVPR, ICCV, and ECCV conferences.

				AP scores on splits																		
class	**mean**	**std**	**range**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**
aero	41.1	2.1	8.0	42.2	43.6	38.0	38.6	43.3	39.6	39.3	41.0	41.8	41.8	42.1	40.5	44.7	36.7	40.8	40.0	40.4	42.1	40.7	44.2
bicycle	49.3	1.7	5.6	49.6	49.2	47.5	49.3	49.0	51.2	51.6	47.7	47.2	47.4	50.2	47.1	49.3	46.9	48.5	49.4	52.5	49.3	50.7	52.1
bird	4.8	1.2	4.0	6.3	4.8	5.2	6.9	5.1	5.3	3.2	3.3	6.0	6.0	4.5	4.3	4.4	4.0	4.5	3.3	3.8	3.2	7.2	5.4
boat	6.2	1.0	3.4	6.0	7.6	8.0	6.4	6.7	5.0	6.2	6.6	6.2	6.3	5.5	6.8	5.5	4.6	4.7	5.3	7.4	6.1	7.7	5.7
bottle	18.3	1.8	6.8	20.1	14.6	18.8	21.0	19.0	16.5	19.1	18.5	18.0	19.8	17.9	16.5	19.8	20.9	18.7	17.6	17.9	14.2	19.7	17.4
bus	54.5	2.0	7.5	54.1	55.9	51.8	55.3	58.2	55.6	55.5	55.5	50.8	54.6	52.7	52.7	52.9	57.6	55.3	54.9	56.0	52.0	55.6	52.5
car	38.4	0.9	3.4	38.3	38.4	38.8	38.0	38.0	38.5	36.9	39.0	36.5	37.6	39.4	39.0	37.5	39.9	38.7	37.6	38.5	39.9	39.8	38.5
cat	22.2	2.7	10.0	24.0	19.2	25.5	22.4	21.9	19.5	22.2	21.6	25.2	24.7	21.9	16.5	20.6	19.1	18.5	23.0	22.8	25.0	26.4	24.1
chair	14.3	1.1	4.3	15.0	14.7	13.5	13.3	13.5	14.0	13.3	15.2	14.6	12.9	12.0	15.9	13.7	15.5	15.4	16.3	14.8	13.7	14.1	14.8
cow	14.2	1.7	5.0	16.1	12.0	13.9	15.9	15.6	11.8	16.3	11.6	14.0	15.1	14.2	13.7	11.6	11.4	15.2	14.1	14.6	15.8	14.4	16.4
table	9.7	1.5	6.2	9.0	10.2	10.7	13.6	7.4	8.5	8.5	11.7	10.4	7.8	9.9	9.3	7.6	9.4	9.3	10.5	10.0	11.0	8.6	10.2
dog	10.7	1.1	3.8	10.7	11.9	11.9	10.6	8.7	10.5	11.5	11.4	12.4	10.1	9.7	11.0	11.5	10.6	9.4	8.6	9.9	11.0	9.5	12.1
horse	38.1	1.4	5.5	37.2	38.2	38.8	38.8	36.3	39.3	37.1	36.7	38.4	38.0	39.5	40.4	39.1	35.8	39.3	38.3	34.9	39.2	38.6	37.9
mbike	40.5	3.4	11.1	33.1	37.5	38.2	41.1	43.1	44.2	43.8	42.3	43.4	38.8	43.5	41.1	41.7	40.9	43.9	40.3	42.4	33.5	41.6	35.7
person	40.4	1.0	4.8	42.3	40.9	41.2	38.9	39.2	40.4	40.1	41.3	40.2	40.8	40.1	41.4	40.0	41.1	39.8	37.5	40.1	40.6	40.7	40.5
plant	5.8	0.8	3.4	4.0	5.7	7.4	6.5	6.4	6.3	5.9	6.2	5.0	5.8	6.4	5.9	4.9	6.6	4.5	6.2	5.0	6.1	6.1	5.2
sheep	24.9	1.6	5.8	26.8	27.7	25.0	27.1	26.7	23.8	25.7	25.6	24.2	23.0	23.4	24.4	22.9	25.6	26.3	24.4	22.6	25.6	25.1	21.9
sofa	17.6	1.7	6.3	18.9	16.0	16.4	17.2	17.0	19.0	13.9	19.4	14.2	16.0	18.3	19.5	19.6	17.2	18.5	17.6	17.3	18.0	20.1	17.1
train	36.2	2.3	10.4	36.4	33.9	32.6	35.8	37.4	37.4	37.4	33.5	36.6	38.4	36.7	37.1	36.2	35.3	35.1	32.5	35.7	35.2	37.4	42.8
tv	34.8	2.6	9.5	33.2	33.0	37.4	34.9	33.0	32.7	33.0	34.8	34.3	34.6	40.6	37.6	35.6	35.4	31.5	31.4	35.8	37.4	31.1	38.7
mAP	26.1	0.3	1.3	26.2	25.8	26.0	26.6	26.3	26.0	26.0	26.1	26.0	26.0	26.4	26.0	26.0	25.7	25.9	25.4	26.1	26.0	26.8	26.7

Table 6.5: Results on 20 different train/val splits of the PASCAL VOC 2011 `train+val` dataset. The configuration used was EHOG + TRUNC + OPTIM + MAX-REG + MIX + ORIENT (BBOX was not used in these experiments).

Table 6.5 shows the results of training and testing the exact same system on 20 different splits of the entire PASCAL VOC 2011 `train+val` datasets. Split 1 is the standard `train`/`val` split that is distributed with the dataset. This split was carefully constructed so that class counts are evenly balanced between the two halves. The remaining 19 splits were generated so that all classes are also evenly split (within 1.5% of perfect balance). The table reports the AUC-AP scores for each class in each split, the mAP over splits for each class, the standard deviation over splits, and the range ($\max - \min$) of scores over the splits. The final row summarizes the mAP score for each split, and then the mean of those summary scores, their standard deviation, and range.

6.4.1 Experimental methodology recommendations

Table 6.5 tells a clear story: with our system (and most likely other methods) AP scores have high variance over random draws of the training and testing data. Comparing two systems on the basis of a single class on a single split of the data is not a good idea.

Testing a single class. To evaluate if a proposed change is meaningful for a single class, the baseline system and the modified system can be run on several train/test splits. For each split, we can pair the baseline AP and the modified system AP and use standard significance testing methods such as a classical or randomized permutation matched-pairs t-test.

Testing multiple classes. The standard deviation of the mAP over splits is much smaller than any of the per-class standard deviations. This suggests a reasonable approach to assess if a modification improves results, on average over all classes, is to collect AP scores on a single split, pair them by class, and then apply a matched-pairs t-test.

6.5 Bounding box prediction (BBOX)

Table 6.5 shows results without BBOX. Since BBOX is a post processing step, we applied it to each of the individual results in that table in order to a get post-BBOX AP scores. Table 6.6 summarizes these results. For each class, we compute the change in AP score as Δ_{BBOX} = post-BBOX AP − pre-BBOX AP. The table shows the means and standard deviations of those changes computed over the 20 training set splits. Similarly, we computed the change in mAP for each split. The table reports the mean and standard deviation of these changes. BBOX helps most for the vehicle classes *bus* and *car*, and offers a small but consistent improvement in many other classes. Over all splits BBOX improves the mAP by 0.67 points on average, with individual changes ranging from 0.41 to 0.84.

		setup: EHOG + TRUNC + OPTIM + MIX + ORIENT																				
		aero	**bike**	**bird**	**boat**	**bottle**	**bus**	**car**	**cat**	**chair**	**cow**	**table**	**dog**	**horse**	**mbike**	**person**	**plant**	**sheep**	**sofa**	**train**	**tv**	**mAP**
Δ_{BBOX}	**mean**	1.54	1.11	-0.03	0.06	0.29	2.34	2.49	1.24	0.14	-0.07	-0.09	0.18	0.41	0.80	0.41	-0.17	0.10	0.37	1.42	0.94	0.67
	std	1.01	0.77	0.18	0.39	0.27	0.87	0.57	0.67	0.30	0.31	0.46	0.30	0.87	0.45	0.44	0.21	0.56	0.51	0.73	0.48	0.14
	min	-0.24	-0.53	-0.49	-0.99	-0.33	0.47	1.50	0.33	-0.43	-0.71	-1.11	-0.30	-1.19	-0.34	-0.97	-0.64	-1.12	-0.88	0.08	-0.02	0.41
	max	3.47	2.07	0.34	0.63	0.90	3.71	3.58	2.76	0.81	0.68	0.99	0.84	1.97	1.70	0.96	0.19	1.35	1.35	2.71	2.10	0.84

Table 6.6: Effect of BBOX averaged over the 20 PASCAL VOC 2011 `train+val` splits. Scores were computed using AUC-AP and are shown as percentages (as in the other tables).

6.6 Regularization: ℓ_2 *vs.* max component (MAX-REG)

Table 6.7 shows the change in AP when training and testing on the first five splits using max component regularization *vs.* ℓ_2 regularization. Positive $\Delta_{\text{MAX-REG}}$ values indicate that max regularization performed better than ℓ_2 regularization. Max regularization consistently produced a small improvement in mAP over the five splits, ranging from 0.16 to 1.17 with a mean improvement of 0.62. The results indicate that max regularization may have a small beneficial effect, though the results are somewhat weak.

For these experiments we fixed the LSVM regularization tradeoff hyper parameters at $C = 0.002$ and $C = 0.006$ for max regularization and ℓ_2 regularization, respectively. We ver-

ified that these were reasonable choices by looking at the range of performance for *aeroplane*, *bicycle*, *cat*, and *horse* over $C \in \{0.001, 0.0013, 0.002, 0.003, 0.004\}$ for max regularization, and $C \in \{0.003, 0.006, 0.009, 0.012\}$ for ℓ_2 regularization.

		setup: EHOG + TRUNC + BBOX + OPTIM + MIX + ORIENT																			
		aero	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	mbike	person	plant	sheep	sofa	train	tv	mAP
$\Delta_{\text{MAX-REG}}$	mean	0.77	2.09	0.16	0.18	-0.55	0.45	0.96	0.43	-0.15	-0.20	0.16	0.15	2.66	3.21	0.02	-0.85	0.82	0.34	2.34	-0.66	0.62
	std	2.94	0.66	0.59	0.47	0.95	1.56	0.32	5.57	0.53	0.96	1.74	3.50	1.03	1.84	1.03	0.66	0.82	1.62	3.08	1.74	0.42
	min	-1.86	1.24	-0.41	-0.54	-1.87	-1.24	0.46	-6.43	-0.64	-1.30	-2.28	-4.77	1.36	-0.00	-1.45	-1.68	-0.10	-1.15	-2.68	-3.65	0.16
	max	5.62	2.71	1.14	0.74	0.42	2.68	1.30	8.12	0.74	0.83	2.37	4.89	3.90	4.50	1.40	-0.20	1.80	2.86	5.64	0.93	1.17

Table 6.7: Effect of MAX-REG averaged over splits 1-5 of our 20 PASCAL VOC 2011 `train+val` splits. Scores were computed using AUC-AP and are shown as percentages (as in the other tables).

6.7 LSVM *vs.* WL-SSVM

The primary technical difference between WL-SSVM and LSVM is that WL-SSVM uses a *structural* loss, rather than a *binary classification* loss. Intuitively, this means that the WL-SSVM objective function aims to create a margin between the score of the best output for an example and the score of any other output. The margin requirement is determined by the structural loss function.

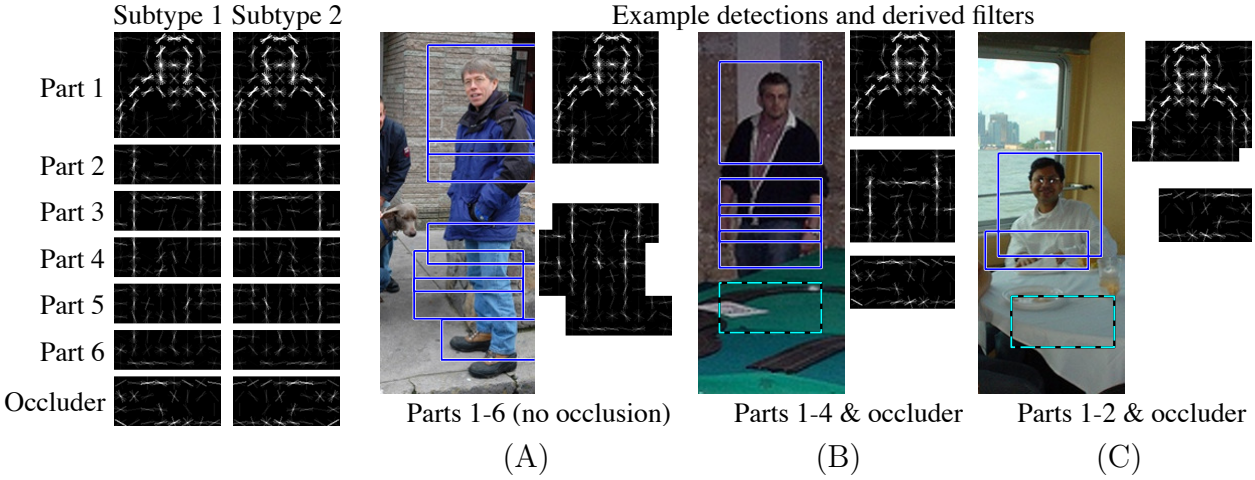


Figure 6.1: A model where we would expect the structural loss used is WL-SSVM to produce a better results than the binary classification loss used by LSVM. See the text for more details.

We should only expect WL-SSVM to outperform LSVM when the use of a structural loss is important for producing good detections. Figure 6.1 shows the person grammar model from Chapter 5 where we expect a structural loss to improve detection results. Observe how detections (B) and (C) — if pasted on to (A) — would be "good" detections for (A) in the sense that they correctly detect a *portion* of the visible person. However, these detections are "bad" in the sense that they do not describe the visible portion of the person as accurately as the detection in (A) does. We would like to learn a model that outputs a derivation for a fully visible person, when the person is fully visible.

The structure of the person grammar model allows it to generate outputs that are "subsets" of each other, in the sense that one derivation may place a subset of the filters that are placed by another derivation. This allows the model to capture the visibility patterns in the data more accurately. At the same time, however, it also increases the space of possible errors that the model can make. The structural loss in WL-SSVM provides a mechanism for preventing this new type of error from eroding the gain that should come from increasing the model's representational capacity.

During training with WL-SSVM, the structural loss function will attempt to create a margin between the score of the detection in (A) and the scores of alternative outputs, such as outputs that correspond to partially visible people, as in (B) and (C). In contrast, if we were training with a LSVM, the objective would try to make the detections in (A), (B), and (C) each score +1. Then, at test time, the model would be more likely to make a mistake since the score of the full person output might tie with the scores of the partial person outputs. Section 3.1.4 has a more in-depth discussion of the relationship between LSVM and WL-SSVM.

Table 6.8 compares the effect of training the person grammar with LSVM *vs.* WL-SSVM on splits 1-5 of our 20 PASCAL 2011 splits. Training with WL-SSVM boosts the AP score by 2 points on average. The improvement is consistent across all five splits.

Now we look at the application of WL-SSVM to our mixture models for the twenty

Person grammar		AP scores on splits				
objective	**mAP**	**1**	**2**	**3**	**4**	**5**
LSVM	46.0	47.5	46.3	45.3	45.3	45.6
WL-SSVM	48.0	49.0	48.4	48.4	48.0	46.5

Table 6.8: Effect of WL-SSVM on the person grammar model using the first five PASCAL 2011 `train/val` splits. Scores were computed using AUC-AP and are shown as percentages (as in the other tables).

PASCAL categories. It is not clear, *a priori*, that WL-SSVM should improve results. For most categories, it does not seem that the mixture components could be easily confused with each other. Unlike in the person grammar model, the mixture components are not "visual subsets" of each other, with the possible exception of the person mixture model.

Table 6.9 shows the change in performance when WL-SSVM instead of LSVM on the PASCAL 2007 dataset. Positive $\Delta_{\text{WL-SSVM}}$ values indicate that WL-SSVM performed better than LSVM. If WL-SSVM offers a performance advantage for the mixture models, the effect size is likely quite small.

	setup: EHOG + TRUNC + BBOX + OPTIM + MIX + ORIENT + MAX-REG																			
	aero	**bike**	**bird**	**boat**	**bottle**	**bus**	**car**	**cat**	**chair**	**cow**	**table**	**dog**	**horse**	**mbike**	**person**	**plant**	**sheep**	**sofa**	**train**	**tv**	**mAP**
$\Delta_{\textbf{WL-SSVM}}$	4.17	-0.36	1.11	1.07	1.14	1.22	0.62	-0.69	-2.55	0.51	-3.82	1.92	2.03	-1.15	0.45	-0.94	1.52	-1.65	-0.33	-0.19	0.20

Table 6.9: Effect of WL-SSVM on PASCAL VOC 2007. Scores were computed using AUC-AP and are shown as percentages (as in the other tables).

CHAPTER 7

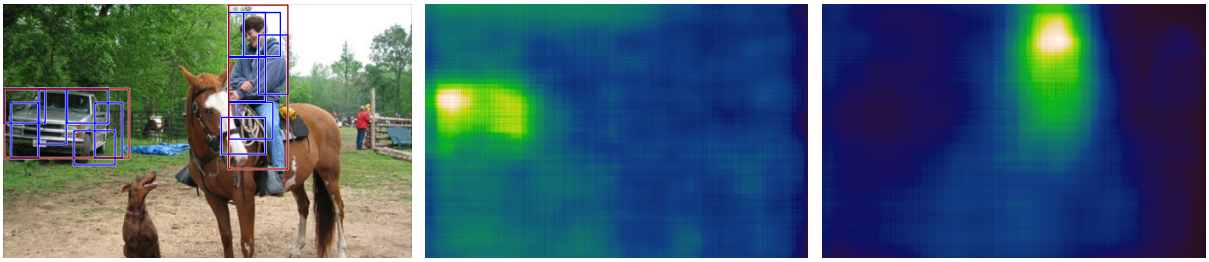
CASCADED DETECTION

We describe a general method for building cascade classifiers from part-based deformable models such as pictorial structures. We focus primarily on the case of star-structured models and show how a simple algorithm based on partial hypothesis pruning can speed up object detection by more than one order of magnitude without sacrificing detection accuracy. In our algorithm, partial hypotheses are pruned with a sequence of thresholds. In analogy to probably approximately correct (PAC) learning, we introduce the notion of probably approximately admissible (PAA) thresholds. Such thresholds provide theoretical guarantees on the performance of the cascade method and can be computed from a small sample of positive examples. Finally, we outline a cascade detection algorithm for a general class of models defined by a grammar formalism. This class includes not only tree-structured pictorial structures but also richer models that can represent each part recursively as a mixture of other parts.

7.1 Introduction

A popular approach for object detection involves reducing the problem to binary classification. The simplest and most common example of this approach is the sliding window method. In this method a classifier is applied at all positions, scales, and, in some cases, orientations of an image. However, testing all points in the search space with a non-trivial classifier can be very slow. An effective method for addressing this problem involves applying a cascade of simple tests to each hypothesized object location to eliminate most of them very quickly [70, 38, 21, 64, 10, 39].

Another line of research, separate from cascade classifiers, uses part-based deformable models for detection. In this case an object hypothesis specifies a configuration of parts, which leads to a very large (exponential) hypothesis space. There has been significant success



(a) Input image and detections (b) Car workmap (c) Person workmap

Figure 7.1: Visualization of the amount of work performed by our algorithm over different regions of an image (a) using a car model (b) and a person model (c). Note how the computational effort is concentrated around the regions containing the target object class.

in algorithmic methods for searching over these large hypothesis spaces, including methods that are "asymptotically optimal" for tree-structured models [30]. However, these methods are still relatively slow when compared to simple classifiers defined by cascades. In this chapter we describe a method for building cascades for part-based deformable models such as pictorial structures. In the most general setting, this method leads to a cascade version of top-down dynamic programming for a general class of grammar based models.

We focus primarily on the case of star-structured models due to their recent strong performance on difficult benchmarks such as the PASCAL datasets [34, 32, 23, 24, 25]. For star models, we obtain a fairly simple algorithm for early hypothesis pruning. This algorithm leads to a detection method over 20 times faster than the standard detection algorithm, which is based on dynamic programming and generalized distance transforms, without sacrificing detection accuracy. Figure 7.1 illustrates the amount of work done by our algorithm in different areas of an image using two different models.

As described in [34, 32], detection with a deformable part model can be done by considering all possible locations of a distinguished "root" part and, for each of those, finding the best configuration of the remaining parts. In this case we need to compute an optimal configuration for each location of the root. These problems are not independent because the possible locations of the remaining parts are shared among different root locations. For tree-structured models one can use dynamic programming to account for this sharing [30].

In practice one is only interested in root locations that lead to high scoring configurations. The basic idea of our algorithm is to use a hierarchy of models defined by an ordering of the original model's parts. For a model with $n + 1$ parts, including the root, we obtain a sequence of $n + 1$ models. The i-th model in this sequence is defined by the first i parts from the original model. Using this hierarchy, we can prune low scoring hypotheses after looking at the best configuration of a subset of the parts. Hypotheses that score high under a weak model are evaluated further using a richer model. This process is analogous to a classical cascade and is similar to the cascades of [10, 64] in that the score of a weaker model is reused when computing the score of a richer one. However, when using deformable part models individual applications of the cascade are not independent, so, in analogy to classical dynamic programming, work done evaluating one hypothesis is also reused when evaluating others.

Our sequential search for parts is related to [1], where the authors propose a sequential search for semi-local features that fit a global arrangement. The work in [1] also considered the problem of selecting parameters that lead to fast search with a low false negative rate, by making some assumptions on the form of the distribution of local features and analyzing statistics of training data. We use an alternative approach (see below) that makes fewer assumptions and relies more heavily on the training data.

The time it takes to evaluate a hypothesis for a part-based model is highly dependent on the complexity of the individual part models. Besides simplifying a model by removing some of its parts, we also consider simplifications that arise from replacing the original part appearance models with simpler ones that are much faster to compute. In this case, for a model with $n + 1$ parts we get a hierarchy of $2(n + 1)$ models. The first $n + 1$ models are obtained by sequentially adding parts with simplified appearance models. The second $n + 1$ models are obtained by sequentially replacing each simplified appearance model with its full one.

Our algorithm prunes partial hypotheses using thresholds on their scores. Admissible

thresholds would not prune any partial hypothesis that leads to a complete detection scoring above a global threshold. We define the error of a set of thresholds to be the fraction of full hypotheses scoring above the global threshold that are incorrectly pruned. To select pruning thresholds, we introduce the notion of probably approximately admissible (PAA) thresholds. PAA thresholds have a low error with high probability.

We show that PAA thresholds can be obtained by looking at statistics of partial hypothesis scores over positive examples. This leads to a simple method for picking *safe* and *effective* thresholds. The thresholds are safe because they have low error with high probability. They are effective because they lead to a fast cascade with significant pruning.

[30] notes that by using dynamic programming and distance transforms the relationships among parts in a tree-structured model can be taken into account "for free." That is, it takes very little additional time to detect whole object configurations as opposed to individually detecting parts on their own. Our results push this idea further. In practice we find that it is possible to detect whole object configurations much faster than detecting each individual part.

7.2 Object detection with star models

We start by defining a general framework for object detection with star-structured deformable part models that includes the setting in [34, 32].

Let M be a model with a root part v_0 and n additional parts $v_1, \ldots, v_n$. Let Ω be a space of locations for each part within an image. For example, $\omega \in \Omega$ could specify a position and scale. Let $m_i(\omega)$ be the score for placing v_i in location ω. This score depends on the image data, but we assume the image is implicitly defined to simplify notation.

For a non-root part, let $a_i(\omega)$ specify the ideal location for v_i as a function of the root location. Let Δ be a space of displacements, and let $\oplus : \Omega \times \Delta \to \Omega$ be a binary operation taking a location and a displacement to a "displaced location." Let $d_i(\delta)$ specify a deformation cost for a displacement of v_i from its ideal location relative to the root.

An object configuration specifies a location for the root and a displacement for each additional part from its ideal location relative to the root. The score of a configuration is the sum of the scores of the parts at their locations minus deformation costs associated with each displacement.

$$\text{score}(\omega, \delta_1, \ldots, \delta_n) = m_0(\omega) + \sum_{i=1}^{n} m_i(a_i(\omega) \oplus \delta_i) - d_i(\delta_i) \tag{7.1}$$

We can define an overall score for a root location based on the maximum score of a configuration rooted at that location. In a star model each part is only attached to the root, so the score can be factored as follows.

$$\text{score}(\omega) = m_0(\omega) + \sum_{i=1}^{n} \text{score}_i(a_i(\omega)) \tag{7.2}$$

$$\text{score}_i(\eta) = \max_{\delta_i \in \Delta} (m_i(\eta \oplus \delta_i) - d_i(\delta_i)) \tag{7.3}$$

Here $\text{score}_i(\eta)$ is the maximum, over displacements of the part from its ideal location, of the part score minus the deformation cost associated with the displacement.

For the models in [34, 32], $m_i(\omega)$ is the response of a filter in a dense feature pyramid, and $d_i(\delta)$ is a (separable) quadratic function of δ. To detect objects [34, 32] look for root locations with an overall score above some threshold, $\text{score}(\omega) \geq T$. A dynamic programming algorithm is used to compute $\text{score}(\omega)$ for *every* location $\omega \in \Omega$. Using the fast distance transforms method from [30] the detection algorithm runs in $O(n|\Omega|)$ time if we assume that evaluating the appearance model for a part at a specific location takes $O(1)$ time. In practice, evaluating the appearance models is the bottleneck of the method.

7.3 Star-cascade detection

Here we describe a cascade algorithm for star models that uses a sequence of thresholds to prune detections using subsets of parts.

Note that we are only interested in root locations where $\text{score}(\omega) \geq T$. By evaluating

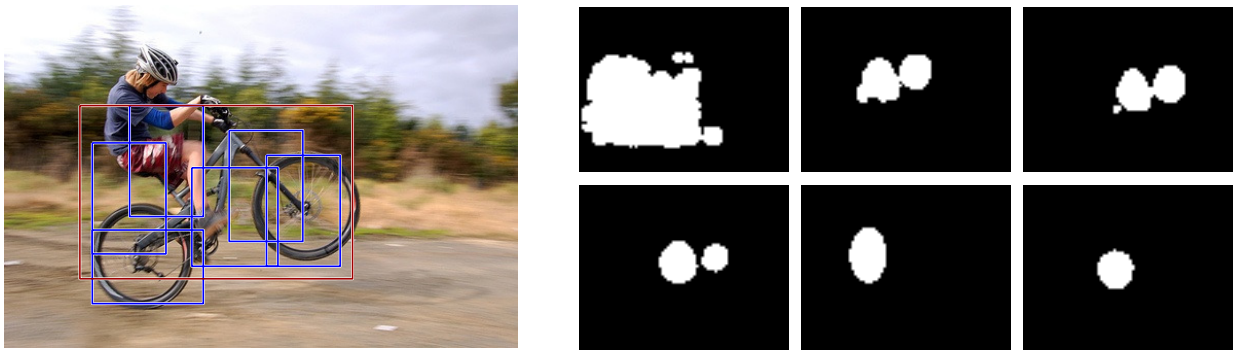


Figure 7.2: A sample image with a bicycle detection (left). Each image in the right shows (in white) positions where a particular part appearance model was evaluated at the scale of the bicycle detection. The images are shown in "cascade order" from left to right and top to bottom. The image for the root part, which was evaluated first, is not shown because its appearance model is evaluated at all locations. After evaluating the first non-root part, nearly all locations were pruned by the cascade.

parts in a sequential order we can avoid evaluating the appearance model for most parts almost everywhere. For example, when detecting people we might evaluate the score of the head part at each possible location and decide that we do not need to evaluate the score of the torso part for most locations in the image. Figure 7.2 shows an example run of the algorithm.

We use $\tilde{m}_i(\omega)$ to denote a memoized version of $m_i(\omega)$. In a memoized function whenever a value is computed we store it to avoid recomputing it later. Memoization can be implemented by maintaining an ω-indexed array of already-computed values and checking in this array first whenever $\tilde{m}_i(\omega)$ is called to avoid computing the appearance model at the same location more than once.

The cascade algorithm (Algorithm 1) for a star-structure model with $n+1$ parts takes a global threshold T and a sequence $2n$ of intermediate thresholds. To simplify the presentation we assume the root appearance model is evaluated first even though this ordering is not a requirement.

For each root location $\omega \in \Omega$, we evaluate score(ω) in n stages. The variable s accumulates the score over stages. In the i-th stage we compute score$_i(\eta)$, the contribution of part v_i, using the variable p. During evaluation of score(ω) there are two opportunities for pruning.

Hypothesis pruning. If the score at ω with the first i parts is below t_i, then the hypothesis at ω is pruned without evaluating parts v_i through v_n (line 5). Intuitively, placing the remaining parts will not make score(ω) go above T.

Deformation pruning. To compute v_i's contribution we need to search over deformations $\delta \in \Delta$. The algorithm will skip a particular δ if the score of the first i parts minus $d_i(\delta)$ is below t'_i (line 8). Intuitively, displacing v_i by δ costs too much to allow the score(ω) to go above T.

Note that memoizing the appearance models is important because several root locations might want to evaluate $m_i(\omega)$ at the same location.

For a fixed global threshold T, any input to Algorithm 1 that correctly computes score(ω) whenever it is above T is called a T-admissible set of thresholds. If given T-admissible thresholds, Algorithm 1 returns exactly the same set of detections as the standard dynamic programming algorithm. In the next section we investigate the case of *good inadmissible* thresholds that produce a cascade with a low error rate but still allow aggressive pruning.

The worst-case time of Algorithm 1 is $O(n|\Omega||\Delta|)$ if $m_i(\omega)$ is taken to cost $O(1)$, which is slower than the standard dynamic programming algorithm with distance transforms. However, in practice, for typical models Δ can be safely made relatively small (*cf.* Section 7.6). Moreover, searching over Δ is usually no more expensive than evaluating $m_i(\omega)$ at a single location, because the spatial extent of a part is of similar size as its range of displacement. The worse-case time of both methods is the same, $O(n|\Omega||\Delta|)$, if we assume evaluating $m_i(\omega)$ takes $O(|\Delta|)$ time.

Data: Thresholds $((t_1, t'_1), \ldots, (t_n, t'_n))$ and T

Result: Set of detections D

$D \leftarrow \emptyset$
for $\omega \in \Omega$ **do**
  $s \leftarrow \tilde{m}_0(\omega)$
  for $i = 1$ **to** n **do**
    if $s < t_i$ **then** skip ω
    $p \leftarrow -\infty$
    for $\delta \in \Delta$ **do**
      if $s - d_i(\delta) < t'_i$ **then** skip δ
      $p \leftarrow \max(p, \tilde{m}_i(a_i(\omega) \oplus \delta) - d_i(\delta))$
    end
    $s \leftarrow s + p$
  end
  if $s \geq T$ **then** $D \leftarrow D \cup \{\omega\}$
end
return D

Algorithm 1: star-cascade

Figure 7.2 shows how, in practice, the cascade algorithm avoids evaluating $m_i(\omega)$ for most locations $\omega \in \Omega$ except for one or two parts.

7.4 Pruning thresholds

Suppose we have a model M and a detection threshold T. Let $x = (\omega, I)$ be an example of a location within an image I where score$(\omega) \geq T$. Let $\mathcal{D}$ be a distribution over such examples.

For a sequence of thresholds $t = ((t_1, t'_1), \ldots, (t_n, t'_n))$ let csc-score(t, ω) be the score computed for ω by the cascade algorithm using t. If the cascade prunes ω, then we set csc-score$(t, \omega) = -\infty$.

We define the *error* of t on $\mathcal{D}$ as the probability that the cascade algorithm will incorrectly compute score(ω) on a random example from $\mathcal{D}$,

$$\text{error}(t) = P_{x \sim \mathcal{D}}(\text{csc-score}(t, \omega) \neq \text{score}(\omega)). \tag{7.4}$$

We would like to find a sequence of thresholds that has a small error. Note that in practice we are only interested in having a small error on positive examples. In particular we do not care if the cascade incorrectly prunes a negative example that scores above T. Thus we can take $\mathcal{D}$ to be a distribution over high-scoring positive examples.

Below we show that we can learn a good sequence of thresholds by looking at a small number of examples from $\mathcal{D}$. In analogy to PAC learning [45] we would like to select thresholds that have a small error with high probability.

We say a sequence of thresholds t is (ϵ, δ) *probably approximately admissible* (PAA) if the probability that the error of t is greater than ϵ is bounded by δ,

$$P(\text{error}(t) > \epsilon) \leq \delta. \tag{7.5}$$

Let $\delta_1, \ldots, \delta_n$ be the optimal displacements for the non-root parts of M on an example $x = (\omega, I)$. We can define partial scores that take into account the first i parts and the first i parts minus the i-th deformation cost,

$$x_i = m_0(\omega) + \sum_{j=1}^{i-1} m_j(a_j(\omega) + \delta_j) - d_j(\delta_j), \tag{7.6}$$

$$x'_i = x_i - d_i(\delta_i). \tag{7.7}$$

The star-cascade algorithm will find an optimal configuration and score for x if and only if $x_i \geq t_i$ and $x'_i \geq t'_i$ for all $1 \leq i \leq n$.

Let X be m independent samples from $\mathcal{D}$. We can select thresholds by picking,

$$t_i = \min_{x \in X} x_i \qquad t'_i = \min_{x \in X} x'_i \tag{7.8}$$

These are the tightest thresholds that make no mistakes on X. The following theorem shows they also have low error with high probability provided that m is sufficiently large.

Theorem 1. *If we select t according to Eq. 7.8 using $m \geq 2n/\epsilon \ln(2n/\delta)$ samples from $\mathcal{D}$ then t is (ϵ, δ) probably approximately admissible with respect to $\mathcal{D}$.*

Proof. By a union bound $\text{error}(t) \leq \epsilon$ if $P_{x \sim \mathcal{D}}(x_i < t_i)$ and $P_{x \sim \mathcal{D}}(x'_i < t'_i)$ are $\leq \epsilon/(2n)$ for all i. Thus $\text{error}(t) \leq \epsilon$ unless for some i all m samples of the x_i or x'_i are above the $\epsilon/(2n)$-th percentile of their distribution. The probability of that event is $\leq 2n(1-\epsilon/(2n))^m$. To bound this by δ we only need $m \geq 2n/\epsilon \ln(2n/\delta)$ samples. □

7.5 Simplified part appearance models

So far we have considered a cascade for star models that is defined by a hierarchy of $n+1$ models. Given a predefined part order, the i-th model is formed by adding the i-th part to the $(i-1)$-st model. The goal of the cascade is to detect objects while making as few appearance model evaluations as possible. The star-cascade algorithm achieves this goal by pruning hypotheses using intermediate scores.

A complementary approach is to consider a simplified appearance model for each part, $\hat{m}_i(\omega)$, that computes an inexpensive approximation of $m_i(\omega)$.

A hierarchy of $2(n+1)$ models can be defined with the first $n+1$ models constructed as before, but using the simplified appearance models, and the second $n+1$ models defined by sequentially removing one of the remaining simplified appearance models $\hat{m}_i$ and replacing it with the original one m_i.

With simplified parts, the star-cascade operates as before for stages 1 through $n+1$, except that pruning decisions are based on the (memoized) evaluations of the simplified appearance models $\hat{m}_i(\omega)$. During the remaining stages, full appearance models replace simplified ones. When replacing an appearance model, the algorithm must redo the search over the deformation space because the optimal placement may change. But now we will

often prune a hypothesis before evaluating any of the expensive appearance models.

This version of the algorithm requires $4n + 1$ intermediate thresholds. Just as before, these thresholds can be selected using the method from the previous section.

For the models in [34, 32], simplified appearance models can be defined by projecting the HOG features and the weight vectors in the part filters to a low dimensional space. We did PCA on a large sample of HOG features from training images in the PASCAL datasets. A simplified appearance model can be specified by the projection into the top k principal components. For the 31-dimensional HOG features used in [32], a setting of $k = 5$ leads to appearance models that are approximately 6 times faster to evaluate than the original ones. This approach is simple and only introduces a small amount of overhead — the cost of projecting each feature vector in the feature pyramid onto the top k principal components.

7.6 General grammar models

Here we consider a fairly general class of grammar models for representing objects in terms of parts. It includes tree-structured pictorial structure models as well as more general models that have variable structure. For example, we can define a person model in which the face part is composed of eyes, a nose, and either a smiling or frowning mouth. We follow the framework and notation in [33].

Let $\mathcal{N}$ be a set of nonterminal symbols and $\mathcal{T}$ be a set of terminal symbols. Let Ω be a set of possible locations for a symbol within an image. For $\omega \in \Omega$ we use $X(\omega)$ to denote the placement of a symbol at a location in the image.

Appearance models for the terminals are defined by a function $\text{score}(A, \omega)$ that specifies a score for $A(\omega)$. The appearance of nonterminals is defined in terms of expansions into other symbols. Possible expansions are defined by a set of scored production rules of the form

$$X(\omega_0) \xrightarrow{s} Y_1(\omega_1), \ldots, Y_n(\omega_n), \tag{7.9}$$

where $X \in \mathcal{N}$, $Y_i \in \mathcal{N} \cup \mathcal{T}$, and $s \in \mathbb{R}$ is a score.

To avoid enumerating production rules that differ only by symbol placement, we define grammar models using a set of parameterized production schemas of the form

$$X(\omega_0(z)) \overset{\alpha(z)}{\to} Y_1(\omega_1(z)), \ldots, Y_n(\omega_n(z)). \tag{7.10}$$

Each schema defines a collection of productions consisting of one production for each value of a parameter z. Given a fixed value of z, the functions $\omega_0(z), \ldots, \omega_n(z)$, and $\alpha(z)$ yield a single production of the form in Eq. 7.9.

Star models can be represented using a nonterminal X_i and a terminal A_i for each part. We have $\text{score}(A_i, \omega) = m_i(\omega)$. A placement of the root nonterminal X_0 defines ideal locations for the remaining parts. This is captured by an instance of the following production for each ω,

$$X_0(\omega) \overset{0}{\to} A_0(\omega), X_1(a_1(\omega)), \ldots, X_n(a_n(\omega)). \tag{7.11}$$

We can encode these productions using a schema where z ranges over Ω. These rules are called *structural rules*.

A part can be displaced from its ideal location at the expense of a deformation cost. This is captured by an instance of the following production for each ω and δ,

$$X_i(\omega) \overset{-d_i(\delta)}{\to} A_i(\omega \oplus \delta). \tag{7.12}$$

We can encode these productions using a schema where z ranges over $\Omega \times \Delta$. These rules are called *deformation rules*.

We restrict our attention to acyclic grammars. We also require that no symbol may appear in the right hand side of multiple schemas. We call this class *no-sharing acyclic grammars*. It includes pictorial structures defined by arbitrary trees as well as models where each part can be one of several subtypes. But it does not include models where a single part is used multiple times in one object instance such as a car model where one wheel part is

used for both the front and rear wheels.

For acyclic grammars, we can extend the scores of terminals to scores for nonterminals by the recursive equation

$$\text{score}(X, \omega) = \max_{X(\omega) \xrightarrow{s} Y_1(\omega_1), \ldots, Y_n(\omega_n)} s + \sum_{i=1}^{n} \text{score}(Y_i, \omega_i), \tag{7.13}$$

where the max is over rules with $X(\omega)$ in the left hand side. Since the grammar is acyclic, the symbols can be ordered such that a bottom-up dynamic programming algorithm can compute score tables $V_X[\omega] = \text{score}(X, \omega)$.

Scores can also be computed using a recursive top-down procedure. To compute $\text{score}(X, \omega)$ we consider every rule with $X(\omega)$ in the left hand side and sequentially compute the scores of placed symbols in the right hand side using recursive calls. Computed scores should be memoized to avoid recomputing them in the future.

For object detection we have a root symbol S and we would like to find all locations ω where $\text{score}(S, \omega) \geq T$. It is natural to introduce pruning into the top-down algorithm in analogy to the star-cascade method (Algorithm 1).

As the top-down method traverses derivations in depth-first left-right order, we can keep track of a "prefix score" for the current derivation. Upon reaching $X(\omega)$ we can compare the current prefix score to a threshold $t(X)$. If the prefix score is below $t(X)$, then we could pretend $\text{score}(X, \omega) = -\infty$ without computing it. This is a form of pruning. However, generally there will be multiple requests for $\text{score}(X, \omega)$ and pruning may be problematic when memoized scores are reused. The value memoized for $X(\omega)$ depends on the prefix score of the first request for $\text{score}(X, \omega)$. Due to pruning, the memoized value might be different than what a later request would compute. In particular, if a later request has a higher prefix score, the associated derivation should undergo less pruning.

To address this issue, we define $\text{pscore}(Y, \omega)$ to be the maximum prefix score over all requests for $Y(\omega)$. In a grammar with no sharing there is a single schema with Y in the

right hand side. For each schema of the form in Eq. 7.10 we have

$$\text{pscore}(Y_i, \omega) = \max_{z \in \omega_i^{-1}(\omega)} \text{pscore}(X, \omega_0(z)) + \alpha(z) + \sum_{j=1}^{i-1} \text{score}(Y_j, \omega_j(z)). \tag{7.14}$$

Thus pscores for Y_i can be computed once we have pscores for X and scores for $Y_1, \ldots, Y_{i-1}$. The set of parameter values z yielding $\omega = \omega_i(z)$ is denoted by $\omega_i^{-1}(\omega)$.

The grammar-cascade algorithm goes over schemas in a depth-first left-right order. It computes $\text{pscore}(X, \omega)$ before computing $\text{score}(X, \omega)$, and it prunes computation by comparing $\text{pscore}(X, \omega)$ to a threshold $t(X)$.

The procedure `compute` takes as input a symbol X and a table of prefix scores $P_X[\omega] \approx \text{pscore}(X, \omega)$. It returns a table of values $V_X[\omega] \approx \text{score}(X, \omega)$. These tables are not exact due to pruning. As in the star-cascade, we can pick thresholds using a sample of positive examples. For each symbol X we can pick the highest threshold $t(X)$ that does not prune optimal configurations on the positive examples.

Data: X, P_X

Result: V_X

1 $V_X[\omega] \leftarrow -\infty$

2 **if** $X \in \mathcal{T}$ **then**

3 **if** $P_X[\omega] \geq t(X)$ **then** $V_X[\omega] \leftarrow \text{score}(X, \omega)$

4 **return** V_X

5 **end**

6 **foreach** $X(\omega_0(z)) \overset{\alpha(z)}{\rightarrow} Y_1(\omega_1(z)), \ldots, Y_n(\omega_n(z))$ **do**

7 $V_0[\omega] \leftarrow -\infty$

8 **if** $P_X[\omega] \geq t(X)$ **then** $V_0[\omega] \leftarrow P_X[\omega]$

9 **for** $i = 1$ **to** n **do**

10 $P_i[\omega] \leftarrow \max_{z \in \omega_i^{-1}(\omega)} \alpha(z) + \sum_{j=0}^{i-1} V_j[\omega_j(z)]$

11 $V_i \leftarrow \texttt{compute}(Y_i, P_i)$

12 **end**

13 $V[\omega] \leftarrow \max_{z \in \omega_0^{-1}(\omega)} \alpha(z) + \sum_{j=1}^{n} V_j[\omega_j(z)]$

14 $V_X[\omega] \leftarrow \max(V_X[\omega], V[\omega])$

15 **end**

16 **return** V_X

Procedure compute

For a terminal X, `compute` evaluates $\text{score}(X, \omega)$ at locations ω with high pscores. For a nonterminal, `compute` loops over schemas with X in the left hand side. Line 10 computes pscores for Y_i before calling `compute` recursively to obtain scores for Y_i. Line 13 computes scores for X under a particular schema. The result V_X is the running max of scores under different schemas.

To understand the worst case runtime of this algorithm we need to consider the max over z that appears in lines 10 and 13. Suppose each schema is a structural rule similar to Eq. 7.11 with a bounded number of symbols in the right hand side or a deformation rule

similar to Eq. 7.12.

For a structural rule Eq. 7.11, $\omega_0^{-1}(\omega) = \{\omega\}$. In this case, line 13 simply sums the scores of the right hand side symbols after shifting each by its ideal displacement $a_i(\omega)$. For a deformation rule Eq. 7.12, $\omega_0^{-1}(\omega) = \{\omega\} \times \Delta$. In this case, line 13 takes a max over Δ. The situations are similar for line 10. Thus, assuming it takes $O(1)$ time to evaluate score(A, ω) for a terminal, the runtime of the algorithm is $O(|\Omega||\Delta|)$ per schema. When scores over deformation rules can be computed via fast distance transforms the runtime becomes $O(|\Omega|)$ per schema.

Detection is performed by calling `compute` on a root symbol S with $P_S[\omega] = 0$. In the case of tree-structured pictorial structure models where fast distance transforms can be used, the time of `compute`(S, P_S) is $O(n|\Omega|)$ for a model with n parts. This is the same as bottom-up dynamic programming with fast distance transforms. In this case the grammar-cascade algorithm has better worst-case time than the star-cascade algorithm, but by implementing both methods we found that the specialized star-cascade algorithm outperformed `compute` by about a factor of two. This empirical result comes from a confluence of two factors: the restricted structure of the star model avoids the need to maintain prefix score tables, and for the models we consider $|\Delta|$ is small enough that brute force search over it, for a small number of locations, outperforms fast distance transforms over the full space Ω.

7.7 Experimental results

To evaluate our algorithm for star-structured models we compared it to the baseline detection method based on dynamic programming and distance transforms. We used the publicly available system from [32] as a testbed. We note that [32] provides an implementation of the baseline detection algorithm that is already quite efficient.

We evaluated our algorithm by looking at the detection time speedup and average precision (AP) score with respect to the baseline. The evaluation was done over all 20 classes of the PASCAL 2007 dataset [23] as well as on the INRIA Person dataset [18]. For the

PASCAL experiments we obtained the six-component models used by the UoC-TTI entry in the 2009 PASCAL VOC Challenge [25]. For the INRIA experiments we obtained the one-component model from [32]. These models achieve state-of-the-art detection results. Our experiments show that the cascade algorithm achieves a significant speedup, of more than 20 times on average, with negligible decrease in detection accuracy.

The PASCAL models were trained on the 2009 training and validation data, which includes the 2008 data as a subset. We wanted "fresh" positive training examples for selecting thresholds, separate from the examples used to train the models, so we conducted our evaluation on the PASCAL 2007 dataset. Testing on the 2007 dataset ensured that the statistics for the threshold training and test data were the same. Note that testing on the 2007 dataset using models trained on the 2009 dataset might not lead to the best possible detection accuracy, but we are only interested in the relative performance of the cascade and the baseline method.

In the case of the INRIA Person dataset we did not have access to fresh positive examples, so we used the same examples with which the model was trained. Even though the PAA threshold theory does not apply in this setting, the cascade achieved exactly the same AP scores as the baseline.

Our implementation of the cascade algorithm has a single parameter controlling the number of components used for the PCA approximation of the low-level features. This was set to 5 in advance based on the magnitude of the eigenvalues from the PCA of HOG features.

We compared the runtime of the cascade algorithm versus the baseline for two global detection threshold settings. A higher global threshold allows for more pruning in the cascade at the cost of obtaining a lower recall rate. The first setting was selected so that the resulting precision-recall curve should reach the precision-equals-recall point. Empirically we found that this setting results in a detector with typical AP scores within 5 points of the maximum score. This setting is tuned for speed without sacrificing too much recall. The

second setting results in the maximum possible AP score with less emphasis on speed. This configuration requires picking a global threshold so the detector achieves its full recall range. We approximated this goal by selecting a global threshold such that the detector would return results down to the precision equals 5% level. For each global detection threshold we picked pruning thresholds using the procedure outlined in Section 7.4.

Figures 7.3 and 7.4 illustrate precision-recall curves obtained with the cascade and baseline methods. The performance of the cascade algorithm follows the performance of the baseline very closely. The complete experimental results are summarized in Tables 7.1 and 7.2. We see that the cascade method achieves AP scores that are essentially identical to the baseline for both global threshold settings. Sometimes the cascade achieves slightly higher AP score due to pruning of false positives. The maximum recall obtained with the cascade is only slightly below the baseline, indicating that very few true positives were incorrectly pruned. The difference in recall rates is reported as the *recall gap*.

	aero	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	mbike	person	plant	sheep	sofa	train	tv	inria
Speedup factor	22.7	22.1	16.5	11.6	22.1	36.0	13.3	25.6	23.4	23.2	29.8	15.2	16.2	32.6	12.7	23.3	32.8	18.1	23.3	27.2	13.5
Baseline AP	21.1	43.1	10.6	12.2	24.0	42.2	48.0	15.9	13.4	19.0	7.1	10.7	31.3	32.9	34.4	12.0	20.3	20.8	29.3	36.3	80.1
Cascade AP	21.1	42.9	10.4	12.4	24.1	42.5	48.1	15.5	13.4	19.0	8.0	10.7	31.3	33.0	34.8	12.0	20.3	20.2	28.8	36.5	80.1
Recall gap	0.7	3.9	1.1	3.0	0.4	4.7	1.3	2.0	1.2	2.0	1.5	1.8	0.9	1.5	0.0	1.0	3.7	1.7	4.6	0.3	0.3

Table 7.1: Results for the global threshold set so each PR curve would reach the precision = recall point. Mean speedup for all classes = 22.0.

	aero	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	mbike	person	plant	sheep	sofa	train	tv	inria
Speedup factor	13.6	18.4	16.9	9.9	12.3	19.0	10.1	13.6	13.4	15.1	19.1	12.0	13.1	21.5	5.6	11.6	23.9	15.9	14.3	9.8	11.1
Baseline AP	22.8	49.4	10.6	12.9	27.1	47.4	50.2	18.8	15.7	23.6	10.3	12.1	36.4	37.1	37.2	13.2	22.6	22.9	34.7	40.0	85.6
Cascade AP	22.7	49.3	10.6	13.0	26.6	47.4	50.2	18.8	15.7	23.1	11.3	12.3	36.0	37.1	37.6	13.6	22.7	23.1	34.2	40.0	85.6
Recall gap	0.4	1.2	0.2	0.8	1.5	2.3	0.7	0.8	0.1	1.6	0.0	1.0	1.4	1.8	0.0	2.3	3.7	1.7	2.1	0.0	0.7

Table 7.2: Results for the global threshold set so each PR curve would reach precision = 0.05. Mean speedup for all classes = 14.3.

For the purpose of timing the algorithms, we ignored the time it takes to compute the low-level feature pyramid from an image as that is the same for both methods (and can be shared among different detectors). Feature pyramid generation took an average of 459ms per image on the PASCAL dataset and 730ms on the INRIA dataset. With the precision-

equals-recall threshold, the cascade detector ran 22 times faster than the baseline on average. As an example, the mean detection time per image for the motorbike model was 10.1s for the baseline versus 313ms for the cascade, and the mean time per image for the person model was 8.5s for the baseline versus 682ms for the cascade.

We also tested the star-cascade algorithm without PCA filters. In this mode, the mean speedup dropped to 8.7 over the baseline at the precision-equals-recall level.

Note that [32] reports detection times of around 2 seconds per image because it uses a parallel implementation of the baseline algorithm. We turned off that feature to facilitate comparison. Both the baseline and the cascade are equally easy to parallelize. For example, one could search over different scales at the same time. All experiments were conducted using single-threaded implementations on a 2.67GHz Intel Core i7 920 CPU computer running Linux.

7.8 Discussion

The results of this chapter are both theoretical and practical. At a theoretical level we have shown how to construct a cascade variant of a dynamic programming (DP) algorithm. From an abstract viewpoint, a DP algorithm fills values in DP tables. In the cascade version the tables are partial — not all values are computed. Partial DP tables are also used in A* search algorithms. However, the cascade variant of DP runs without the overhead of priority queue operations and with better cache coherence. A second theoretical contribution is a training algorithm for the thresholds used in the cascade and an associated high-confidence bound on the error rate of those thresholds — the number of desired detections that are missed because of the pruning of the intermediate DP tables.

At a practical level, this chapter describes a frame-rate implementation of a state-of-the-art object detector. The detector can easily be made to run at several frames per second on a multicore processor. This should open up new applications for this class of detectors in areas such as robotics and HCI. It should also facilitate future research by making richer models

computationally feasible. For example, the techniques described in this chapter should make it practical to extend the deformable model paradigm for object detection to include search over orientation or other pose parameters. We believe the performance of deformable model detectors can still be greatly improved.

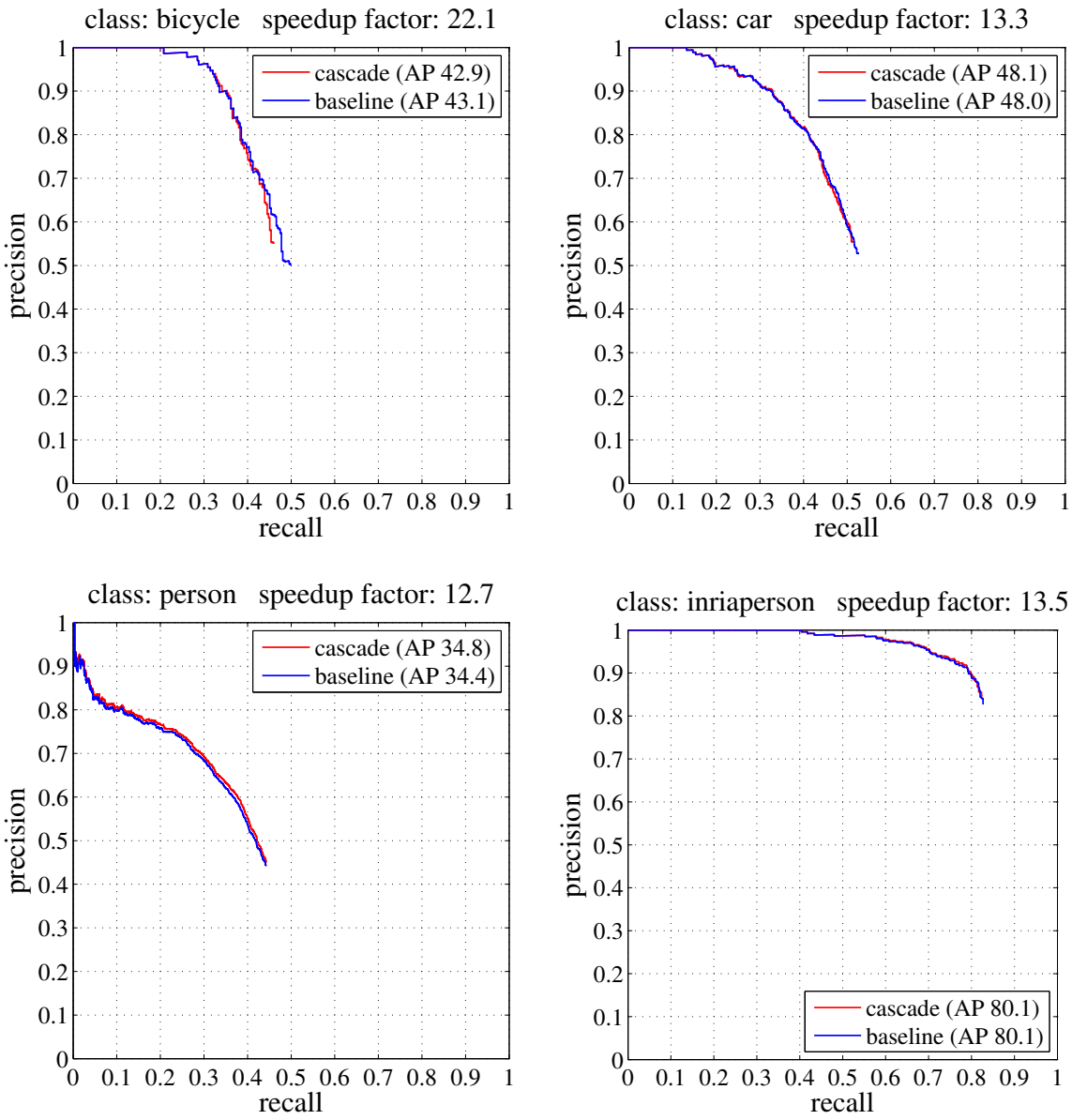


Figure 7.3: Sample precision-recall curves for bicycle, car, person, and INRIA person with the global threshold set to hit the precision-equals-recall point.

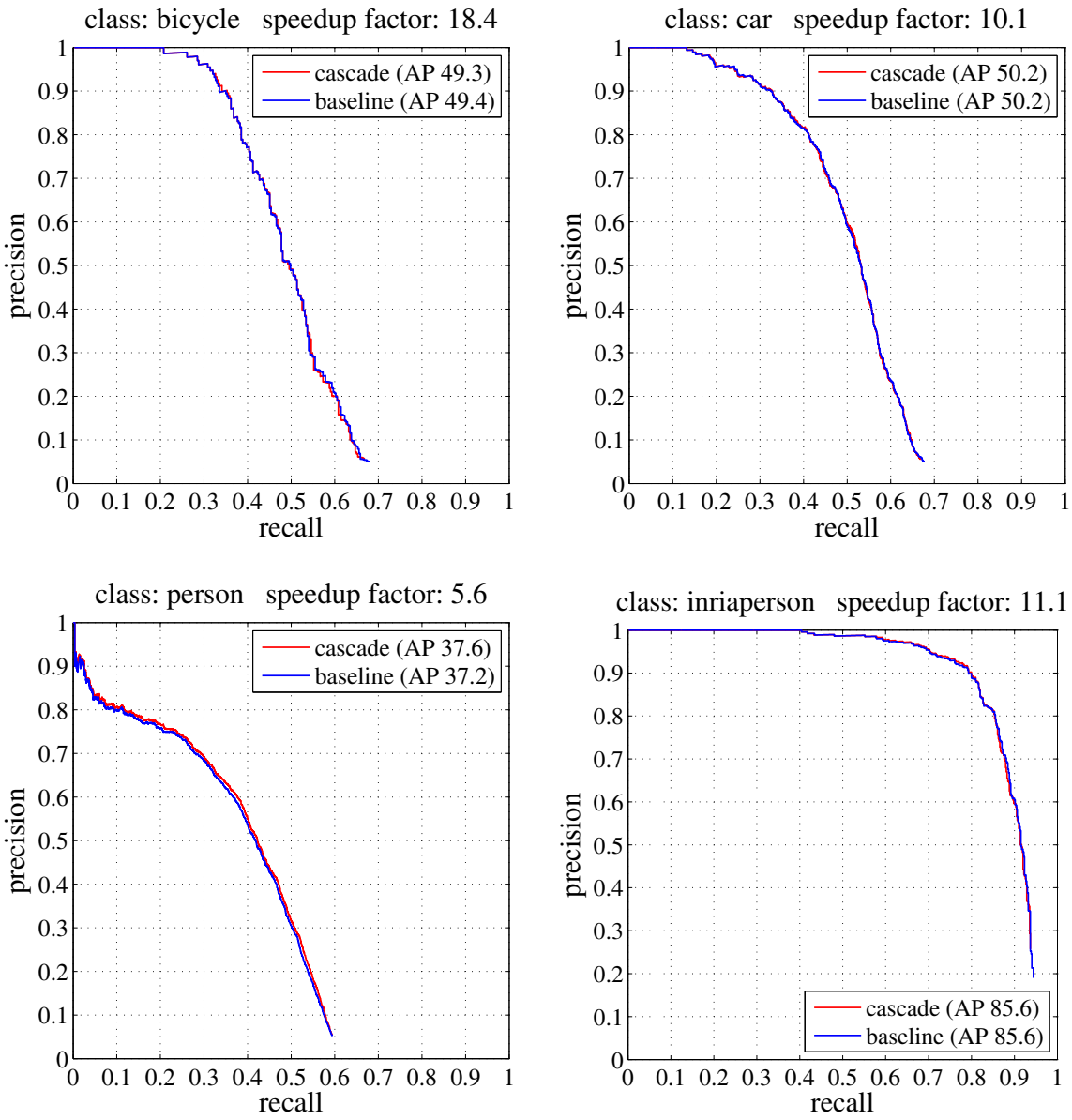


Figure 7.4: Sample precision-recall curves for bicycle, car, person, and INRIA person with the global threshold set to hit the precision = 0.05 level.

CHAPTER 8

CONCLUSION

This dissertation presents models and methods that have become the "go to" object detection system for the computer vision community. Numerous recent papers use our software as a detection subroutine in systems for high-level vision tasks such as scene classification [48], generation of textual descriptions of images [46], semantic segmentation [55, 9], and 3D scene understanding [40, 56]. Nearly all of the top-performing detection methods in the most recent PASCAL VOC Challenge [27] are extensions of this work that are either built directly on top of our software [78] or reimplement it [81, 68]. Despite this success, the state-of-the-art leaves vast room for improvement.

This work focuses on improving object detection performance by building richer models and developing new tools for learning their parameters. This path has not been easy. Successful learning of simple mixture models proved challenging and required developing careful techniques for avoiding instabilities in the learning process. Without these techniques, early attempts to learn models ended in bad local optima. Later, these insights made it possible to develop a high-performing grammar model for detecting people. However, we had specify the grammar's rules by hand.

There are large, open problems relating to grammar learning and to designing representations for objects and images. Clearly we do not want to hand specify a grammar for each object category. Instead, we should learn the structure of an object detection grammar from training data. Given the ability to cheaply acquire new training annotations, an attractive intermediate step is to learn grammar models from more detailed labels, rather than without any supervision.

Another major problem that will arise as we make more sophisticated models is that of score compatibility and linear grammars. In a linear grammar, the scores generated by a derivation that places a few filters may have a smaller dynamic range than the scores of alternative derivations that place large numbers of filters. Imagine detecting trains composed

of differing numbers of carriages. In order to make scores compatible, the model needs some form of nonlinearity for normalization. Neural grammars [33] are one possible approach, though early efforts indicate that parameter learning will be difficult.

We may be reaching the limits of current techniques for image representation. The types of models that we would ideally like to build are substantially different from those that we are able to build in practice. Ideally, a person detection grammar would model fine details of pose, clothing, and interactions with other objects. To model objects in more detail, we will likely need to augment, or entirely replace, the low-level features that we currently use. These features are essentially bag models that throw away too much information and represent the image at too coarse a resolution. The convergence in performance of our person detector and the poselets person detector [11] might be an indication that we have squeezed all of the available performance out of HOG features.

APPENDIX A

DETECTION POST-PROCESSING

A.1 Bounding box prediction

The desired output of an object detection system is not entirely clear. The goal in the PASCAL challenge is to predict the bounding boxes of objects. In [34], a bounding box was generated from the exact position and extent of the root filter in a detection. Yet detection with one of our models localizes each part filter in addition to the root filter. Furthermore, part filters are localized with greater spatial precision than root filters. It is clear that the root-based approach discards potentially valuable information gained from using a multiscale deformable part model.

In `voc-release4`, we use the complete configuration of an object hypothesis, z, to predict a bounding box for the object. This is implemented using functions that map a feature vector $g(z)$, to the upper-left, (x_1, y_1), and lower-right, (x_2, y_2), corners of the bounding box. For a model with N parts, $g(z)$ is a $2(N+1)$ dimensional vector containing the normalized offsets from the center of the original detection window (possibly clipped by the image boundary) to the center of the root filter and each of the N part filters in the image. Offsets are normalized by dividing their x component by the width of the original detection window and dividing their y component by the height.

Each object in the PASCAL training data is labeled by a bounding box. After training a model, we use the output of our detector on each instance to learn four linear functions for predicting *scale normalized offsets*, (dx_1, dy_1, dx_2, dy_2), from $g(z)$. This is done via linear least-squares regression, independently for each component of a mixture model. To predict a new bounding box from an existing detection window, the offsets are multiplied by the scale of the detection window and added to the window's corners to produce a new detection window.

Figure A.1 illustrates an example of bounding prediction for a car detection. This simple

Figure A.1: A car detection and the bounding box predicted from the object configuration.

method yields small but noticeable improvements in performance for some categories in the PASCAL datasets (see Chapter 6).

A.2 Integrating contextual information

We have implemented a simple procedure to rescore detections using contextual information.

Let $(D_1, \ldots, D_k)$ be a set of detections obtained using k different models (for different object categories) in an image I. Each detection $(B, s) \in D_i$ is defined by a bounding box $B = (x_1, y_1, x_2, y_2)$ and a score s. We define the context of I in terms of a k-dimensional vector $c(I) = (\sigma(s_1), \ldots, \sigma(s_k))$ where s_i is the score of the highest scoring detection in D_i, and $\sigma(x) = 1/(1 + \exp(-\frac{3}{2}x))$ is a logistic function for renormalizing the scores. If there are no detections for a particular category, we set the score for that category to the global detection threshold (-1.1).

To rescore a detection (B, s) in an image I we build a 25-dimensional feature vector with the original score of the detection, the top-left and bottom-right bounding box coordinates, and the image context,

$$g = (\sigma(s), x_1', y_1', x_2', y_2', c(I))^\intercal. \tag{A.1}$$

The coordinates $x_1' = x_1/w(I)$, $y_1' = y_1/h(I)$, $x_2' = x_2/w(I)$, $y_2' = y_2/h(I) \in [0, 1]$ are normalized by the width, $w(I)$, and height, $h(I)$, of the image. We use a category-specific classifier to score this vector to obtain a new score for the detection. The classifier is trained to distinguish correct detections from false positives by integrating contextual information

defined by g.

To get training data for the rescoring classifier we run our object detectors on images that are annotated with bounding boxes around the objects of interest (such as provided in the PASCAL datasets). Each detection returned by one of our models leads to an example g that is labeled as a true positive or false positive detection, depending on whether or not it significantly overlaps an object of the correct category.

This rescoring procedure leads to a noticeable improvement in the average precision on several categories in the PASCAL datasets (see Chapter 6). In our experiments we used the same dataset for training models and for training the rescoring classifiers. If held-out data were available for training, the sigmoid with fixed parameters could be replaced by a sigmoid with learned parameters, for example by using Platt's methods for calibrating SVM outputs [58, 49]. We used SVMs with cubic kernels for rescoring.

APPENDIX B

MARGIN-BOUND PRUNING

Consider training a linear SVM from examples $\{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)\}$. The primal objective function is

$$E(\mathbf{w}) = \frac{1}{2}\|\mathbf{w}\|^2 + C \sum_{n=1}^{N} \max(0, 1 - y_n(\mathbf{w} \cdot \mathbf{x}_n)). \tag{B.1}$$

Gradient-based methods, such as subgradient descent or (L)BFGS, produce a sequence of weight vectors $\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_t, \ldots$ by repeatedly computing $\nabla E(\mathbf{w})$ and then updating the current iterate according to a method-specific rule. Gradient computation involves taking a subgradient of each hinge loss term in the sum. The hinge-loss subgradient used in practice is

$$\begin{array}{ll} \vec{0} & \text{if } m_n(\mathbf{w}) = y_n(\mathbf{w} \cdot \mathbf{x}_n) > 1, \\ \mathbf{x}_n & \text{otherwise.} \end{array}$$

We call the quantity m_n the *margin*. Computing the margin requires taking the dot product $\mathbf{w} \cdot \mathbf{x}_n$, which is expensive if $\mathbf{w}$ is high dimensional.

Examples with $m_n(\mathbf{w}) > 1$ do not contribute to the gradient. Linear support vector machines tend to have sparse solutions — *i.e.*, most examples have $m_n(\mathbf{w}^*) > 1$ at the solution $\mathbf{w}^*$. We would like to avoid computing this dot product for many of the examples while the sequence $\{\mathbf{w}_t\}$ converges to $\mathbf{w}^*$.

Here we describe a technique that reduces the number of times this dot product is computed. In our experiments, we gain about a factor of two speedup. The key idea is: in iteration t, if an example $\mathbf{x}_n$ has margin $m_n(\mathbf{w}_t) > 1$, then we can bound how much $\mathbf{w}_t$ can change before we need to compute $\mathbf{x}_n$'s subgradient again. This method is most effective when the weight vector is changing slowly.

First we describe the technique for classical SVMs, and then we extend it to structural SVMs. The structural SVM formulation is easily applied to latent structural SVM and weak-label structural SVM.

B.1 The SVM case

Our goal is to design an efficient test that checks if we need to compute the subgradient for example $\mathbf{x}$ in iteration t. A negative answer must certify that $\mathbf{x}$ does not contribute to $\nabla E(\mathbf{w}_t)$, and can therefore be skipped.

Let $(\mathbf{x}, y)$ be a training pair and $\mathbf{w}$ a weight vector. First, we compute the margin $m(\mathbf{w}) = y(\mathbf{w} \cdot \mathbf{x})$. Now, suppose we change the weight vector to $\mathbf{w}' = \mathbf{w} + \Delta\mathbf{w}$. Based on our knowledge of $\mathbf{w}$ and $m(\mathbf{w})$, we wish to quickly check if $m(\mathbf{w}') = y(\mathbf{w}' \cdot \mathbf{x}) > 1$. If true, this condition would imply that $\mathbf{x}$ does not contribute to $\nabla E(\mathbf{w}')$, and can be skipped. For notational convenience, let $m = m(\mathbf{w})$ and $m' = m(\mathbf{w}')$. We can relate m' to the precomputed quantity m,

$$\begin{aligned} m' &= y(\mathbf{w}' \cdot \mathbf{x}_n) && \text{(B.2)} \\ &= y\left((\mathbf{w} + \Delta\mathbf{w}) \cdot \mathbf{x}\right) && \text{(B.3)} \\ &= y(\mathbf{w} \cdot \mathbf{x}) + y(\Delta\mathbf{w} \cdot \mathbf{x}) && \text{(B.4)} \\ &= m + y(\Delta\mathbf{w} \cdot \mathbf{x}) && \text{(B.5)} \\ &= m + \Delta m. && \text{(B.6)} \end{aligned}$$

We need to test if $m + \Delta m > 1$. We have already computed m, but computing Δm has the same cost as evaluating the dot product that we wish to avoid. Instead, we replace the exact value of Δm by the following lower bound.

$$\Delta m = y(\Delta\mathbf{w} \cdot \mathbf{x}) = y\|\Delta\mathbf{w}\|\|\mathbf{x}\| \cos\theta \geq -\|\Delta\mathbf{w}\|\|\mathbf{x}\| \tag{B.7}$$

We have that

$$m + \Delta m \geq m - \|\Delta\mathbf{w}\|\|\mathbf{x}\|, \tag{B.8}$$

and therefore

$$m - \|\Delta\mathbf{w}\|\|\mathbf{x}\| > 1 \implies m' > 1 \text{ and } \mathbf{x} \text{ does not contribute to } \nabla E(\mathbf{w}'). \tag{B.9}$$

To use this test in a gradient-based optimization algorithm, we record a history of the previous H weight vectors

$$\mathcal{W} = \{\mathbf{w}_{t-1}, \mathbf{w}_{t-2}, \ldots, \mathbf{w}_{t-H}\}.$$

We use $H = 50$ in the experiments below. For each example, we maintain a pair (h_n, m_n), where h_n is the index of the most recent historical weight vector $\mathbf{w}_{h_n} \in \mathcal{W}$ such that the margin $m_n = y_n(\mathbf{w}_{h_n} \cdot \mathbf{x}_n) > 1$. If no such weight vector exists (*i.e.*, the example's margin is ≤ 1 or h_n is too old and thus no longer in $\mathcal{W}$), then the example is flagged for subgradient computation.

To initialize the algorithm, we precompute $\|\mathbf{x}_n\|$ and flag all examples for subgradient computation. At the start of iteration t, we compute $\Delta\mathbf{w}_i = \mathbf{w}_t - \mathbf{w}_{t-i}$ and $\|\Delta\mathbf{w}_i\|$ for all $i \in [1, H]$. We expect $N \gg H$, and therefore this computation is amortized over the training set. Then, for each example $\mathbf{x}_n$:

1. If $\mathbf{x}_n$ is flagged for subgradient computation, compute its subgradient.
2. Else, compute the test value $t = m_n - \|\Delta\mathbf{w}_{h_n}\|\|\mathbf{x}_n\|$.
3. If $t \leq 1$, then compute $\mathbf{x}_n$'s subgradient. Otherwise, skip $\mathbf{x}_n$.
4. If $\mathbf{x}_n$'s subgradient was computed, update its pair to $(t, y_n(\mathbf{w}_t \cdot \mathbf{x}_n))$ if $y_n(\mathbf{w}_t \cdot \mathbf{x}_n) > 1$. Otherwise, flag $\mathbf{x}_n$ for subgradient computation in the next iteration.

This algorithm skips computing subgradients only when they are provably $\vec{0}$, and thus it computes the exact gradient $\nabla E(\mathbf{w}_t)$ in each iteration t.

B.2 The structural SVM case

In this section, we extend the margin-bound pruning technique to structural SVMs. With this extension it becomes straightforward to apply the technique to the slave problems that

arise when optimizing a latent structural SVM or weak-label structural SVM with the convex-concave procedure.

The structural SVM objective function is

$$\min_{\mathbf{w}} \frac{1}{2}\|\mathbf{w}\|^2 + C \sum_{n=1}^{N} \max_{y \in \mathcal{Y}} \left[\mathbf{w} \cdot \boldsymbol{\psi}(x_n, y) + L(y_n, y)\right] - \mathbf{w} \cdot \boldsymbol{\psi}(x_n, y_n). \tag{B.10}$$

Let

$$\hat{y}_n(\mathbf{w}) = \operatorname*{argmax}_{y \in \mathcal{Y}} \mathbf{w} \cdot \boldsymbol{\psi}(x_n, y) + L(y_n, y). \tag{B.11}$$

Then the subgradient of the structural hinge loss for example x_n is

$$\begin{cases} \vec{0} & \text{if } \hat{y}_n = y_n, \\ \boldsymbol{\psi}(x_n, \hat{y}_n(\mathbf{w})) - \boldsymbol{\psi}(x_n, y_n) & \text{otherwise.} \end{cases}$$

We define the margin for a training pair (x, y) as

$$m(\mathbf{w}) = \mathbf{w} \cdot \boldsymbol{\psi}(x, y) - \left[\mathbf{w} \cdot \boldsymbol{\psi}(x, \hat{y}'(\mathbf{w})) + L(y, \hat{y}')\right], \tag{B.12}$$

where

$$\hat{y}'(\mathbf{w}) = \operatorname*{argmax}_{\hat{y} \in \mathcal{Y} \setminus \{y\}} \mathbf{w} \cdot \boldsymbol{\psi}(x, \hat{y}) + L(y, \hat{y}), \tag{B.13}$$

is a loss adjusted prediction that *excludes the label* y. We assume that $L(y, y) = 0$. If the training label y were not excluded from the maximization in this definition, then $m(\mathbf{w}) = 0$ would be ambiguous: it could correspond to either $\hat{y}(\mathbf{w}) = y$ or $\hat{y}(\mathbf{w}) \neq y$. With this definition, $m(\mathbf{w}) > 0$ implies that the subgradient is $\vec{0}$.

As in the SVM case, for a training pair (x, y) and weight vector $\mathbf{w}$, we compute $m(\mathbf{w})$. Now, we bound how much an updated weight vector $\mathbf{w}' = \mathbf{w} + \Delta\mathbf{w}$ can change before we need to recompute the subgradient for example x.

For notational convenience, let $m = m(\mathbf{w})$ and $m' = m(\mathbf{w}')$.

$$m' = \mathbf{w}' \cdot \boldsymbol{\psi}(x, y) - \mathbf{w}' \cdot \boldsymbol{\psi}(x, \hat{y}'(\mathbf{w}')) - L(y, \hat{y}'(\mathbf{w}')) \tag{B.14}$$

$$= (\mathbf{w} + \Delta\mathbf{w}) \cdot \boldsymbol{\psi}(x, y) - (\mathbf{w} + \Delta\mathbf{w}) \cdot \boldsymbol{\psi}(x, \hat{y}'(\mathbf{w}')) - L(y, \hat{y}'(\mathbf{w}')) \tag{B.15}$$

$$= \mathbf{w} \cdot \boldsymbol{\psi}(x, y) - \left[\mathbf{w} \cdot \boldsymbol{\psi}(x, \hat{y}'(\mathbf{w}')) + L(y, \hat{y}'(\mathbf{w}'))\right] + \Delta\mathbf{w} \cdot \boldsymbol{\psi}(x, y) - \Delta\mathbf{w} \cdot \boldsymbol{\psi}(x, \hat{y}'(\mathbf{w}')) \tag{B.16}$$

$$\geq m + \Delta\mathbf{w} \cdot \boldsymbol{\psi}(x, y) - \Delta\mathbf{w} \cdot \boldsymbol{\psi}(x, \hat{y}'(\mathbf{w}')) \tag{B.17}$$

$$= m + \Delta m. \tag{B.18}$$

The inequality comes from the fact that we replaced the prediction $\hat{y}'(\mathbf{w}')$ with the prediction $\hat{y}'(\mathbf{w})$, which is the maximizer of $\max_{\hat{y} \in \mathcal{Y} \setminus \{y\}} \mathbf{w} \cdot \boldsymbol{\psi}(x, \hat{y}) + L(y, \hat{y})$ by definition.

Now, we lower bound Δm with

$$\Delta m = \Delta\mathbf{w} \cdot \boldsymbol{\psi}(x, y) - \Delta\mathbf{w} \cdot \boldsymbol{\psi}(x, \hat{y}'(\mathbf{w}')) \tag{B.19}$$

$$\geq -\|\Delta\mathbf{w}\| \|\boldsymbol{\psi}(x, y)\| - \Delta\mathbf{w} \cdot \boldsymbol{\psi}(x, \hat{y}'(\mathbf{w}')) \tag{B.20}$$

$$\geq -\|\Delta\mathbf{w}\| \|\boldsymbol{\psi}(x, y)\| - \|\Delta\mathbf{w}\| \|\boldsymbol{\psi}_{\max}(x)\|, \tag{B.21}$$

where

$$\boldsymbol{\psi}_{\max}(x) = \boldsymbol{\psi}(x, \operatorname*{argmax}_y \|\boldsymbol{\psi}(x, y)\|) \tag{B.22}$$

is the feature vector for example x with the maximum norm. The max-norm feature vector is used to avoid computing $\hat{y}'(\mathbf{w}')$ and leads to an upper bound on the second term in Eq. B.19. When using constraint generating methods (such as data mining), we replace $\boldsymbol{\psi}_{\max}$ with the longest constraint generated for example x thus far. Typically only a small number of constraints are generated per example, and the max-norm constraint can be tracked easily as constraints are generated.

Finally, the test

$$m - \|\Delta\mathbf{w}\| \left(\|\boldsymbol{\psi}(x, y)\| + \|\boldsymbol{\psi}_{\max}(x)\|\right) > 0 \tag{B.23}$$

implies that

$$m' \geq m + \Delta m > 0. \tag{B.24}$$

This test can be plugged into the algorithm presented in the previous section.

B.3 Experimental results

We tested the effect of margin-bound pruning by applying it the cache-restricted slave problems that arise from training an LSVM or a WL-SSVM (Section 3.1.2). To quantify the change in performance we trained a model and counted the number of subgradient computations that were pruned during the entire training procedure. Let P be the number of pruned computations, and let T be the total number of subgradient computations that would be required without pruning. We define the speedup as $T/(T-P)$: the total number of subgradients that would have been computed divided by the number that were actually computed.

In our implementation, we used a weight vector history of size $H = 50$ and the minConf LBFGS package [61]. Our speedup metric, which is based on treating each subgradient computation as a unit of work, should closely match the wall-clock speedup because the overhead of margin-bound pruning is small. The main overhead is the cost of computing $\Delta\mathbf{w}_i$ and $\|\Delta\mathbf{w}_i\|$ for $i \in [1, 50]$, once per LBFGS gradient request. This cost is amortized over the total number of feature vectors in the cache, which is typically around 20,000 to 70,000.

We measured the speedup while training *bicycle*, *car*, *horse*, and *sofa* models on the PASCAL 2007 dataset. The speedup factors were 2.24, 2.28, 1.96, and 1.94, respectively.

REFERENCES

[1] Y. Amit and D. Geman. A computational model for visual selection. *Neural Computation*, 11(7):1691–1715, 1999.

[2] Y. Amit and A. Kong. Graphical templates for model registration. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18(3):225–236, 1996.

[3] Y. Amit and A. Trouve. POP: Patchwork of parts models for object recognition. *International Journal of Computer Vision*, 75(2):267–282, 2007.

[4] S. Andrews, I. Tsochantaridis, and T. Hofmann. Support vector machines for multiple-instance learning. In *Advances in Neural Information Processing Systems*, 2003.

[5] E.J. Bernstein and Y. Amit. Part-based statistical models for object classification and detection. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2005.

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

[7] M. Blaschko and C. Lampert. Learning to localize objects with structured output regression. In *European Conference on Computer Vision*, pages 2–15. Springer-Verlag, 2008.

[8] M. Blaschko, A. Vedaldi, and A. Zisserman. Simultaneous object detection and ranking with weak supervision. In *Advances in Neural Information Processing Systems*, pages 235–243. 2010.

[9] X. Boix, J.M. Gonfaus, J. van de Weijer, A.D. Bagdanov, J. Serrat, and J. Gonzàlez. Harmony potentials. *International Journal of Computer Vision*, 96(1):83–102, 2012.

[10] L. Bourdev and J. Brandt. Robust object detection via soft cascade. In *IEEE Conference on Computer Vision and Pattern Recognition*, volume 2, pages 236–243. IEEE, 2005.

[11] L. Bourdev, S. Maji, T. Brox, and J. Malik. Detecting people using mutually consistent poselet activations. In *European Conference on Computer Vision*, pages 168–181. Spring-Verlag, 2010.

[12] M.C. Burl, M. Weber, and P. Perona. A probabilistic approach to object recognition using local photometry and global geometry. In *European Conference on Computer Vision*, 1998.

[13] R. Collobert, F. Sinz, J. Weston, and L. Bottou. Trading convexity for scalability. In *International Conference on Machine Learning*, pages 201–208. ACM, 2006.

[14] T.F. Cootes, G.J. Edwards, and C.J. Taylor. Active appearance models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(6):681–685, 2001.

[15] J. Coughlan, A. Yuille, C. English, and D. Snow. Efficient deformable template detection and localization without user initialization. *Computer Vision and Image Understanding*, 78(3):303–319, June 2000.

[16] D. Crandall, P. Felzenszwalb, and D. Huttenlocher. Spatial priors for part-based recognition using statistical models. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2005.

[17] N. Dalal. Inria person dataset. http://pascal.inrialpes.fr/data/human/index.html.

[18] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 886–893. IEEE, 2005.

[19] A.P. Dempster, N.M. Laird, and D.B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 1–38, 1977.

[20] C.B. Do, Q. Le, C.H. Teo, O. Chapelle, and A.J. Smola. Tighter bounds for structured estimation. In *Advances in Neural Information Processing Systems*, pages 281–288. 2008.

[21] M. Elad, Y. Hel-Or, and R. Keshet. Rejection based classifier for face detection. *Pattern Recognition Letters*, 23(12):1459–1471, 2002.

[22] M. Enzweiler, A. Eigenstetter, B. Schiele, and D. M. Gavrila. Multi-cue pedestrian classification with partial occlusion handling. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 990–997. IEEE, 2010.

[23] 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.

[24] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The PASCAL Visual Object Classes Challenge 2008 (VOC2008) Results. http://www.pascal-network.org/challenges/VOC/voc2008/workshop/index.html.

[25] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The PASCAL Visual Object Classes Challenge 2009 (VOC2009) Results. http://www.pascal-network.org/challenges/VOC/voc2009/workshop/index.html.

[26] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The PASCAL Visual Object Classes Challenge 2010 (VOC2010) Results. http://www.pascal-network.org/challenges/VOC/voc2010/workshop/index.html.

[27] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The PASCAL Visual Object Classes Challenge 2011 (VOC2011) Results. http://www.pascal-network.org/challenges/VOC/voc2011/workshop/index.html.

[28] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The pascal visual object classes (voc) challenge. *International Journal of Computer Vision*, 88(2):303–338, June 2010.

[29] P. Felzenszwalb and D. Huttenlocher. Distance transforms of sampled functions. Technical Report 2004-1963, Cornell University CIS, 2004.

[30] P. Felzenszwalb and D. Huttenlocher. Pictorial structures for object recognition. *International Journal of Computer Vision*, 61(1), 2005.

[31] P. F. Felzenszwalb, R. B. Girshick, and D. McAllester. Discriminatively trained deformable part models, release 4. http://people.cs.uchicago.edu/~pff/latent-release4/.

[32] P. F. Felzenszwalb, R. B. Girshick, D. McAllester, and D. Ramanan. Object detection with discriminatively trained part based models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(9):1627–1645, 2010.

[33] P. F. Felzenszwalb and D. McAllester. Object detection grammars. Technical Report 2010-02, Univerity of Chicago, CS Dept., 2010.

[34] P. F. Felzenszwalb, D. McAllester, and D. Ramanan. A discriminatively trained, multiscale, deformable part model. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8. IEEE, 2008.

[35] R. Fergus, P. Perona, and A. Zisserman. Object class recognition by unsupervised scale-invariant learning. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2003.

[36] R. Fergus, P. Perona, and A. Zisserman. A sparse object category model for efficient learning and exhaustive recognition. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2005.

[37] M.A. Fischler and R.A. Elschlager. The representation and matching of pictorial structures. *IEEE Transactions on Computer*, 22(1), 1973.

[38] F. Fleuret and D. Geman. Coarse-to-fine face detection. *International Journal of Computer Vision*, 41(1):85–107, 2001.

[39] S. Gangaputra and D. Geman. A design principle for coarse-to-fine classification. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 1877–1884. IEEE, 2006.

[40] A. Geiger, M. Lauer, and R. Urtasun. A generative model for 3d urban scene understanding from movable platforms. In *IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, June 2011.

[41] U. Grenander, Y. Chow, and D.M. Keenan. *HANDS: A Pattern-Theoretic Study of Biological Shapes*. Springer-Verlag, 1991.

[42] A. Gunawardana, M. Mahajan, A. Acero, and J.C. Platt. Hidden conditional random fields for phone classification. In *Ninth European Conference on Speech Communication and Technology*, 2005.

[43] Y. Jin and S. Geman. Context and hierarchy in a probabilistic image model. In *IEEE Conference on Computer Vision and Pattern Recognition*, volume 2, pages 2145–2152. IEEE, 2006.

[44] Y. Ke and R. Sukthankar. PCA-SIFT: A more distinctive representation for local image descriptors. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2004.

[45] M. Kearns and U. Vazirani. *An Introduction to Computational Learning Theory*. MIT Press, 1994.

[46] G. Kulkarni, V. Premraj, S. Dhar, S. Li, Y. Choi, A.C. Berg, and T.L. Berg. Baby talk: Understanding and generating simple image descriptions. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 1601–1608. IEEE, 2011.

[47] B. Leibe, A. Leonardis, and B. Schiele. Robust object detection with interleaved categorization and segmentation. *International Journal of Computer Vision*, 77(1):259–289, 2008.

[48] L.J. Li, H. Su, E.P. Xing, and L. Fei-Fei. Object bank: A high-level image representation for scene classification and semantic feature sparsification. In *Advances in Neural Information Processing Systems*. MIT Press, 2010.

[49] H.T. Lin, C.J. Lin, and R.C. Weng. A note on plattâĂŹs probabilistic outputs for support vector machines. *Machine learning*, 68(3):267–276, 2007.

[50] D.C. Liu and J. Nocedal. On the limited memory bfgs method for large scale optimization. *Mathematical programming*, 45(1):503–528, 1989.

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

[52] D. McAllester and J. Keshet. Generalization bounds and consistency for latent structural probit and ramp loss. In *Advances in Neural Information Processing Systems 24*, pages 2205–2212. 2011.

[53] Y. Ohta, T. Kanade, and T. Sakai. An analysis system for scenes containing objects with substructures. In *International Joint Conference on Pattern Recognition*, pages 752–754, 1978.

[54] C.P. Papageorgiou, M. Oren, and T. Poggio. A general framework for object detection. In *IEEE International Conference on Computer Vision*, 1998.

[55] O.M. Parkhi, A. Vedaldi, CV Jawahar, and A. Zisserman. The truth about cats and dogs. In *IEEE International Conference on Computer Vision*, pages 1427–1434. IEEE, 2011.

[56] N. Payet and S. Todorovic. Scene shape from texture of objects. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 2017–2024. IEEE, 2011.

[57] W.H. Plantinga and C.R. Dyer. An algorithm for constructing the aspect graph. In *Foundations of Computer Science, 1985., 27th Annual Symposium on*, pages 123–131, 1986.

[58] J. C. Platt. Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods. In *Advances in Large Margin Classifiers*, pages 61–74. MIT Press, 1999.

[59] A. Quattoni, S. Wang, L.P. Morency, M. Collins, and T.J. Darrell. Hidden conditional random fields. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(10):1848–1852, October 2007.

[60] N.D. Ratliff, J.A. Bagnell, and M.A. Zinkevich. (online) subgradient methods for structured prediction. In *International Conference on Artificial Intelligence and Statistics*, 2007.

[61] M. Schmidt. minconf - projection methods for optimization with simple constraints in matlab. http://www.di.ens.fr/~mschmidt/Software/minConf.html, Retrieved Jan. 23, 2012.

[62] H. Schneiderman and T. Kanade. A statistical method for 3d object detection applied to faces and cars. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2000.

[63] S. Shalev-Shwartz, Y. Singer, and N. Srebro. Pegasos: Primal estimated sub-gradient solver for SVM. In *International Conference on Machine Learning*, 2007.

[64] J. Šochman and J. Matas. Waldboost-learning for time constrained sequential detection. In *IEEE Conference on Computer Vision and Pattern Recognition*, volume 2, pages 150–156. IEEE, 2005.

[65] B. Taskar, C. Guestrin, and D. Koller. Max-margin markov networks. In *Advances in Neural Information Processing Systems*. MIT Press, Cambridge, MA, 2003.

[66] D. Tran and D. Forsyth. Improved human parsing with a full relational model. In *European Conference on Computer Vision*, pages 227–240. Springer-Verlag, 2010.

[67] I. Tsochantaridis, T. Joachims, T. Hofmann, and Y. Altun. Large margin methods for structured and interdependent output variables. *Journal of Machine Learning Research*, 6(2):1453–1484, 2006.

[68] A. Vedaldi and A. Zisserman. Oxford DPM MK. http://pascallin.ecs.soton.ac.uk/challenges/VOC/voc2011/workshop/index.html.

[69] A. Vedaldi and A. Zisserman. Structured output regression for detection with partial truncation. In *Advances in Neural Information Processing Systems*, pages 1928–1936. 2009.

[70] P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. In *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, volume 1, pages 511–518. IEEE, 2001.

[71] P.A. Viola and M.J. Jones. Robust real-time face detection. *International Journal of Computer Vision*, 57(2):137–154, May 2004.

[72] X. Wang, T.X. Han, and S. Yan. An hog-lbp human detector with partial occlusion handling. In *IEEE International Conference on Computer Vision*, pages 32–39. IEEE, 2009.

[73] Y. Wang, D. Tran, and Z. Liao. Learning hierarchical poselets for human parsing. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 1705–1712. IEEE, 2011.

[74] M. Weber, M. Welling, and P. Perona. Towards automatic discovery of object categories. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2000.

[75] C.N.J. Yu and T. Joachims. Learning structural svms with latent variables. In *International Conference on Machine Learning*, pages 1169–1176. ACM, 2009.

[76] A.L. Yuille, P.W. Hallinan, and D.S. Cohen. Feature extraction from faces using deformable templates. *International Journal of Computer Vision*, 8(2):99–111, 1992.

[77] A.L. Yuille and A. Rangarajan. The concave-convex procedure. *Neural Computation*, 15(4):915–936, 2003.

[78] J. Zhang, Y. Yu, Y. Huang, C. Wang, W. Ren, J. Wu, K. Huang, and T. Tan. Object detection based on data decomposition, spatial mixture modeling and context. http://pascallin.ecs.soton.ac.uk/challenges/VOC/voc2011/workshop/nlpr.pdf.

[79] L.L. Zhu, Y. Chen, A. Torralba, W. Freeman, and A. Yuille. Part and appearance sharing: Recursive compositional models for multi-view multi-object detection. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 1919–1926. IEEE, 2010.

[80] L.L. Zhu, Y. Chen, and A. Yuille. Unsupervised learning of probabilistic grammar-markov models for object categories. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(1):114–128, 2009.

[81] L.L. Zhu, Y. Chen, A. Yuille, and W. Freeman. Latent hierarchical structural learning for object detection. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 1062–1069. IEEE, 2010.

[82] S.C. Zhu and D. Mumford. A stochastic grammar of images. *Foundations and Trends in Computer Graphics and Vision*, 2(4):259–362, 2007.