

UC San Diego

UC San Diego Electronic Theses and Dissertations

Title

Interactive learning and prediction algorithms for computer vision applications

Permalink

<https://escholarship.org/uc/item/5g98q58h>

Author

Branson, Steven

Publication Date

2012

Peer reviewed|Thesis/dissertation

UNIVERSITY OF CALIFORNIA, SAN DIEGO

**Interactive Learning and Prediction Algorithms For Computer Vision
Applications**

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

in

Computer Science

by

Steven Branson

Committee in charge:

Professor Serge Belongie, Chair
Professor Sanjoy Dasgupta
Professor Gert Lanckriet
Professor Lawrence Saul
Professor Nuno Vasconcelos

2013

Copyright
Steven Branson, 2013
All rights reserved.

The dissertation of Steven Branson is approved, and it is acceptable in quality and form for publication on microfilm and electronically:

Chair

University of California, San Diego

2013

DEDICATION

To my parents and sister.

TABLE OF CONTENTS

Signature Page	iii
Dedication	iv
Table of Contents	v
List of Figures	viii
Acknowledgements	xvii
Vita	xix
Abstract of the Dissertation	xx
Chapter 1	Introduction	1
	1.1 Overview and Contributions	2
	1.2 Preliminaries	4
	1.2.1 Structured Prediction	5
	1.2.2 Structured Learning	5
	1.2.3 Interactive Labeling	7
	1.2.4 Active Learning	7
	1.2.5 Active Prediction	8
Chapter 2	Visipedia	9
	2.1 The Birds-200 Dataset	13
	2.1.1 Motivation For a Bird Dataset	14
	2.1.2 Dataset Specification and Collection	15
	2.1.3 Benchmarks and Baseline Experiments	15
Chapter 3	Interactive Computer Vision	23
	3.1 Visual Recognition With Humans in the Loop	24
	3.1.1 Introduction	25
	3.1.2 Related Work	27
	3.1.3 Methods and Algorithm Description	29
	3.1.4 Extension to Part-Based Models	32
	3.1.5 Datasets and Implementation Details	41
	3.1.6 Experiments	44
	3.2 An Interactive Part Labeling Tool	52
	3.2.1 Model and Notation	52
	3.2.2 Incorporating User Input	52
	3.2.3 Creating an Interactive User Interface	53

Chapter 4	Interactive Learning Algorithms	58
	4.1 Introduction	58
	4.2 Online Structured Learning	62
	4.3 Experiments	66
	4.4 Conclusion	69
Chapter 5	Methodologies For Diagnosing Errors in Learned Computer Vi- sion Systems	70
	5.1 Introduction	70
	5.2 Algorithm and Main Framework	73
	5.2.1 Optimization Algorithm	73
	5.2.2 Diagnostic Statistics	74
	5.2.3 Theoretical Guarantees	76
	5.2.4 Non-Traditional Online Updates	77
	5.3 Proof of Thm 5.2.1	80
	5.4 Online Dual Update Step	84
Chapter 6	Faster Customized Optimization Algorithms For Computer Vision Learning Problems	88
	6.1 Introduction	89
	6.2 Background and Related Work	91
	6.3 Notation and Algorithm Summary	94
	6.4 Optimization Algorithms	96
	6.5 Notation	98
	6.6 Multi-Sample Update	98
	6.7 Applications	101
	6.7.1 Cost-Sensitive Multiclass SVMs	102
	6.7.2 Sliding Window Object Detection	102
	6.7.3 Deformable Part Model Based Detection	103
	6.8 Experiments	103
	6.8.1 Multiclass SVM	103
	6.8.2 Part-Based Detection	104
	6.8.3 Analysis of Results	106
	6.9 Conclusion	107
Chapter 7	Example Applications	108
	7.1 Cost-Sensitive Multiclass SVMs:	109
	7.2 Alignment Models	110
	7.2.1 Sliding Window Object Detection	111
	7.2.2 Deformable Part Models	112
	7.2.3 Pose Mixture Models and Occlusion Reasoning	115
	7.2.4 Annotation Interfaces For Part-Based Models	117
	7.3 Model Sharing Methods	121

	7.3.1	Joint Learning of Attribute Classifiers	121
	7.3.2	Part Sharing Models	123
	7.3.3	Part-Attribute Sharing Models	124
	7.3.4	Approximate Inference Algorithms and Extensions .	126
	7.4	Tracking	126
	7.5	Segmentation	128
	7.6	Behavior and Action Recognition	130
	7.6.1	Behavior Bout features	131
	7.6.2	Scoring and Predicting Behaviors	132
	7.6.3	Loss Functions For Behavior Recognition	133
	7.7	Features and Implementation Details	134
	7.7.1	Template Features	135
	7.7.2	Bag-of-Words Features	135
	7.7.3	Fisher Vector Features	136
Chapter 8		Synthetic Distributions For Which Popular Machine Learning Al- gorithms Perform Poorly	139
	8.1	Synthetic Distribution Definition	140
	8.2	No Linear Classifier Is Better Than Random Chance	142
	8.3	Nearest Neighbor Requires Exponential Data	143
	8.4	Bag-of-Words Scales Badly With Number of Classes	144
Chapter 9		Conclusion	149
	9.1	Summary of Contributions	149
	9.2	Future Work	150
Bibliography		151

LIST OF FIGURES

Figure 2.1:	Applications of Visipedia: a) A user takes a picture of a bird, which the system recognizes the object and uses to bring up the corresponding Wikipedia article. This is enabled by the recognition algorithms described in Section 7.3.3 and Section 3.1. b) Interactive part diagrams can be embedded into Wikipedia pages. This is the output of our part annotation tool described in Section 7.2.4, which can be created in an automated or semi-automated fashion (Section 3.2). c) The user clicks on an unknown structure on the beak of a pigeon, and the system helps to identify it as a cere. This is enabled by the localization algorithms described in Section 7.2.3.	10
Figure 2.2:	Applications of Visipedia: a) Classes can be annotated with semantic attributes, which can be used to render HTML using customizable templates, or to browse and search through classes by attributes. This was created using existing mediawiki software, and populated with our data from the Birds-200 dataset. b) Images that are labeled with part and attribute annotations enable dynamic image galleries or image search by attribute. c) Users represent visual concepts using exemplars or diagrams.	11
Figure 2.3:	Overview of the Architecture of Visipedia: An overview of the approach this dissertation takes toward building Visipedia. A visual language that both humans and computers can understand is established; we use a certain definition of parts (Section 7.2.3) and attributes (Section 7.3.3). Images can be annotated in terms of this language either by humans using the annotation tools shown in Fig 2.6-2.8, in automated fashion using learned computer vision algorithms (Section 7.2.3,7.3.3), or in semi-automated fashion using some combination of the two (Chapter 3). The semantically annotated images can be used to provide a number of web-based features/services that can be used to enrich Wikipedia/Visipedia articles (Fig 2.1-2.2). The semantically annotated images can also be used to train machine learning algorithms that can be used to improve the efficiency of automated prediction algorithms (Chapters 4-6). The vocabulary of parts and attributes is defined by experts/Wikipedians using tools such as those depicted in Fig 7.4 and Fig 2.2(a).	12
Figure 2.4:	CUB-200-2011 Example Images , showing a few example images from 20 different classes (the full dataset contains 200 classes)	17

Figure 2.5:	Collected Parts and Attributes. (a) The 15 part location labels collected for each image. (b) The 28 attribute-groupings that were collected for each image, and the associated part for localized attribute detectors.	18
Figure 2.6:	MTurk GUI for collecting part location labels, deployed on 11,788 images for 15 different parts and 5 workers per image.	18
Figure 2.7:	MTurk GUI for collecting bounding box labels, deployed on 11,788 images.	19
Figure 2.8:	MTurk GUI for collecting attribute labels, deployed on 11,788 images for 28 different questions and 312 binary attributes.	19
Figure 2.9:	Dataset Statistics. (a) Distribution of the number of images per class (most classes have 60 images). (b) Distribution of the size of each image in pixels (most images are roughly 500X500). (c) Distribution of the ratio of the area of the bird’s bounding box to the area of the entire image. (d) The average amount of time it took MTurkers to label each part.	20
Figure 2.10:	Categorization Results for 200-way bird species classification. The top 2 images show confusion matrices when using a universal bird detector to detect the most likely location of all parts and then evaluating a multiclass classifier. The bottom 2 images show confusion matrices when evaluating a multiclass classifier on the ground truth part locations. The 2 images on the left show results with 5 training images per class, and the images on the right show results with 52 training images per class.	21
Figure 2.11:	Example Part Detection Results, with good detection results on the left and bad detection results on the right. A loss of 1.0 indicates that the predicted part locations are about as good as the average MTurk labeler.	22
Figure 3.1:	Screen Capture of an iPhone App for Bird Species Recognition: A user takes a picture of a bird she wants to recognize, and it is uploaded to a server. The server runs computer vision algorithms to localize the different parts of the bird and predict the bird species (debug output of the algorithms is shown in the image on the right). The computer system intelligently selects a series of questions to ask (<i>click on the head, what is the primary color of the bird?</i>) that are designed to reduce its ambiguity about the predicted bird species as quickly as possible	24

Figure 3.2:	Examples of classification problems that are easy or hard for humans. While basic-level category recognition (left) and recognition of low-level visual attributes (right) are easy for humans, most people struggle with finer-grained categories (middle). By defining categories in terms of low-level visual properties, hard classification problems can be turned into a sequence of easy ones.	25
Figure 3.3:	Examples of the visual 20 questions game on the 200 class Bird dataset. Human responses (shown in red) to questions posed by the computer (shown in blue) are used to drive up recognition accuracy. In the left image, computer vision algorithms can guess the bird species correctly without any user interaction. In the middle image, computer vision reduces the number of questions to 2. In the right image, computer vision provides little help.	27
Figure 3.4:	Visualization of the basic algorithm flow. The system poses questions to the user, which along with computer vision, incrementally refine the probability distribution over classes.	28
Figure 3.5:	Examples of user responses for each of the 25 attributes. The distribution over $\{Guessing, Probably, Definitely\}$ is color coded with blue denoting 0% and red denoting 100% of the five answers per image attribute pair.	31
Figure 3.6:	Extension to Part-Based Methods: Our system can query the user for input in the form of binary attribute questions or part clicks. In this illustrative example, the system provides an estimate for the pose and part locations of the object at each stage. Given a user-clicked location of a part, the probability distributions for locations of the other parts in each pose will adjust accordingly. The right-most column depicts the maximum likelihood estimate for part locations.	32
Figure 3.7:	Probabilistic Model. 3.7(a): The spatial relationship between parts has a hierarchical independence structure. 3.7(b): Our model employs attribute estimators, where part variables θ_p are connected using the hierarchical model shown in 3.7(a).	35
Figure 3.8:	Different Models of User Responses: <i>Left:</i> Classification performance on Birds-200 (Method 1) without computer vision. Performance rises quickly (blue curve) if users respond deterministically according to whatbird.com attributes. MTurk users respond quite differently, resulting in low performance (green curve). A learned model of MTurk responses is much more robust (red curve). <i>Right:</i> A test image where users answer several questions incorrectly and our model still classifies the image correctly.	45

Figure 3.9:	Performance on Birds-200 when using computer vision: Left Plot: comparison of classification accuracy (Method 1) with and without computer vision when using MTurk user responses. Two different computer vision algorithms are shown, one based on per-class 1-vs-all classifiers and another based on attribute classifiers. Right plot: the number of questions needed to identify the true class (Method 2) drops from 11.11 to 6.43 on average when incorporating computer vision.	46
Figure 3.10:	Examples where computer vision and user responses work together: <i>Left:</i> An image that is only classified correctly when computer vision is incorporated. Additionally, the computer vision based method selects the question <code>HasThroatColorWhite</code> , a different and more relevant question than when vision is not used. In the right image, the user response to <code>HasCrownColorBlack</code> helps correct computer vision when its initial prediction is wrong.	47
Figure 3.11:	Images that are misclassified by our system: <i>Left:</i> The Parakeet Auklet image is misclassified due to a cropped image, which causes an incorrect answer to the belly pattern question (the Parakeet Auklet has a plain, white belly, see Fig. 3.3). <i>Right:</i> The Sayornis and Gray Kingbird are commonly confused due to visual similarity. . .	48
Figure 3.12:	Performance on Animals With Attributes: Left Plot: Classification performance (Method 1), simulating user responses using soft class-attributes (see [40]). Right Plot: The required number of questions needed to identify the true class (Method 2) drops from 5.94 to 4.11 on average when incorporating computer vision.	49
Figure 3.13:	Classification accuracy as a function of time when 3.13(a) maximizing expected information gain; and 3.13(b) minimizing amount of human labor, measured in time. Performance is measured as the average number of seconds to correctly classify an image (described in Section 3.1.6).	50

Figure 3.14:	Four examples of the behavior of our system. 3.14(a): The system estimates the bird pose incorrectly but is able to localize the head and upper body region well, and the initial class prediction captures the color of the localized parts. The user’s response to the first system-selected part click question helps correct computer vision. 3.14(b): The bird is incorrectly detected, as shown in the probability maps displaying the likelihood of individual part locations for a subset of the possible poses (not visible to the user). The system selects “ <i>Click on the beak</i> ” as the first question to the user. After the user’s click, the other part location probabilities are updated and exhibit a shift towards improved localization and pose estimation. 3.14(c): Certain infrequent poses (<i>e.g.</i> frontal views) were not discovered by the initial off-line clustering (see Figure 7.5). The initial probability distributions of part locations over the image demonstrate the uncertainty in fitting the pose models. The system tends to fail on these unfamiliar poses. 3.14(d): The system will at times select both part click and binary questions to correctly classify images.	51
Figure 3.15:	Visualization of Interactive Part Labeling Tool: The system displays in realtime the maximum likelihood location of all parts as the user drags different parts. The machine predicted bird location is initially entirely incorrect, but improves significantly after the user drags the location of the head. All 15 parts are correct after the user drags the head, crown, and back.	53
Figure 3.16:	Visualization of Model and Algorithms: a) Object part location variables are assumed to have a hierarchical relationship. Dynamic programming is run in both directions up and down the tree. b-d) Visualization of the main algorithms used for the interactive interface. In each algorithm, part nodes are traversed in the order indicated by the arrows. When processing each node, solutions over the sub-graphs highlighted in blue are combined to form the solution highlighted in red. b) Using a standard dynamic programming algorithm, information is propagated from the child nodes up to the root using Eq 3.28-3.29, c) In a second top-down algorithm, information is passed from the root back down to the children using Eq 3.30-3.32, d) When a user response updates the variable θ_p , information is propagated using a breadth first traversal of the tree beginning at p using Eq 3.34-3.37.	55

Figure 4.1:	Interactive Labeling and Online Learning of Part Models: A part model is trained in online fashion, where annotation becomes increasingly automated as more images are labeled. The diagram shows how the interactive labeling interface changes on a particular test image as the size of the training set increases, with green lines representing parts that were dragged by the user.	59
Figure 4.2:	Typical Results on Birds-200, with blue dots denoting parts predicted by a deformable part model trained on a 1000 image training set, and red dots denoting parts that were corrected by a simulated user (as described in Section 4.3)	65
Figure 4.3:	Results on Birds-200: (a) Average part prediction accuracy as a function of the number of annotated parts per image. Each curve shows performance for a different training set size, indicating annotation becomes progressively more automated as more images are labeled. The legend shows the average number of parts that needed to be labeled until all 13 were correct. An upward curving plot indicates interactive labeling is effective. (b) Part prediction accuracy as a function of labeling time per image.	68
Figure 5.1:	Proposed framework for annotation, training, and diagnosing different types of errors. Items in gray are executed by a computer, while items in pink are executed by a human. Items drawn using diamonds indicate a test or diagnosis technique. Tests to determine whether or not the model or feature space has saturated, sufficient training examples have been labeled, or sufficient computation time has been spent are automated by the computer system and can be communicated to the researcher/annotator via dynamically generated plots or figures. A test to determine if an example has been mislabeled is semi-automated by the computer system: it proposes examples that are likely to be mislabeled, which in turn must be verified by a human annotator. All ensuing changes such as newly labeled training examples, manually corrected labels, and changes to the model or feature space are implemented using online updates to the learned model without re-training the system.	71
Figure 6.1:	We introduce efficient optimization that make structured learning computationally tractable to very large datasets, significantly reducing train time for deformable part models, object detection, and cost-sensitive multiclass learning	89

Figure 6.2:	Three approaches for training a bird detector: Note that the example image is purposefully chosen as it contains two distinct sets of negative patches; easy from the background and hard from the ‘bird of paradise’ plant. (Left) Training a binary classifier requires a random sampling of negatives, which might be un-informative. (Right) A traditional structured SVM finds, in each iteration, the hardest sample to update the model, which is wasteful since multiple patches contain discriminative information. (Middle) The proposed method, SVM - Importance Sampling, incorporates domain knowledge in the importance sampling routine and inexpensively updates the model with respect to a set of samples in each sequential iteration.	91
Figure 6.3:	Results on ImageNet Rand 200: ImageNet has a strong hierarchy as illustrated in fig. 6.3(a), which motivates the hierarchical loss function – clearly, the missclassification between the gorilla subspecies is less severe than that of horse - giraffe. Figure 6.3(b) show that the structural SVM formulations better optimize this loss, even after post processing as described in sec. 6.8.1 is performed on the non - structural SVMs. Note also that the proposed method, SVM-IS, converges significantly faster than SVM^{struct}	105
Figure 6.4:	Results on CUB-200-2011: (a) Typical part-detection results on CUB-200-2011, with failure cases in the right column. Note that our detector works across extreme variations in pose. (b) Our method SVM-IS converges significantly faster than all other methods. Batch algorithms (SVM^{struct} and mining hard negatives) make little progress in the allotted time (the entire plot shown represents only 25 passes through the training set). Our method yields roughly 50 times speedup over the already fast SGD solver [8]. Timing results are on a single core 2.7GHz Intel Xeon CPU.	106
Figure 7.1:	Part-Based Model: a) Spatial relationships between different parts is represented by a tree-structured graph (only a subset of this tree is shown for visualization purposes). b) Annotated part locations in a particular image. c) A step-by-by GUI for collecting part annotations where a user annotates only one part at a time via a simple mouse click.	112

Figure 7.2:	Components of a Part Based Model: a) Each part in an image X is represent by a location $\theta_p = \{x_p, y_p, s_p, r_p, v_p\}$ (x-y location, scale, rotation, pose). b) Appearance features (in this case HOG features) are extracted with respect to a part location and stored into a vector $\varphi_p(\theta_p; X)$. c) A sliding window part detector computes a detection score $\psi_p(\theta_p; X)$ for every possible value of θ_p . d) Spatial features $\lambda_{pq}(\theta_p, \theta_q)$ occur over every parent-child part, with a quadratic cost penalizing changes in location, scale, and orientation. Each possible pair of poses for child and parent parts has a different spring cost and different pose transition score	116
Figure 7.3:	Hyper-Supervised Part Annotation Given a Predefined Bird Model: In contrast to the GUI in Figure 7.1, the user can label the scale, orientation, and pose of each part, in addition to the pixel location. Annotation occurs as a course-to-fine step-by-step procedure. The model can be used to align birds that vary significantly in shape and pose.	118
Figure 7.4:	GUI For Building Custom Part Models: A web-based GUI for defining a custom part model for new categories. Models created with this tool can be used to annotate images as in Figure 7.3 a) The user can add new parts using the tree control on the right. b) When annotating a new image, the user can select a pose of a part from a set of previously defined poses. If no previously defined pose matches the current image, the user can creates a new pose. c) When creating a new pose (in this case a frontal view of the breast), the user can draw a custom vector graphics visualization of that pose. d) When annotating a different image, a user can select the same frontal view pose of the breast, adjusting the location of 2 control points to rotate/scale/translate the pose visualization.	119
Figure 7.5:	Pose Clusters For the Part <i>Bird Body</i>: Images in our dataset are clustered by k -means on the spatial offsets of part locations from parent part locations. Semantic labels of clusters were manually assigned by visual inspection. Left/right orientation is in reference to the image.	120
Figure 7.6:	Graphical Models For Alignment and Model Sharing: a) Simple multiclass classification methods train independent classifiers for each of C classes, and featurize an entire image without modeling object location. b) Attribute-based methods employ a layer of M attribute detectors, which are functions of low-level image features and shared among classes. c) Localized models employ class detectors that can be complex functions of the object's location in the image. d) Part-attribute methods can employ both part and attribute detectors that are shared among classes.	122

Figure 7.7:	Overview of Behavior Detection in Flies: In this example, 50 flies are tracked in a circular arena. Features are computed as statistics or transformations of tracked trajectories in candidate time-localized behavior bouts. The system predicts a segmentation of a tracked trajectory into the set of predicted behaviors with the highest score.	131
Figure 7.8:	Feature Visualization: Three types of features are extracted from the bird image on the left, localized around the depicted part locations. From left to right: 1) 7×7 HOG templates, 2) color histograms, 3) bag-of-words encoded SIFT features.	134
Figure 8.1:	Synthetic Distribution For Which Linear Classifiers and Nearest Neighbor Do Poorly: Images are generated using the procedure described in Section 8.1, with $C = 3$ classes, $n = 4$ examples per class, $P = 3$ parts, and $k = 2$ attributes per part. Each row shows 4 images from each of the 3 classes, and classes have class-attribute vectors $\mathbf{a}^{c_1} = 001001$, $\mathbf{a}^{c_2} = 010101$, $\mathbf{a}^{c_3} = 010011$	141
Figure 8.2:	Object recognition methods scale poorly with the number of classes. Is this partially a consequence of an over-reliance on bag-of-words? a) An upper bound on the classification accuracy of bag-of-words methods for our synthetic distribution, with different parameter settings for the number of parts P and number of attributes $M = Pk$ (see Eq 8.23), predicts poor performance of methods based on bag-of-words as the number of classes increases. b-d) Plots taken from [31, 18, 9] show that multiclass classification accuracy for state-of-the-art methods scales badly with the number of classes on Caltech-256, ImageNet, and Birds-200.	145

ACKNOWLEDGEMENTS

First and foremost, I was truly lucky to have Serge Belongie as my research advisor. He is an exceptionally well grounded person who always puts the well-being of his students above himself. His research vision that spans into areas beyond computer vision has helped keep me focused on the big picture. He excels at areas that I have struggled with, such as communicating ideas to other people, and this helped me improve in the areas I needed most. His continued patience and concern has helped get me through times that were difficult or stressful.

I would also like to thank Pietro Perona, who was like a second advisor to me. Pietro is truly an impressive person; he is unique in terms of his vision of research, his ability to understand things very quickly, his ability to get straight to the point, and his kindness as a person.

I am grateful to Sanjoy Dasgupta, Gert Lanckriet, Lawrence Saul, Nuno Vasconcelos for being on my committee and teaching the most useful classes at UCSD that I have taken (or sat in on).

I would like to thank my sister, Kristin, who has always been a role model to me. She is the most unselfish and considerate person I know and has been invaluable as a sister, friend, and collaborator. I am thankful for my parents for always supporting me and being people I can completely rely on.

I am especially grateful to Boris Babenko, who served as a mentor to me and helped shape my research interests more than anyone. I look up to Boris as a researcher and a person. I am grateful to my collaborators, Catherine Wah, Oscar Beijbom, Peter Welinder, Eyrun Eyjolfsson, Grant Van Horn, and Florian Schroff, who have all been a pleasure to work with and know as people. I would also like to thank Kai Wang, Sam Kwak, Nakul Verma, Vincent Rabaud, Carolina Galleguillos, Ryan Farrell, and Ron Appel for always giving useful advice and being helpful.

- Parts of Section 3.1 are based on the paper “Multiclass Recognition and Part Localization with Humans in the Loop” by C. Wah and S. Branson and P. Perona and S. Belongie [78]. The dissertation author contributed to developing the algorithm, experiments, and writing of the paper.

- Chapter 4 and Section 3.2 are based on the paper “Strong Supervision From Weak Annotation: Interactive Training of Deformable Part Models” by S. Branson and P. Perona and S. Belongie [8]. The dissertation author developed the algorithm and experiments and wrote most of the paper.
- Chapter 6 is based on work with Oscar Beijbom and Serge Belongie that will be submitted to CVPR 2013. The dissertation author contributed to developing the algorithms and experiments and writing the paper.

VITA

- 2003 B. S. in Computer Science, Stanford University
- 2003 M. S. in Computer Science, Stanford University
- 2013 Ph. D. in Computer Science, University of California, San Diego

PUBLICATIONS

- B. Babenko, S. Branson, and S. Belongie, “Similarity Metrics For Categorization From Monolithic to Category Specific.” *International Conference on Computer Vision (ICCV)*, 2009.
- S. Branson, C. Wah, F. Schroff, P. Welinder, B. Babenko, P. Perona, S. Belongie. “Visual Recognition With Humans in the Loop,” *European Conference on Computer Vision (ECCV)*, 2010.
- P. Welinder, S. Branson, S. Belongie, P. Perona. “The Multidimensional Wisdom of Crowds.” *Neural Information Processing Systems (NIPS)*, 2010.
- P. Welinder, S. Branson, T. Mita, C. Wah, F. Schroff, S. Belongie, P. Perona. “Caltech-UCSD Birds 200.” *California Institute of Technology Tech Report*, 2010.
- C. Wah, S. Branson, P. Perona, S. Belongie. “Multiclass Recognition and Part Localization with Humans in the Loop.” *International Conference on Computer Vision (ICCV)*, 2011.
- S. Branson, P. Perona, S. Belongie. “Strong Supervision From Weak Annotation: Interactive Training of Deformable Part Models.” *International Conference on Computer Vision (ICCV)*, 2011.
- C. Wah, S. Branson, P. Perona, S. Belongie. “Interactive Localization and Recognition of Fine-Grained Visual Categories” *First Workshop on Fine-Grained Visual Categorization (CVPR Workshop)*, 2011.
- C. Wah, S. Branson, P. Perona, S. Belongie. “Birds-200: A Dataset For Fine-Grained Visual Categorization” *First Workshop on Fine-Grained Visual Categorization (CVPR Workshop)*, 2011.
- C. Wah, S. Branson, P. Welinder, S. Belongie, P. Perona. “The Caltech-UCSD Birds-200-2011 Dataset.” *California Institute of Technology Tech Report*, 2012.
- E. Eyjolfsson, X. P. Burgos-Artizzu, S. Branson, K. Branson, D. J. Anderson, P. Perona. “Learning animal social behavior from trajectory features”, *Visual observation and analysis of animal and insect behavior (ICPR Workshop)*, 2012.
- M. Kabra, A. Robie, M. Rivera-Alba, S. Branson, K. Branson. “Automatic Animal Behavior Annotation”, *Nature and Methods*, 2012. To appear.

ABSTRACT OF THE DISSERTATION

**Interactive Learning and Prediction Algorithms For Computer Vision
Applications**

by

Steven Branson

Doctor of Philosophy in Computer Science

University of California, San Diego, 2013

Professor Serge Belongie, Chair

In this dissertation, we explore three different types of interactive methods in computer vision. First, we introduce interactive variants of popular computer vision algorithms, and have applied one of them to create a practical web-based bird species recognition tool. Second, we explore a simple form of active learning that interleaves online learning and interactive labeling of structured objects, and show that it has good properties in theory and practice in terms of scalability to large datasets and complex image models, with some bounds on total annotation effort. Lastly, we investigate interactive feedback methods to researchers and annotators, with the objective of diagnosing errors due to insufficient training data, a bad model or feature space, annotation error, or insufficient computation time. We have combined these methods into a common

software package and applied them to a variety of problems including object detection, pose registration and part-based methods, model sharing methods based on parts and attributes, cost-sensitive multiclass classification, and behavior detection and segmentation.

Chapter 1

Introduction

Over the last two decades, the role of machine learning in computer vision has steadily grown, such that (arguably) the majority of computer vision researchers have come to believe in learning-based methods as the most promising way forward toward solving many or most computer vision problems. Learning-based computer vision is still a young field. In the years prior to me starting my PhD (*e.g.*, the 1990s and early 2000s), there was a rapid influx of new types of machine learning algorithms and a lot of excitement about the possibilities. During the time of my PhD, many have shared the sentiment that learning algorithms—which are often applied as black boxes—didn’t work as well as we expected and we didn’t know why. In the last few years, I believe there has been a positive shift toward viewing learning algorithms as useful and promising; however, they have limitations and need to be applied more realistically.

Terms such as *in the wild*, *big data*, *crowdsourcing*, and *attributes* were all major buzz words (or fads in many people’s opinions) at the time of this dissertation. I have dabbled in all such areas. These trends are all signs of computer vision (and machine learning) transitioning into being more of an applied field; *in the wild* refers to transitioning away from artificial datasets that are collected in controlled environments, *big data* emphasizes the importance of moving to larger-sized datasets, *crowdsourcing* was prioritized as a means of collecting those larger-sized datasets, and *attributes* is a vision-specific applied area that has gained popularity due to the desire to scale computer vision to a larger number of classes or tasks.

My dissertation focuses primarily on searching for answers to the following

questions:

- How can we help transition machine-learning-based computer vision systems from relying on artificial datasets to systems that work well enough to be deployed in real applications?
- How can we scale learning-based computer vision systems to larger problems, finding a feasible solution that considers human annotation time, computation time, and appropriateness and complexity of statistical models?
- When a learned computer vision system performs poorly, how can we diagnose what went wrong?

Toward solving these problems, we have focused primarily on interactive methods, an area that is often more neglected by theoretical approaches to the problem. A more developed view of the interface between learning systems and human users, annotators, and researchers can help solve these problems because: 1) It will allow us to become more in touch with what problems people want to computer vision to solve. 2) It is likely that learning problems in computer vision will require training sets of significantly larger scale, such that more thought has to be put into how to collect them. 3) Solving computer vision will in all likelihood require solving a collection of problems, such that we should be working on developing better methodologies to gradually solve problems rather than expecting someone to invent a single algorithm that instantaneously solves everything.

1.1 Overview and Contributions

In Chapter 2 we describe Visipedia, a user-generated visual encyclopedia, which is the main driving application behind my dissertation. We outline how each of the other chapters of this dissertation contribute toward building this application.

In Chapter 3, we introduce two novel human-in-the-loop applications, one that is applicable to multiclass classification and another that is applicable to pose registration. These have been applied toward producing practical web-based systems, one for bird

species recognition (see Figure 3.1), and one for semantic annotation of images (see Figure 2.1(b)).

In Chapters 4-6 we describe components of a structured learning and annotation software package that we have implemented, which provides a number of features including: computational tractability to large datasets, built-in support for active learning via interactive labeling, ability to add new training examples or relabel examples in online fashion, and feedback mechanisms to diagnose different sources of test error.

In Chapter 4, we describe a simple form of active learning that consists of alternating between online learning and semi-automated interactive annotation. The main advantage of this method is that it takes into consideration not only annotation effort but also computational tractability. It is applicable to models that are state-of-the-art for object detection and pose registration [29, 88, 93]. By contrast, most existing active learning methods are missing details to make them practical in applied settings, due to significant increases in computational costs [74, 34] or limitation to simpler statistical models [60, 61]. Thus most existing methods must rely on either 1) focusing on problems of limited scope, or 2) simulated active learning experiments due to inability to provide realtime interactive learning. Our method is particularly relevant to detection in computer vision, where people have avoided more complex alignment models due to concerns over prohibitive annotation costs. We provide a theoretical argument to suggest that when our style of interactive learning is applied, the most important factor that determines annotation time is the number of training examples required to learn a model with low test error. In this context, the cost of annotating more complex alignment models will be mostly absorbed by interactive labeling. In Chapter 8, we provide an argument that suggests that more complex alignment models may have better scalability properties than many of the most commonly used machine learning techniques and features that are used in computer vision.

In Chapter 5, we discuss feedback mechanisms to help diagnose different sources of test error due to insufficient training data, a bad model or feature space, annotation error, or insufficient computation time. We also derive online updates to apply subsequent changes to fix them without recollecting an entirely new dataset or re-training from scratch.

In Chapter 6, we develop customized optimization algorithms that incorporate a small amount of application-specific tweaking for common problems such as object detection and deformable part model training. We show that these customized optimization algorithms are faster than methods based on mapping problems into binary classification and than commonly used structured learning packages.

In Chapter 7, we show how a number of different computer vision applications can be mapped into this learning and annotation package, including cost-sensitive multiclass SVMs, sliding window object detection, deformable part models, pose mixture models with occlusion reasoning, attribute-based classification, multiclass detection with shared parts, localized attribute detection, behavior segmentation, object tracking, articulated tracking, and segmentation. Most of these methods are implemented and included in a software toolbox that I am planning to release shortly (see <http://vision.ucsd.edu/visipedia/code>). The implementation is inclusive of methods that obtain state-of-the-art performance on popular datasets for multiclass classification [53] (on datasets such as Caltech-256 [31] and ImageNet [18]) and object detection [29] (on datasets such as VOC Pascal [23]). It includes methods that have improved state-of-the-art detection performance by using different types of annotation on applications such as human body detection [88], faces [93], and birds [8]. It also includes a few novel methods in emerging problems, including a more sophisticated part/pose model with occlusion reasoning (Section 7.2.3), improved localized attribute detection and sharing models for multiclass object detection (Section 7.2.3), and a more advanced time-localized behavior and action detection method (Section 7.6).

1.2 Preliminaries

In this section, we outline some of the methods and problem definitions that are re-occurring in many parts of this dissertation. We define these problems in the context of how they are used in this dissertation and not in the most general sense.

1.2.1 Structured Prediction

Let X be a data point (*e.g.* an image) and $Y = y_1 \dots y_O$ be a multidimensional structured label. For example, for object detection, one could define $Y = \{x, y, w, h\}$ as the four coordinates of the bounding box of an object. For deformable part models, one could define $Y = y_1 \dots y_P$ as a set of P part locations, each of which defines the bounding box of a particular part. For image segmentation, one could define $Y = y_1 \dots y_N$ as a set of class labels for each of N pixels in the image.

In this dissertation, we consider structured output functions that predict the label with highest score according to a linear score function:

$$g(X; \mathbf{w}) = \arg \max_Y \langle \mathbf{w}, \Psi(X, Y) \rangle \quad (1.1)$$

where $\langle \mathbf{w}, \Psi(X, Y) \rangle$ is a discriminative score measuring the goodness of a particular label Y , $\Psi(X, Y)$ is a feature space extracted with respect to label Y (*e.g.*, features extracted from within a bounding box $Y = \{x, y, w, h\}$), and \mathbf{w} is a vector of weights that parameterize the model.

Eqn 1.1 can be interpreted as performing maximum likelihood inference on some log-linear Markov random field with customizable structure

$$g(X; \mathbf{w}) = \arg \max_Y p(Y|X) \quad (1.2)$$

$$p(Y|X) \propto \exp\{\langle \mathbf{w}, \Psi(X, Y) \rangle\} \quad (1.3)$$

Although Eqn 1.1 uses a linear score function, one can obtain non-linear decision functions via the max function and definition of Y . For example, Y could encode a mixture component variable that selects between different components of a mixture model.

1.2.2 Structured Learning

Let $\Delta(Y, Y_i)$ be a customizable loss function associated with predicting label Y when the true label is Y_i . For example, $\Delta(Y, Y_i)$ can be a measure of the overlap between predicted part locations and their ground truth locations. In this dissertation, we consider the structured SVM learning objective, which minimizes a hinge-loss-like

training error over a training set $D_n = Z_1 \dots Z_n$ of instance-label pairs $Z_i = (X_i, Y_i)$:

$$F_n(\mathbf{w}) = n \left(\frac{\lambda}{2} \|\mathbf{w}\|^2 + \frac{1}{n} \sum_{i=1}^n \epsilon_i \right) \quad (1.4)$$

$$\text{s.t.}, \forall_{i,Y}, \langle \mathbf{w}, \Psi(X_i, Y) \rangle + \Delta(Y, Y_i) \leq \langle \mathbf{w}, \Psi(X_i, Y_i) \rangle + \epsilon_i \quad (1.5)$$

where λ is a regularization constant. The objective adds slack variables ϵ_i that place an upper bound on $\Delta(Y, Y_i)$ that is convex in \mathbf{w} . An equivalent way of writing Eq 1.4 is

$$F_n(\mathbf{w}) = \sum_{i=1}^n f(\mathbf{w}; Z_i) \quad (1.6)$$

$$f(\mathbf{w}; Z_i) = \frac{\lambda}{2} \|\mathbf{w}\|^2 + \ell(\mathbf{w}, Z_i) \quad (1.7)$$

$$\ell(\mathbf{w}, Z_i) = \max_Y (\langle \mathbf{w}, \Psi(X_i, Y) \rangle + \Delta(Y, Y_i)) - \langle \mathbf{w}, \Psi(X_i, Y_i) \rangle \quad (1.8)$$

For binary classification, where $Y = \pm 1$, $\Psi(X, Y) = .5Y\phi(X)$, and $\Delta(Y, Y_i) = 1[Y \neq Y_i]$, it is easy to see that $\ell(\mathbf{w}, Z_i) = \max(0, 1 - Y\langle \mathbf{w}, \phi(X) \rangle)$ is the familiar SVM hinge loss.

While the structured SVM learning objecting is just one possible learning objective for structured data, it has a few desirable properties: 1) it is a superset of many of the most popular and highest performing learning algorithms for object recognition, object detection, and segmentation, 2) it is strongly convex and efficiently optimizable, and 3) optimizing it can provide some PAC bounds on the expected loss for the loss function Δ .

Note that $\ell(\mathbf{w}, Z_i)$ is non-differentiable and is computed as a maximum over all possible output labels Y , which is in general exponential in the dimensionality of the output space O . Despite this, it is convex in \mathbf{w} (since it is the maximum of a set of affine functions), and as such Eqn 6.19 can be efficiently optimized using techniques such as sub-gradient or cutting plane methods. Both such algorithms can be solved efficiently if one can solve the following inference-like problem:

$$\bar{Y} = \arg \max_Y (\langle \mathbf{w}, \Psi(X, Y) \rangle + \Delta(Y, Y_i)) \quad (1.9)$$

The sub-gradient of Eq 1.7 $\nabla f(\mathbf{w}^{t-1}; Z_t)$ can easily be computed from \bar{Y} as

$$\nabla f(\mathbf{w}; Z_i) = \lambda \mathbf{w} + \nabla \ell(\mathbf{w}; Z_i) \quad (1.10)$$

$$\nabla \ell(\mathbf{w}; Z_i) = \Psi(X, \bar{Y}) - \Psi(X, Y) \quad (1.11)$$

See Chapter 6 for a discussion of optimization methods, and Chapter 7 for a list of example applications that are expressed in this notation.

1.2.3 Interactive Labeling

We define interactive prediction as a modified form of structured prediction, where the predicted output is constrained to agree with a user-provided partial annotation \tilde{Y} to a subset of the variables in the structured output space. Let subscripts be defined such that $\tilde{y}_i \in \tilde{Y}$ refers to the same variable $y_i \in Y$ in a fully labeled output Y . In our problem formulation, interactive prediction solves the optimization problem:

$$\begin{aligned} h(X, \tilde{Y}; \mathbf{w}) &= \arg \max_Y \langle \mathbf{w}, \Psi(X, Y) \rangle & (1.12) \\ \text{s.t. } &\forall_{\tilde{y}_i \in \tilde{Y}}, \tilde{y}_i = y_i \end{aligned}$$

We consider an interactive labeling session to be a semi-automated process for annotating the label of a particular example X . It consists of a sequence of user interactions $\tilde{Y}_t = (\tilde{y})_{j(1)}, (\tilde{y})_{j(2)}, \dots, (\tilde{y})_{j(t)}$, where the user labels the $j(t)$ -th variable at timestep t . The computer system interactively displays the highest scoring consistent solution $Y_t^* = h(X, \tilde{Y}_t; \mathbf{w})$ at each timestep, and the user terminates the labeling session when the loss of the predicted solution is zero $\Delta(Y, Y_t^*) = 0$. Thus some amount of automation occurs when the number of timesteps is smaller than the number of variables in the structured output space $t < O$.

Details of implementing interactive labeling algorithms for a variety of different applications such as deformable part models, segmentation, and tracking are provided in Chapter 7 and Section 3.2.

1.2.4 Active Learning

Active learning is a form of supervised learning in which the learner interactively queries training labels from annotators. In the context of this dissertation, we assume that the objective is to come up with some labeling strategy for learning model parameters \mathbf{w} that obtain sufficiently low test error while minimizing the total expected amount of human interaction. For structured problems, this labeling strategy could en-

tail choosing which training instance (X, Y) to query, or choosing which structured variable $y_i \in Y$ to query.

In Chapter 4, we consider a practical (but unambitious) active learning strategy that aims to be more tractable to higher dimensional label for structured output labels Y . It consists of alternating between using interactive labeling to label a particular (randomly chosen) instance (X, Y) and online learning to update w . It could be combined with more traditional querying strategies (see [60] for a good discussion of possible methods) to choose which training instances to label.

1.2.5 Active Prediction

In interactive labeling, we assume that the user chooses which variable $y_{j(t)}$ to label in each timestep t (by viewing a visualization of the current predicted labels $h(X, \tilde{Y}; w)$). By contrast, in active prediction, the computer system chooses which variable $y_{j(t)}$ to interactively query from the user at each timestep t , with the objective of arriving at the true label Y as quickly as possible. It is similar to active learning in that the goal is to minimize the total expected amount of human interaction; however, the goal is to arrive at a label Y rather than to learn model parameters w . Typically, the parameters of the model w remain fixed during user interaction for a particular example X .

In Section 3.1 we explore an active prediction algorithm that is applicable to multiclass object classification.

Acknowledgements

This chapter was written exclusively as part of this dissertation.

Chapter 2

Visipedia

The driving application behind my dissertation is Visipedia, a joint project between Pietro Perona's Vision Group at Caltech and Serge Belongie's Vision Group at UCSD. Visipedia, short for Visual Encyclopedia, is an augmented version of Wikipedia, where pictures are first-class citizens alongside text. Goals of Visipedia include creation of hyperlinked, interactive images embedded in Wikipedia articles, scalable representations of visual knowledge, largescale machine vision datasets, and visual search capabilities. Toward achieving these goals, Visipedia advocates interaction and collaboration between machine vision and human users.

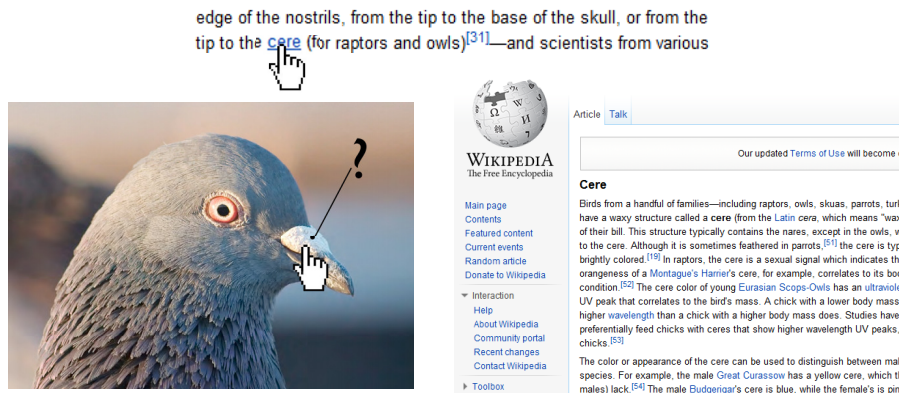
The set of applications that we are hoping to support is described in Figure 2.1-2.2. Two driving motivations behind Visipedia are: 1) An attempt to tap into expert knowledge and citizen science to help solve largescale annotation problems (*e.g.*, bird enthusiasts will be increasingly incentivized and capable of annotating images pertaining to birds), 2) To allow the set of problems that we want computer vision to solve to be defined and grow organically, in the same way that the set of Wikipedia articles written is based on user interest.

To keep the problem tractable for our research group, we have begun with a proof-of-concept in one particular domain (birds, see Section 2.1).



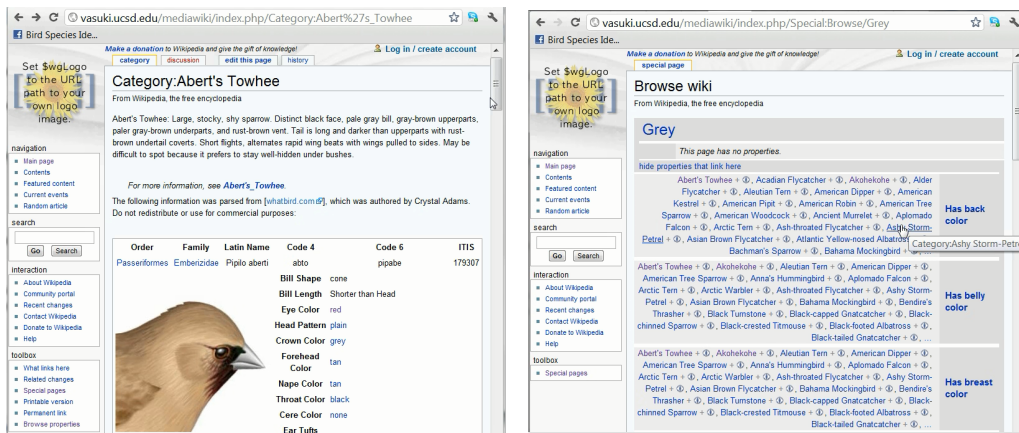
(a) Search By Image

(b) Interactive Part Diagrams

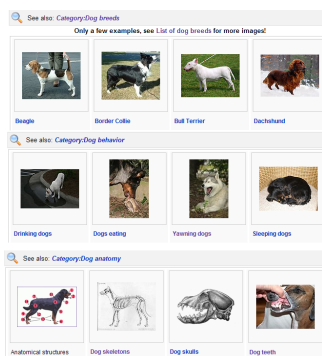


(c) Visual Hyperlinks

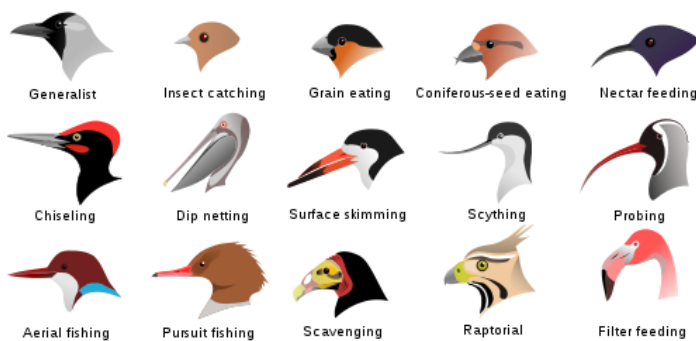
Figure 2.1: Applications of Visipedia: a) A user takes a picture of a bird, which the system recognizes the object and uses to bring up the corresponding Wikipedia article. This is enabled by the recognition algorithms described in Section 7.3.3 and Section 3.1. b) Interactive part diagrams can be embedded into Wikipedia pages. This is the output of our part annotation tool described in Section 7.2.4, which can be created in an automated or semi-automated fashion (Section 3.2). c) The user clicks on an unknown structure on the beak of a pigeon, and the system helps to identify it as a cere. This is enabled by the localization algorithms described in Section 7.2.3.



(a) Browseable Visual Knowledge Representation



(b) Dynamic Image Galleries



(c) Organization of Visual Concepts

Figure 2.2: Applications of Visipedia: a) Classes can be annotated with semantic attributes, which can be used to render HTML using customizable templates, or to browse and search through classes by attributes. This was created using existing mediawiki software, and populated with our data from the Birds-200 dataset. b) Images that are labeled with part and attribute annotations enable dynamic image galleries or image search by attribute. c) Users represent visual concepts using exemplars or diagrams.

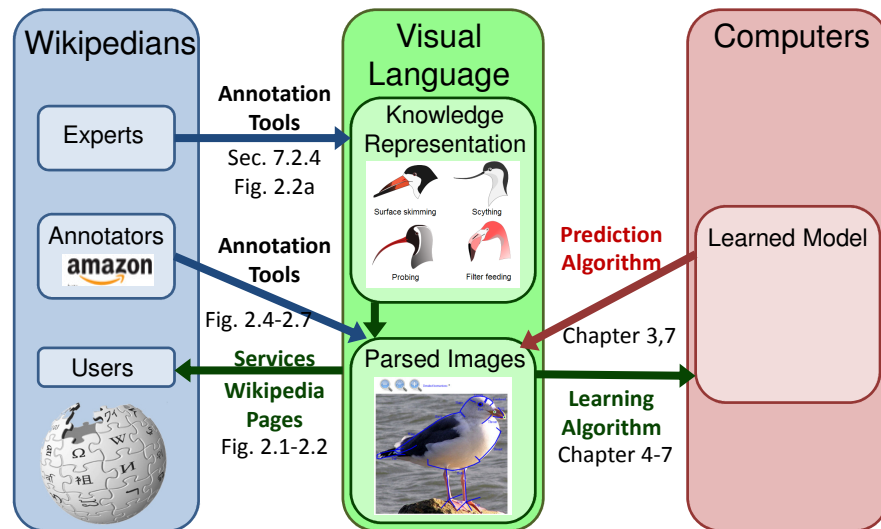


Figure 2.3: Overview of the Architecture of Visipedia: An overview of the approach this dissertation takes toward building Visipedia. A visual language that both humans and computers can understand is established; we use a certain definition of parts (Section 7.2.3) and attributes (Section 7.3.3). Images can be annotated in terms of this language either by humans using the annotation tools shown in Fig 2.6-2.8, in automated fashion using learned computer vision algorithms (Section 7.2.3,7.3.3), or in semi-automated fashion using some combination of the two (Chapter 3). The semantically annotated images can be used to provide a number of web-based features/services that can be used to enrich Wikipedia/Visipedia articles (Fig 2.1-2.2). The semantically annotated images can also be used to train machine learning algorithms that can be used to improve the efficiency of automated prediction algorithms (Chapters 4-6). The vocabulary of parts and attributes is defined by experts/Wikipedians using tools such as those depicted in Fig 7.4 and Fig 2.2(a).

2.1 The Birds-200 Dataset

In this dissertation, we do many experiments on CUB-Birds-200-2011 [86], a dataset that we collected that contains over 10,000 images over 200 different bird species. All images were annotated with bounding boxes, part locations, segmentations, and attribute labels.

Bird species classification is a difficult problem that pushes the limits of the visual abilities for both humans and computers. Birds as a data source have many properties that highlight weaknesses of contemporary computer vision algorithms: 1) success in object recognition has often been limited to datasets where intraclass variation is fairly small and interclass variation is very high (*e.g.* distinguishing faces from airplanes on Caltech-4 [31]), and 2) whereas, computer recognition works very well on highly textured, rigid objects such as book cover or movie poster recognition and fairly well on mostly rigid objects like faces, computer vision algorithms struggle on highly deformable objects. Birds is typically the lowest scoring category in the VOC Pascal Detection (of 20 object categories), due to extreme variation in pose (*e.g.*, flying birds, swimming birds, and perched birds that are partially occluded by branches) and large variations in shape and appearance for different bird species (*e.g.*, consider pelicans vs. sparrows). At the same time, other pairs of bird species are nearly visually indistinguishable, even for expert bird watchers (*e.g.*, many sparrow species are visually similar).

It is our hope that Birds-200 will facilitate research in subordinate categorization by providing a comprehensive set of benchmarks and annotation types for one particular domain (birds). We would like to cultivate a level of research depth that has thus far been reserved for a few select categories such as pedestrians and faces. Focusing on birds will help keep research more tractable from a logistical and computational perspective. At the same time, we believe that many of the lessons learned (in terms of annotation procedures, localization models, feature representations, and learning algorithms) will generalize to other domains such as different types of animals, plants, or objects.

2.1.1 Motivation For a Bird Dataset

Birds-200 has a number of unique properties that we believe are of interest to the computer vision research community:

Subordinate category recognition: Methods that are popular on datasets such as Caltech-101 [31] (*e.g.*, lossy representations based on histogramming and bag-of-words) are often less successful on subordinate categories, due to higher visual similarity of categories. Research in subordinate categorization may help encourage development of features or localization models that retain a greater level of discriminative power.

Multi-class object detection and part-based methods: Part-based methods have recently experienced renewed interest and success [29]. Unfortunately, availability of datasets with comprehensive part localization information is still fairly limited. Additionally, whereas datasets for image categorization often contain hundreds or thousands of categories [31, 18], popular datasets for object detection rarely contain more than 20 or so categories [23] (mostly due to computational challenges). Methods that employ shared part models offer great promise toward scaling object detection to a larger number of categories. Birds-200 contains a collection of 200 different bird species that are annotated using the same basic set of parts, thus making it uniquely suited toward research in shared part models.

Attribute-based methods: Attributes are a form of model sharing that has recently become popular. Most existing datasets for attribute-based recognition (*e.g.*, Animals With Attributes [40]) do not contain localization information. This is an obstacle to research in attributed-based recognition, because visual attributes are often naturally associated with a particular part or object (*e.g.* blue belly or cone-shaped beak).

Crowdsourcing and user studies: Annotations such as part locations and attributes open the door for new research opportunities, but are also subject to a larger degree of annotation error and user subjectivity as compared to object class labels. By releasing annotations from multiple MTurk users per training image, we hope to encourage research in crowdsourcing techniques for combining annotations from multiple users, and facilitate user studies evaluating the reliability and relative merit of different types of annotation.

2.1.2 Dataset Specification and Collection

Bird Species: The dataset contains 11,788 images of 200 bird species. Each species is associated with a Wikipedia article and organized by scientific classification (*order, family, genus, species*). The list of species names was obtained using an online field guide¹. Images were harvested using Flickr image search and then filtered by showing each image to multiple users of Mechanical Turk [85]. Each image is annotated with bounding box, part location, and attribute labels. See Fig 2.4 for example images and Fig 2.9 for more detailed dataset statistics.

Bounding Boxes: Bounding boxes were obtained using the interface in Fig. 2.7.

Attributes: A vocabulary of 28 attribute groupings (see Fig 2.5(b)) and 312 binary attributes (*e.g.*, the attribute group *belly color* contains 15 different color choices) was selected based on an online tool for bird species identification². All attributes are visual in nature, with most pertaining to a color, pattern, or shape of a particular part. Attribute annotations were obtained for each image using the interface in Fig. 2.8.

Part Locations: A total of 15 parts (see Fig 2.5(a)) were annotated by pixel location and visibility in each image using the GUI shown in Fig 2.6(a). The “ground truth” part locations were obtained as the median over locations for 5 different Mechanical Turk users per image.

Segmentations: Segmentations were added to the dataset by Ryan Farrell [27]. A figure-ground segmentation was obtained from 5 different Mechanical Turk users for each image.

2.1.3 Benchmarks and Baseline Experiments

We introduce a set of benchmarks and baseline experiments for studying bird species categorization, detection, and part localization:

1. **Localized Species Categorization:** *Given the ground truth part locations, assign each image to one of 200 bird classes.* This benchmark is intended to facilitate studies of different localization models (*e.g.*, to what extent does localization

¹<http://www.birdfieldguide.com>

²<http://www.whatbird.com>

information improve classification accuracy?), and also provide greater accessibility to existing categorization algorithms. Using RGB color histograms and histograms of vector-quantized SIFT descriptors with a linear SVM, we obtained a classification accuracy of 17.3% (see Fig 2.10(d)).

2. **Part Localization:** *Given the full, uncropped bird images, predict the location and visibility of each bird part.* We measured the distance between predicted part locations and ground truth, normalized on a per-part basis by the standard deviation over part click locations for multiple MTurk users. The maximum error per part was bounded at 5 standard deviations. This was also the error associated with misclassification of part visibility. Using HOG-based part-detectors and a mixture of tree-structured pictorial structures, we obtained an average error of 1.47 standard deviations (by contrast, an average MTurk user should be off by 1 standard deviation). See Fig 2.11 for example part localization results and their associated loss.
3. **Species Categorization/Detection:** *Using only the full, uncropped bird images, assign each image to one of 200 bird classes.* For this benchmark, one can use the method of his/her choice (*e.g.*, image categorization, object detection, segmentation, or part-based detection techniques); however, since the images are uncropped, we anticipate that the problem cannot be solved with high accuracy without obtaining some degree of localization. Detecting the most likely part configuration using a universal bird detector (as for benchmark 2) and then applying a localized species classifier (as for benchmark 1), we obtained a classification accuracy of 10.3% (see Fig 2.10(b)).

Acknowledgements

This section is based on a dataset collected with Catherine Wah, Peter Welinder, Pietro Perona, and Serge Belongie. The dissertation author contributed to building the tools to collect the dataset and experiments.

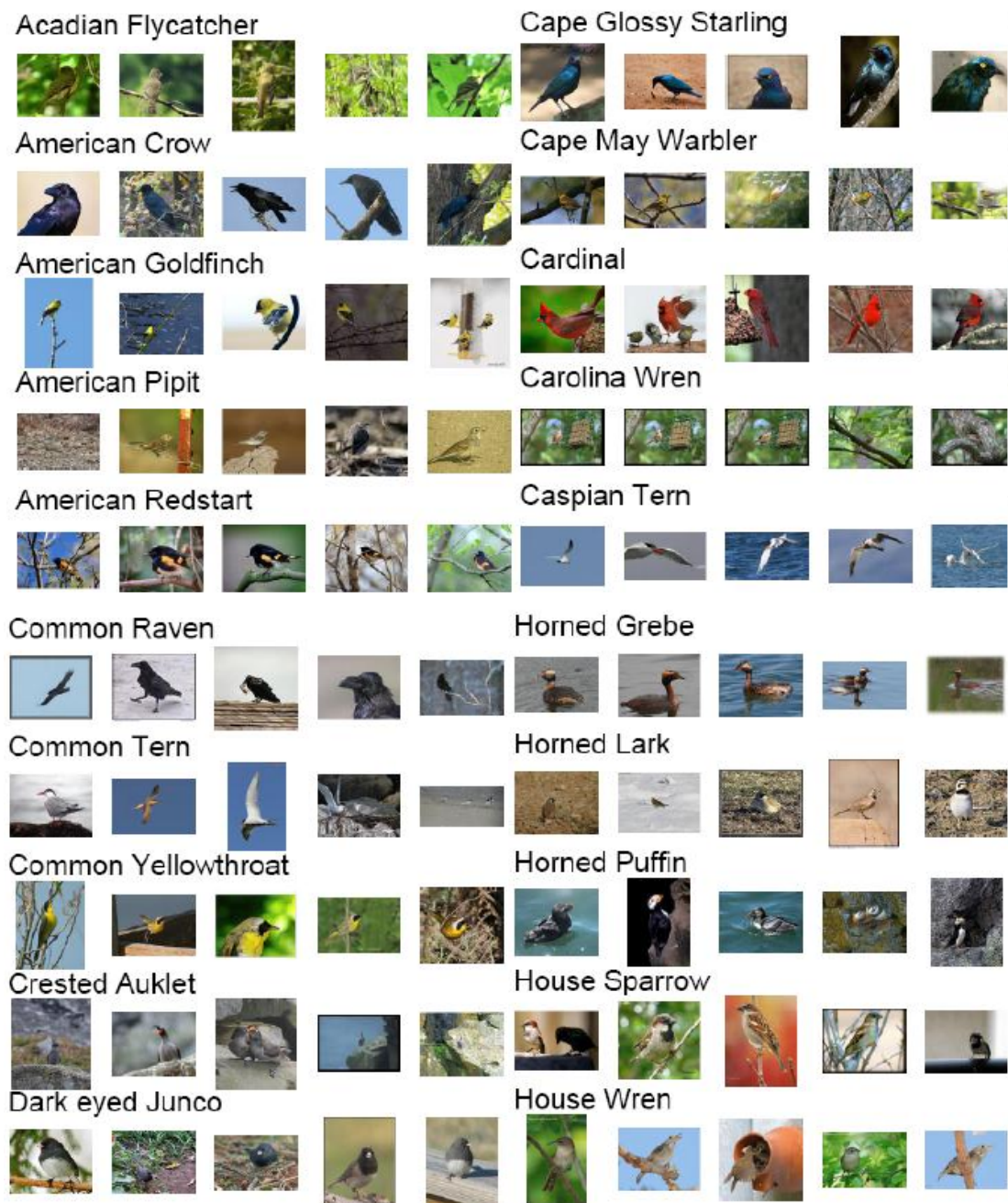
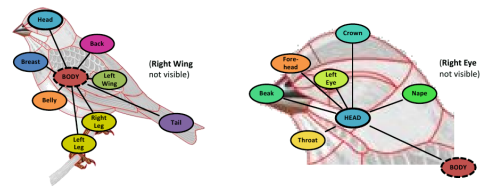


Figure 2.4: CUB-200-2011 Example Images, showing a few example images from 20 different classes (the full dataset contains 200 classes)



(a) Collected Parts


Part	Attributes	Part	Attributes	Part	Attributes
Beak	HasBillShape, HasBillColor, HasBillLength	Back	HasBackColor, HasBackPattern	Breast	HasBreastPattern, HasBreastColor
Belly	HasBellyPattern, HasBellyColor	Fore-head	HasForehead Color	Bird (all parts)	HasSize, HasShape
Throat	HasThroatColor	Nape	HasNapeColor	Head	HasHeadPattern
Crown	HasCrownColor	Eye	HasEyeColor	Leg	HasLegColor
Tail	HasUpperTailColor, HasUnderTailColor, HasTailPattern, HasTailShape	Wing	HasWingPattern, HasWingColor, HasWingShape	Body	HasUnderpartsColor, HasUpperPartsColor, HasPrimaryColor

(b) Attribute Part Associations

Figure 2.5: Collected Parts and Attributes. (a) The 15 part location labels collected for each image. (b) The 28 attribute-groupings that were collected for each image, and the associated part for localized attribute detectors.



Locate the BEAK



Click on the image where the specified bird's part is located.

*Keyboard shortcuts:
left arrow for BACK, right arrow for NEXT.*

Check this box if **more than half** of the part is not visible in the image.

Go Back
Next

1/25

(a) Part GUI

Figure 2.6: MTurk GUI for collecting part location labels, deployed on 11,788 images for 15 different parts and 5 workers per image.

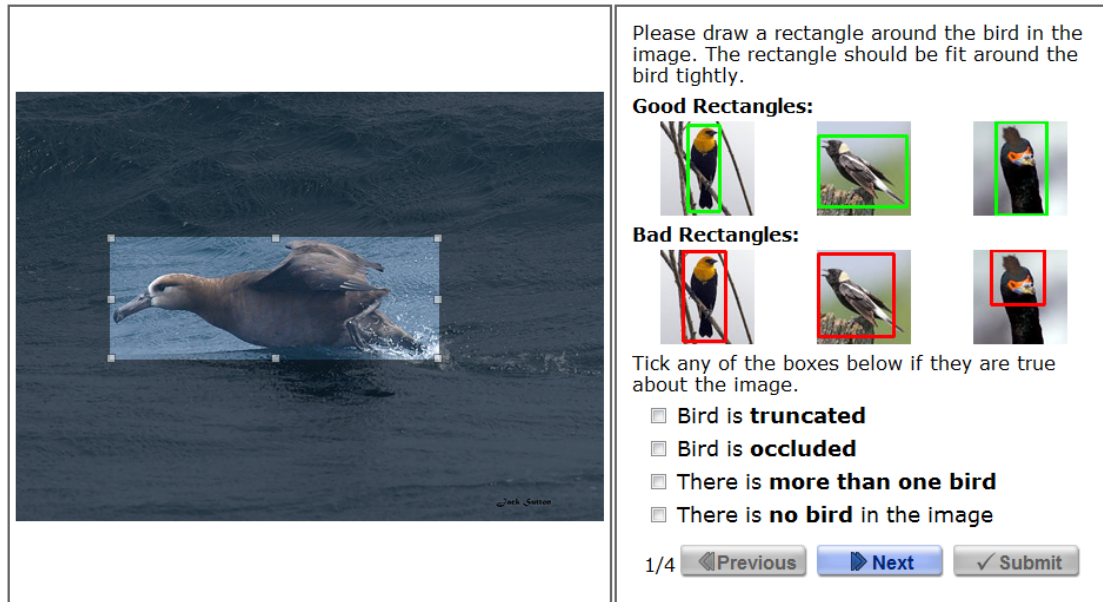


Figure 2.7: MTurk GUI for collecting bounding box labels, deployed on 11,788 images.



Figure 2.8: MTurk GUI for collecting attribute labels, deployed on 11,788 images for 28 different questions and 312 binary attributes.

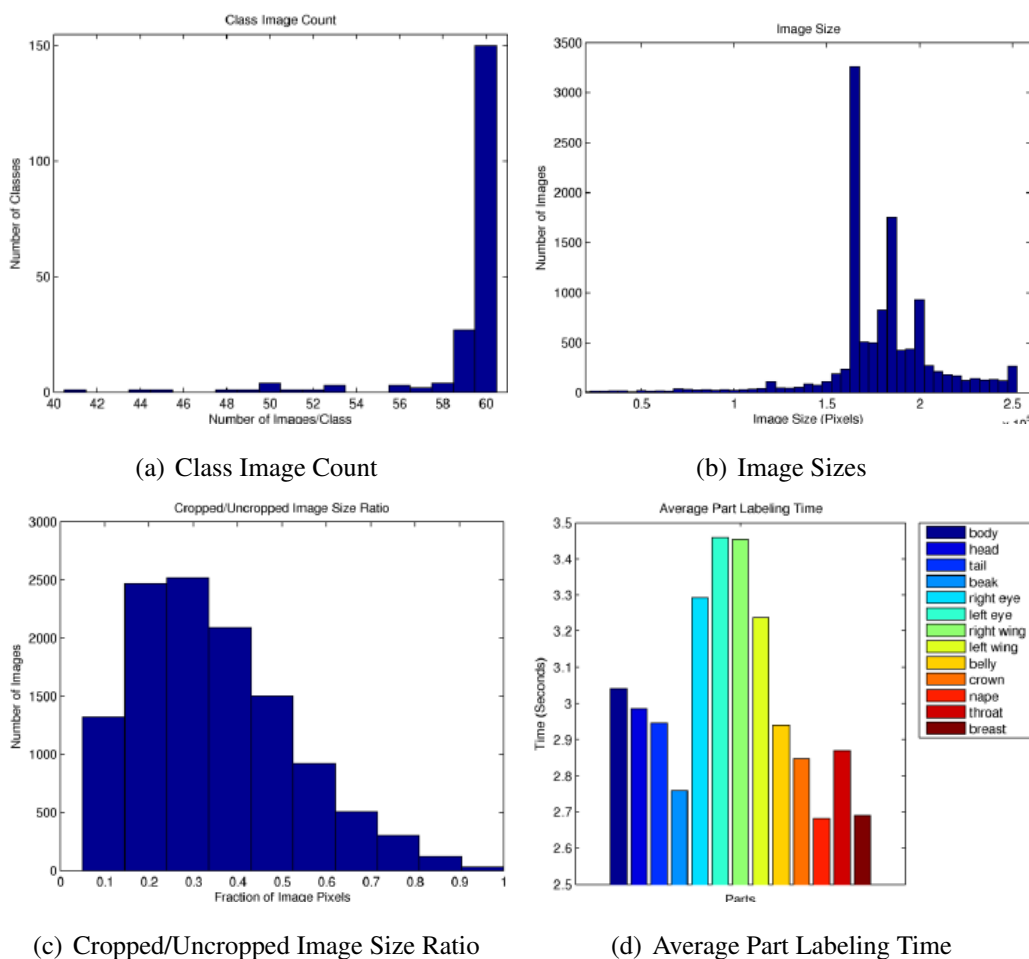


Figure 2.9: Dataset Statistics. (a) Distribution of the number of images per class (most classes have 60 images). (b) Distribution of the size of each image in pixels (most images are roughly 500X500). (c) Distribution of the ratio of the area of the bird’s bounding box to the area of the entire image. (d) The average amount of time it took MTurkers to label each part.

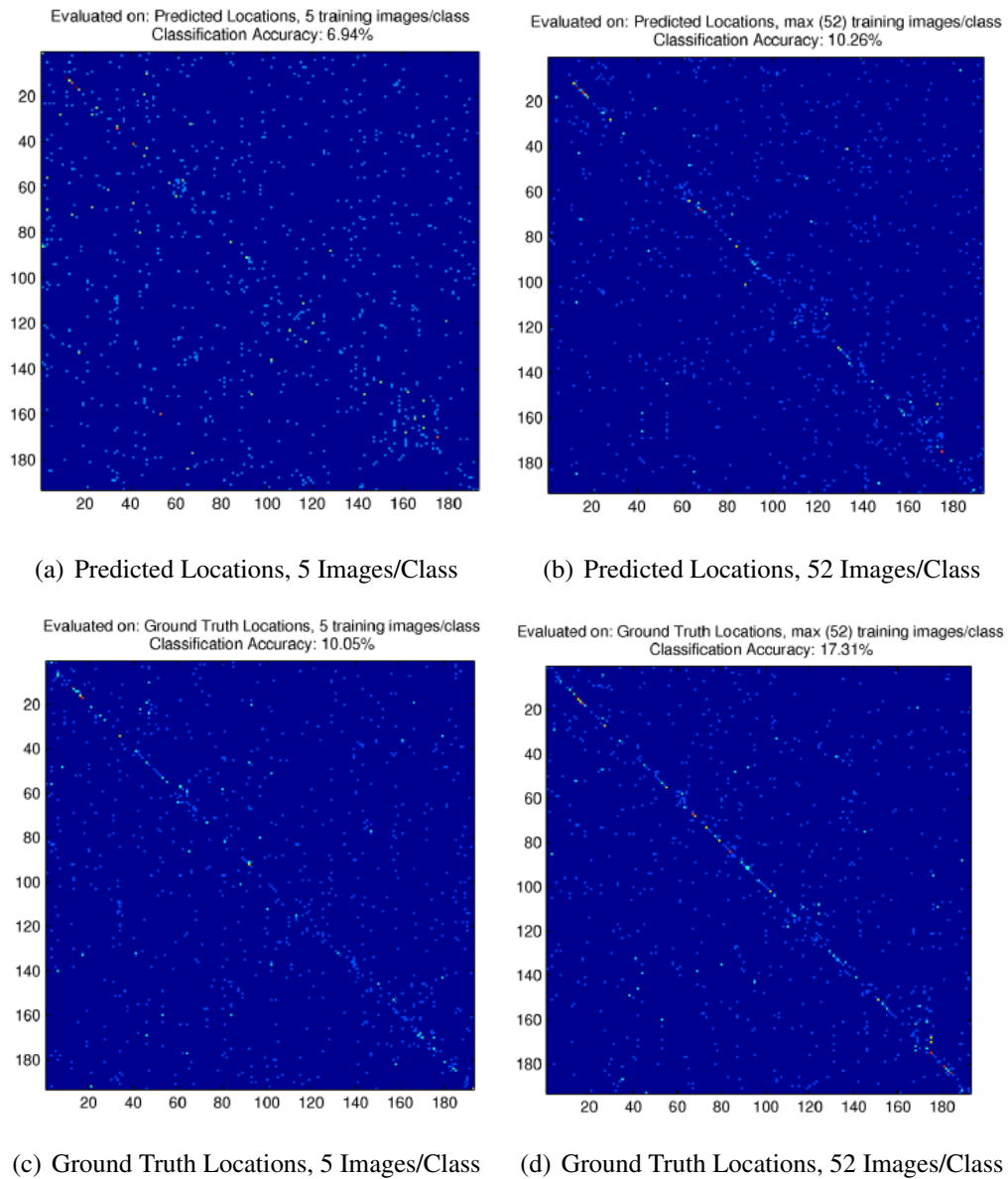


Figure 2.10: Categorization Results for 200-way bird species classification. The top 2 images show confusion matrices when using a universal bird detector to detect the most likely location of all parts and then evaluating a multiclass classifier. The bottom 2 images show confusion matrices when evaluating a multiclass classifier on the ground truth part locations. The 2 images on the left show results with 5 training images per class, and the images on the right show results with 52 training images per class.



Figure 2.11: Example Part Detection Results, with good detection results on the left and bad detection results on the right. A loss of 1.0 indicates that the predicted part locations are about as good as the average MTurk labeler.

Chapter 3

Interactive Computer Vision

In this chapter, we introduce two novel application-specific human-in-the-loop computer vision algorithms, where humans and computers collaborate interactively to solve some computer vision problem. In this case, we acknowledge that the performance of computer vision systems is still imperfect and use computer vision to attempt to semi-automate some task, minimizing the amount of human time required to obtain some useable result. We advocate human-in-the-loop methods as a means of coping with the fact that most contemporary computer vision algorithms perform too poorly to be useable in practice. By contrast, we believe that computer vision as a field has coped with this problem by positioning itself as a more theoretical discipline and not as an applied field, relying on datasets and benchmarks that are frequently criticized as being artificial or toy problems. Deploying human-in-the-loop systems has a few practical benefits: 1) consumers can benefit from computer vision systems today, 2) it helps prioritize computer vision research toward solving real life problems, 3) it provides qualitative feedback to researchers about which components of their system are currently deficient, such that they are incentivized to fix them, 4) it promotes greater transparency in sharing the deficiencies of our algorithms with other researchers (whereas in our current research culture, we are often guilty of hiding the major weaknesses from fellow researchers).

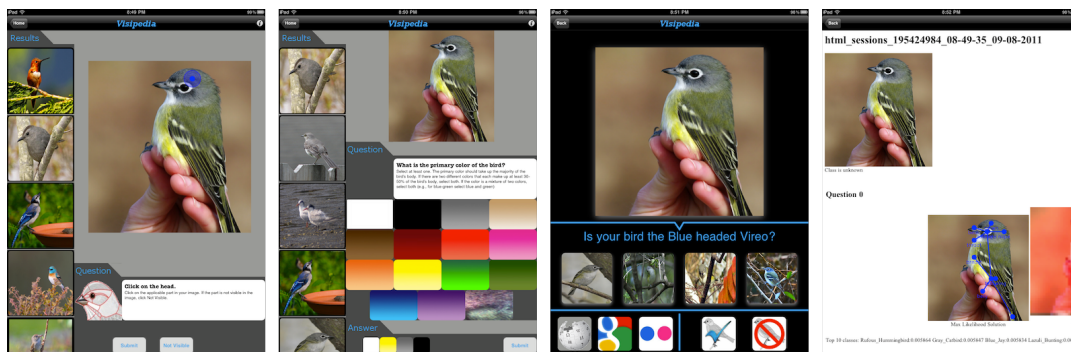


Figure 3.1: Screen Capture of an iPhone App for Bird Species Recognition: A user takes a picture of a bird she wants to recognize, and it is uploaded to a server. The server runs computer vision algorithms to localize the different parts of the bird and predict the bird species (debug output of the algorithms is shown in the image on the right). The computer system intelligently selects a series of questions to ask (*click on the head, what is the primary color of the bird?*) that are designed to reduce its ambiguity about the predicted bird species as quickly as possible

3.1 Visual Recognition With Humans in the Loop

In this section, we present an interactive, hybrid human-computer method for object classification. The method applies to classes of objects that are recognizable by people with appropriate expertise (*e.g.*, animal species or airplane model), but not (in general) by people without such expertise. It can be seen as a visual version of the *20 questions game*, where questions based on simple visual attributes are posed interactively. The goal is to identify the true class while minimizing the number of questions asked, using the visual content of the image. We introduce a general framework for incorporating almost any off-the-shelf multi-class object recognition algorithm into the visual 20 questions game, and provide methodologies to account for imperfect user responses and unreliable computer vision algorithms. We evaluate our methods on *Birds-200*, a difficult dataset of 200 tightly-related bird species, and on the *Animals With Attributes* dataset. Our results demonstrate that incorporating user input drives up recognition accuracy to levels that are good enough for practical applications, while at the same time, computer vision reduces the amount of human interaction required.



Figure 3.2: Examples of classification problems that are easy or hard for humans. While basic-level category recognition (left) and recognition of low-level visual attributes (right) are easy for humans, most people struggle with finer-grained categories (middle). By defining categories in terms of low-level visual properties, hard classification problems can be turned into a sequence of easy ones.

3.1.1 Introduction

Multi-class object recognition has undergone rapid change and progress over the last decade. These advances have largely focused on types of object categories that are easy for humans to recognize, such as motorbikes, chairs, horses, bottles, *etc.* Finer-grained categories, such as specific types of motorbikes, chairs, or horses are more difficult for humans and have received comparatively little attention. One could argue that object recognition as a field is simply not mature enough to tackle these types of finer-grained categories. Performance on basic-level categories is still lower than what people would consider acceptable for practical applications (state-of-the-art accuracy on Caltech-256[31] is $\approx 45\%$, and $\approx 28\%$ in the 2009 VOC detection challenge [24]. Moreover, the number of object categories in most object recognition datasets is still fairly low, and increasing the number of categories further is usually detrimental to performance [31].

On the other hand, recognition of finer-grained subordinate categories is an important problem to study – it can help people recognize types of objects they don't yet know how to identify. We believe a hybrid human-computer recognition method is a practical intermediate solution toward applying contemporary computer vision algorithms to these types of problems. Rather than trying to solve object recognition entirely, we take on the objective of minimizing the amount of human labor required. As research in object recognition progresses, tasks will become increasingly automated, until even-

tually we will no longer need humans in the loop. This approach differs from some of the prevailing ways in which people approach research in computer vision, where researchers begin with simpler and less realistic datasets and progressively make them more difficult and realistic as computer vision improves (*e.g.*, Caltech-4 \rightarrow Caltech-101 \rightarrow Caltech-256). The advantage of the human-computer paradigm is that we can provide usable services to people in the interim-period where computer vision is still unsolved. This may help increase demand for computer vision, spur data collection, and provide solutions for the types of problems people outside the field want solved.

In this work, our goal is to provide a simple framework that makes it as effortless as possible for researchers to plug their existing algorithms into the human-computer framework and use humans to drive up performance to levels that are good enough for real-life applications. Implicit to our model is the assumption that lay-people generally cannot recognize finer-grained categories (*e.g.*, Myrtle Warbler, Thruxton Jackaroo, *etc.*) due to imperfect memory or limited experiences; however, they do have the fundamental visual capabilities to recognize the parts and attributes that collectively make recognition possible (see Fig. 3.2). By contrast, computers lack many of the fundamental visual capabilities that humans have, but have perfect memory and are able to pool knowledge collected from large groups of people. Users interact with our system by answering simple yes/no or multiple choice questions about an image or object, as shown in Fig. 3.3. Similar to the *20-questions game*¹, we observe that the number of questions needed to classify an object from a database of C classes is usually $O(\log C)$ (when user responses are accurate), and can be faster when computer vision is in the loop. Our method of choosing the next question to ask uses an information gain criterion and can deal with noisy (probabilistic) user responses. We show that it is easy to incorporate any computer vision algorithm that can be made to produce a probabilistic output over object classes.

Our experiments in this paper focus on bird species categorization, which we take to be a representative example of recognition of tightly-related categories. The bird dataset contains 200 bird species and over 6,000 images. We believe that similar methodologies will apply to other object domains.

¹See for example <http://20q.net>.

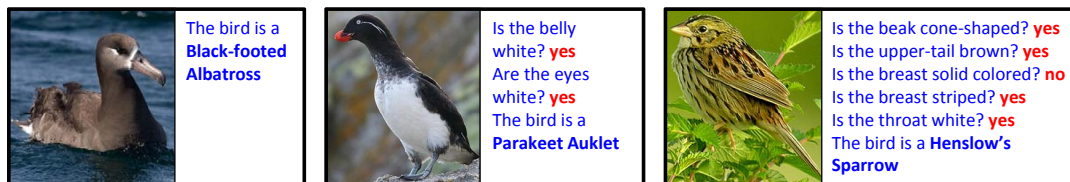


Figure 3.3: Examples of the visual 20 questions game on the 200 class Bird dataset. Human responses (shown in red) to questions posed by the computer (shown in blue) are used to drive up recognition accuracy. In the left image, computer vision algorithms can guess the bird species correctly without any user interaction. In the middle image, computer vision reduces the number of questions to 2. In the right image, computer vision provides little help.

The structure of the paper is as follows: In Section 3.1.2, we discuss related work. In Section 3.1.3, we define the hybrid human-computer problem and basic algorithm, which includes methodologies for modeling noisy user responses and incorporating computer vision into the framework. We describe our datasets and implementation details in Section 3.1.5, and present empirical results in Section 3.1.6.

3.1.2 Related Work

Recognition of tightly related categories is still an open area in computer vision, although there has been success in a few areas such as book covers and movie posters (*e.g.*, rigid, mostly flat objects [50]). The problem is challenging because the number of object categories is larger, with low interclass variance, and variability in pose, lighting, and background causes high intraclass variance. Ability to exploit domain knowledge and cross-category patterns and similarities becomes increasingly important.

There exist a variety of datasets related to recognition of tightly-related categories, including Oxford Flowers 102 [49], UIUC Birds [41], and STONEFLY9 [46]. While these works represent progress, they still have shortcomings in scaling to large numbers of categories, applying to other types of object domains, or achieving performance levels that are good enough for real-world applications. Perhaps most similar in spirit to our work is the Botanist’s Field Guide [3], a system for plant species recognition with hundreds of categories and tens of thousands of images. One key difference is that their system is intended primarily for experts, and requires plant leaves to be

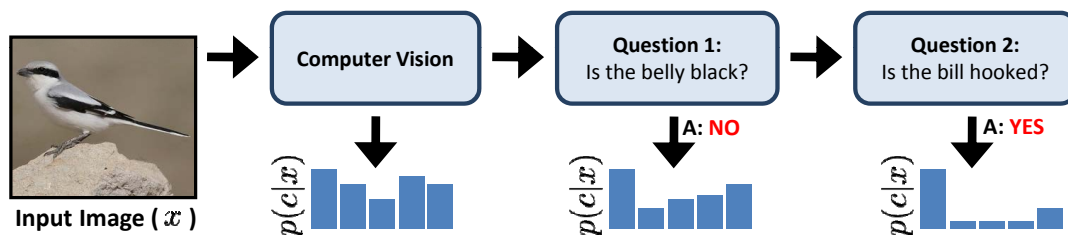


Figure 3.4: Visualization of the basic algorithm flow. The system poses questions to the user, which along with computer vision, incrementally refine the probability distribution over classes.

photographed in a controlled manner at training and test time, making segmentation and pose normalization possible. In contrast, all of our training and testing images are obtained from Flickr in unconstrained settings (see Fig. 3.5), and the system is intended to be used by lay people.

There exists a multitude of different areas in computer science that interleave vision, learning, or other processing with human input. Relevance feedback [92] is a method for interactive image retrieval, in which users mark the relevance of image search results, which are in turn used to create a refined search query. Active learning algorithms [68, 37, 34] interleave training a classifier with asking users to label examples, where the objective is to minimize the total number of labeling tasks. Our objectives are somewhat similar, except that we are querying information at runtime rather than training time. Expert systems [48, 4] involve construction of a knowledge base and inference rules that can help non-experts solve a problem. Our approach differs due to the added ability to observe image pixels as an additional source of information. Computationally, our method also has similarities to algorithms based on information gain, entropy calculation, and decision trees [70, 56, 16].

Finally, a lot of progress has been made on trying to scale object recognition to large numbers of categories. Such approaches include using class taxonomies [65, 32], feature sharing [69], error correcting output codes (ECOC) [20], and attribute based classification methods [40, 26, 39]. All of these methods could be easily plugged into our framework to incorporate user interaction.

3.1.3 Methods and Algorithm Description

Given an image x , our goal is to determine the true object class $c \in \{1 \dots C\}$ by posing questions based on visual properties that are easy for the user to answer (see Fig. 3.2). At each step, we aim to exploit the visual content of the image and the current history of question responses to intelligently select the next question. The basic algorithm flow is summarized in Fig. 3.4.

Let $\mathcal{Q} = \{q_1 \dots q_n\}$ be a set of possible questions (*e.g.*, IsRed?, HasStripes?, *etc.*), and \mathcal{A}_i be the set of possible answers to q_i . The user's answer is some random variable $a_i \in \mathcal{A}_i$. We also allow users to qualify each response with a confidence value $r_i \in \mathcal{V}$, (*e.g.*, $\mathcal{V} = \{\text{Guessing, Probably, Definitely}\}$). The user's response is then a pair of random variables $u_i = (a_i, r_i)$.

At each time step t , we select a question $q_{j(t)}$ to pose to the user, where $j(t) \in 1 \dots n$. Let $j \in \{1 \dots n\}^T$ be an array of T indices to questions we will ask the user. $U^{t-1} = \{u_{j(1)} \dots u_{j(t-1)}\}$ is the set of responses obtained by time step $t - 1$. We use maximum information gain as the criterion to select $q_{j(t)}$. Information gain is widely used in decision trees (*e.g.* [56]) and can be computed from an estimate of $p(c|x, U^{t-1})$.

We define $I(c; u_i|x, U^{t-1})$, the expected information gain of posing the additional question q_i , as follows:

$$\begin{aligned} I(c; u_i|x, U^{t-1}) &= \mathbb{E}_u [\text{KL}(p(c|x, u_i \cup U^{t-1}) \parallel p(c|x, U^{t-1}))] & (3.1) \\ &= \sum_{u_i \in \mathcal{A}_i \times \mathcal{V}} p(u_i|x, U^{t-1}) (\text{H}(c|x, u_i \cup U^{t-1}) - \text{H}(c|x, U^{t-1})) & (3.2) \end{aligned}$$

where $\text{H}(c|x, U^{t-1})$ is the entropy of $p(c|x, U^{t-1})$

$$\text{H}(c|x, U^{t-1}) = - \sum_{c=1}^C p(c|x, U^{t-1}) \log p(c|x, U^{t-1}) \quad (3.3)$$

The general algorithm for interactive object recognition is shown in Algorithm 1. In the next sections, we describe in greater detail methods for modeling user responses and different methods for incorporating computer vision algorithms, which correspond to different ways to estimate $p(c|x, U^{t-1})$.

Algorithm 1 Visual 20 Questions Game

- 1: $U^0 \leftarrow \emptyset$
 - 2: **for** $t = 1$ to 20 **do**
 - 3: $j(t) = \max_k I(c; u_k|x, U^{t-1})$
 - 4: Ask user question $q_{j(t)}$, and $U^t \leftarrow U^{t-1} \cup u_{j(t)}$.
 - 5: **end for**
 - 6: Return class $c^* = \max_c p(c|x, U^t)$
-

Incorporating Computer Vision

When no computer vision is involved it is possible to pre-compute a decision tree that defines which question to ask for every possible sequence of user responses. With computer vision in the loop, however, the best questions depend dynamically on the contents of the image.

In this section, we propose a simple framework for incorporating any multi-class object recognition algorithm that produces a probabilistic output over classes. We can compute $p(c|x, U)$, where U is any arbitrary sequence of responses, as follows:

$$p(c|x, U) = \frac{p(U|c, x)p(c|x)}{Z} = \frac{p(U|c)p(c|x)}{Z} \quad (3.4)$$

where $Z = \sum_c p(U|c)p(c|x)$. Here, we make the assumption that $p(U|c, x) = p(U|c)$; effectively this assumes that the types of noise or randomness that we see in user responses is class-dependent and not image-dependent. We can still accommodate variation in responses due to user error, subjectivity, external factors, and intraclass variance; however we throw away some image-related information (for example, we lose ability to model a change in the distribution of user responses as a result of a computer-vision-based estimate of object pose).

In terms of computation, we estimate $p(c|x)$ using a classifier trained offline (more details in Section 3.1.5). Upon receiving an image, we run the classifier once at the beginning of the process, and incrementally update $p(c|x, U)$ by gathering more answers to questions from the user. One could imagine a system where a learning algorithm is invoked several times during the process; as categories are weeded out by answers, the system would use a more tuned classifier to update the estimate of $p(c|x)$.

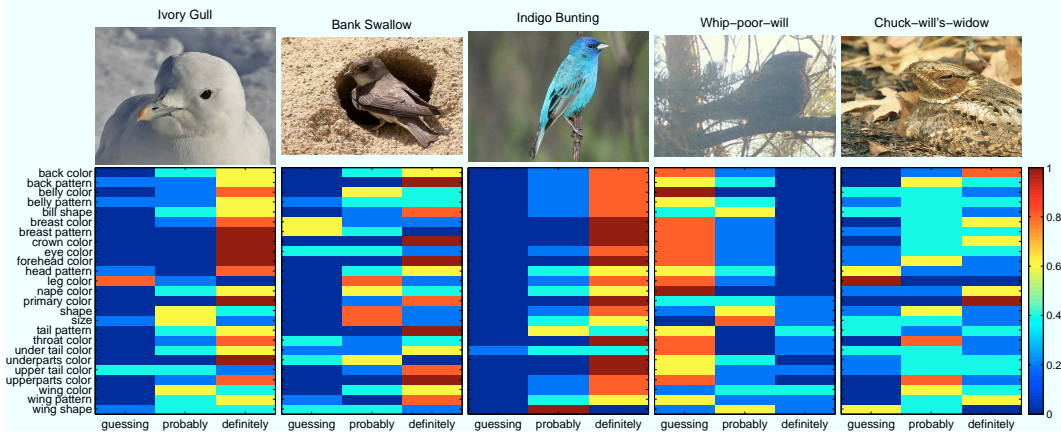


Figure 3.5: Examples of user responses for each of the 25 attributes. The distribution over $\{Guessing, Probably, Definitely\}$ is color coded with blue denoting 0% and red denoting 100% of the five answers per image attribute pair.

However, our preliminary experiments with such methods did not show an advantage². Note that when no computer vision is involved, we simply replace $p(c|x)$ with a prior $p(c)$.

Modeling User Responses

Recall that for each question we may also ask a corresponding confidence value from the user, which may be necessary when an attribute cannot be determined (for example, when the associated part(s) are not visible). We assume that the questions are answered independently given the category:

$$p(U^{t-1}|c) = \prod_i^{t-1} p(u_i|c) \quad (3.5)$$

The same assumption allows us to express $p(u_i|x, U^{t-1})$ in Equation 3.2 as

$$p(u_i|x, U^{t-1}) = \sum_{c=1}^C p(u_i|c)p(c|x, U^{t-1}) \quad (3.6)$$

It may also be possible to use a more sophisticated model in which we estimate a full joint distribution for $p(U^{t-1}|c)$; in our preliminary experiments this approach did not work well due to insufficient training data.

²See (<http://www.vision.caltech.edu/visipedia/birds200.html>) for more details.

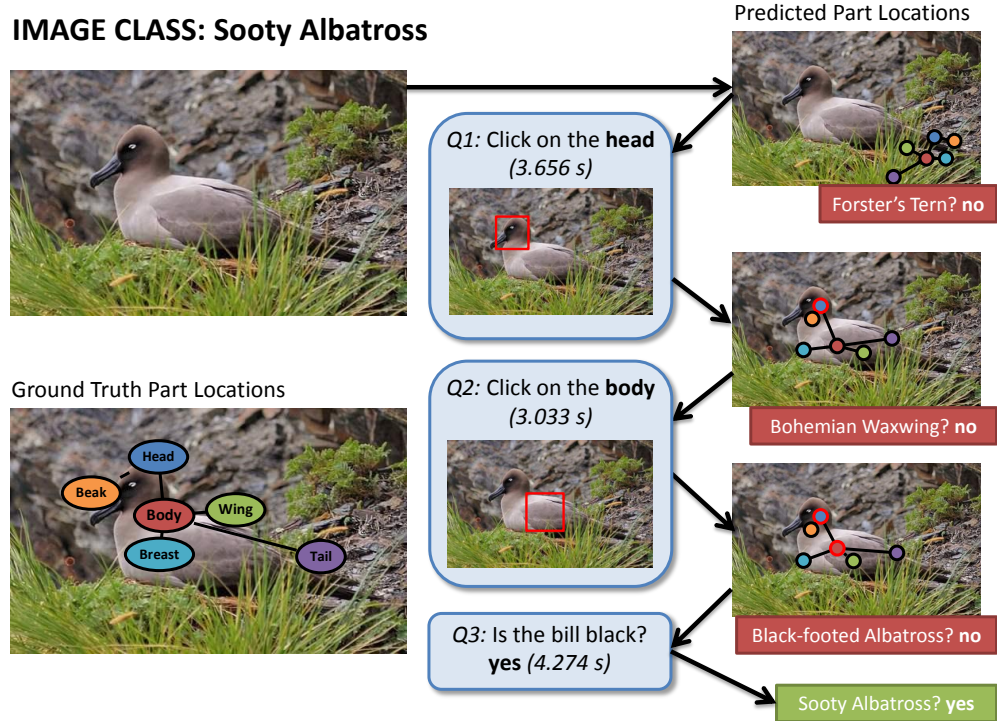


Figure 3.6: Extension to Part-Based Methods: Our system can query the user for input in the form of binary attribute questions or part clicks. In this illustrative example, the system provides an estimate for the pose and part locations of the object at each stage. Given a user-clicked location of a part, the probability distributions for locations of the other parts in each pose will adjust accordingly. The rightmost column depicts the maximum likelihood estimate for part locations.

To compute $p(u_i|c) = p(a_i, r_i|c) = p(a_i|r_i, c)p(r_i|c)$, we assume that $p(r_i|c) = p(r_i)$. Next, we compute each $p(a_i|r_i, c)$ as the posterior of a multinomial distribution with Dirichlet prior $\text{Dir}(\alpha_r p(a_i|r_i) + \alpha_c p(a_i|c))$, where α_r and α_c are constants, $p(a_i|r_i)$ is a global attribute prior, and $p(a_i|c)$ is estimated by pooling together certainty labels. In practice, we use a larger prior term for *Guessing* than *Definitely*, $\alpha_{guess} > \alpha_{def}$, which effectively down weights the importance of any response with certainty level *Guessing*.

3.1.4 Extension to Part-Based Models

In this section, we add a few extensions to the above algorithms and models: 1) we handle localized computer vision algorithms, introducing fast algorithms for multi-

class classification and pose registration using shared part models, 2) handle different heterogeneous types of user input, including part click locations and multiple choice questions, 3) introduce algorithms for predicting the informativeness a part click questions.

Consider for example different types of human annotation tasks in the domain of bird species recognition. For the task “*Click on the beak,*” the location a human user clicks is a noisy representation of the ground truth location of the beak. It may not in isolation solve any single recognition task; however, it provides information that is useful to a machine vision algorithm for localizing other parts of the bird, measuring attributes (*e.g.* cone-shaped), recognizing actions (*e.g.* eating or flying), and ultimately recognizing the bird species. The answer to the question “*Is the belly striped?*” similarly provides information towards recognizing a variety of bird species. Each type of annotation takes a different amount of human time to complete and provides varying amounts of information.

Problem Definition and Notation

Given an image x , our goal is to predict an object class from a set of C possible classes (*e.g.* Myrtle Warbler, Blue Jay, Indigo Bunting) within a common basic-level category (*e.g.* Birds). We assume that the C classes fall within a reasonably homogeneous basic-level category such as birds that can be represented using a common vocabulary of P parts (*e.g.* head, belly, wing), and A attributes (*e.g.* cone-shaped beak, white belly, striped breast). We use a class-attribute model based on the direct-attribute model of Lampert et al. [40], where each class $c \in 1 \dots C$ is represented using a unique, deterministic vector of attribute memberships $\mathbf{a}^c = [a_1^c \dots a_A^c]$, $a_i^c \in 0, 1$. We extend this model to include part localized attributes, such that each attribute $a \in 1 \dots A$ can optionally be associated with a part $\text{part}(a) \in 1 \dots P$ (*e.g.* the attributes *white belly* and *striped belly* are both associated with the part belly). In this case, we express the set of all ground truth part locations for a particular object as $\Theta = \{\theta_1 \dots \theta_P\}$, where the location θ_p of a particular part p is represented as an x_p, y_p image location, a scale s_p , and an aspect v_p (*e.g.* side view left, side view right, frontal view, not visible, *etc.*):

$$\theta_p = \{x_p, y_p, s_p, v_p\}. \quad (3.7)$$

Note that the special aspect *not visible* is used to handle parts that are occluded or self-occluded.

We can optionally combine our computer vision algorithms with human input, by intelligently querying user input at runtime. A human is capable of providing two types of user input which indirectly provide information relevant for predicting the object's class: mouse click locations $\tilde{\theta}_p$ and attribute question answers \tilde{a}_i . The random variable $\tilde{\theta}_p$ represents a user's input of the part location θ_p , which may differ from user to user due to both clicking inaccuracies and subjective differences in human perception (Figure 7.1). Similarly, \tilde{a}_i is a random variable defining a user's perception of the attribute value a_i .

We assume a pool of $A + P$ possible questions that can be posed to a human user $\mathcal{Q} = \{q_1 \dots q_A, q_{A+1} \dots q_{A+P}\}$, where the first A questions query \tilde{a}_i and the remaining P questions query $\tilde{\theta}_p$. Let \mathcal{A}_j be the set of possible answers to question q_j . At each time step t , our algorithm considers the visual content of the image and the current history of question responses to estimate a distribution over the location of each part, predict the probability of each class, and intelligently select the next question to ask $q_{j(t)}$. A user provides the response $u_{j(t)}$ to a question $q_{j(t)}$, which is the value of $\tilde{\theta}_p$ or \tilde{a}_i for part location or attribute questions, respectively. The set of all user responses up to timestep t is denoted by the symbol $U^t = \{u_{j(1)} \dots u_{j(t)}\}$. We assume that the user is consistent in answering questions and therefore the same question is never asked twice.

Probabilistic Model

Our probabilistic model incorporating both computer vision and human user responses is summarized in Figure 3.7(b). Our goal is to estimate the probability of each class given an arbitrary collection of user responses U^t and observed image pixels x :

$$p(c|U^t, x) = \frac{p(\mathbf{a}^c, U^t|x)}{\sum_c p(\mathbf{a}^c, U^t|x)}, \quad (3.8)$$

which follows from the assumption of unique, class-deterministic attribute memberships \mathbf{a}^c [40]. We can incorporate localization information Θ into the model by integrating over all possible assignments to part locations

$$p(\mathbf{a}^c, U^t|x) = \int_{\Theta} p(\mathbf{a}^c, U^t, \Theta|x) d\Theta. \quad (3.9)$$

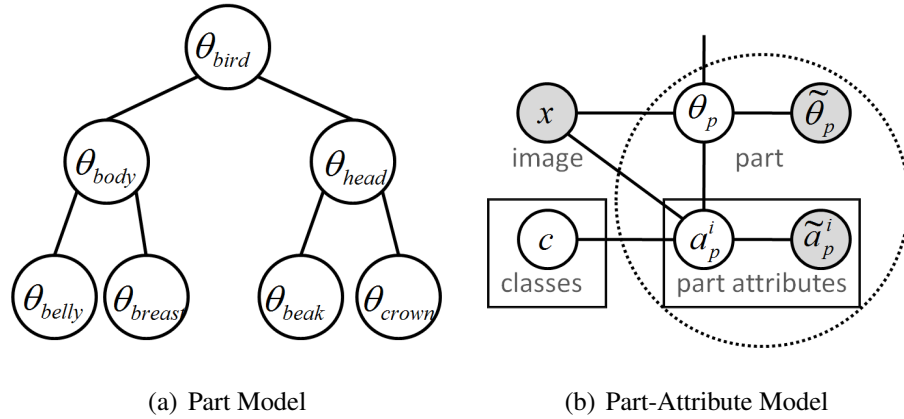


Figure 3.7: Probabilistic Model. 3.7(a): The spatial relationship between parts has a hierarchical independence structure. 3.7(b): Our model employs attribute estimators, where part variables θ_p are connected using the hierarchical model shown in 3.7(a).

We can write out each component of Eq 3.9 as

$$p(\mathbf{a}^c, U^t, \Theta|x) = p(\mathbf{a}^c|\Theta, x)p(\Theta|x)p(U^t|\mathbf{a}^c, \Theta, x) \quad (3.10)$$

where $p(\mathbf{a}^c|\Theta, x)$ is the response of a set of attribute detectors evaluated at locations Θ , $p(\Theta|x)$ is the response of a part-based detector, and $p(U^t|\mathbf{a}^c, \Theta, x)$ models the way users answer questions. We describe each of these probability distributions in Sections 3.1.4, 3.1.4, and 3.1.3 respectively and describe inference procedures for evaluating Eq 3.9 efficiently in Section 3.1.4.

Computer Vision Model

As described in Eq 3.10, we require two basic types of computer vision algorithms: one that estimates attribute probabilities $p(\mathbf{a}^c|\Theta, x)$ on a particular set of predicted part locations Θ , and another that estimates part location probabilities $p(\Theta|x)$.

Attribute Detection

Using the independence assumptions depicted in Figure 3.7(b), we can write the probability

$$p(\mathbf{a}^c|\Theta, x) = \prod_{a_i^c \in \mathbf{a}^c} p(a_i^c|\theta_{\text{part}(a_i)}, x). \quad (3.11)$$

Given a training set with labeled part locations $\theta_{\text{part}(a_i)}$, one can use standard computer vision techniques to learn an estimator for each $p(a_i|\theta_{\text{part}(a_i)}, x)$. In practice, we train a separate binary classifier for each attribute, extracting localized features from the ground truth location $\theta_{\text{part}(a_i)}$. As in [40], we convert attribute classification scores $z_i = f_a(x; \text{part}(a_i))$ to probabilities by fitting a sigmoid function $\sigma(\gamma_a z_i)$ and learning the sigmoid parameter γ_a using cross-validation. When $v_{\text{part}(a_i)} = \textit{not visible}$, we assume the attribute detection score is zero.

Part Detection

We use a pictorial structure to model part relationships (see Figure 3.7(a)), where parts are arranged in a tree-structured graph $T = (V, E)$. A full description of the model is contained in Section 7.2. We review the basic terminology here. We model the detection score $g(x; \Theta)$ as a sum over unary and pairwise potentials $\log(p(\Theta|x)) \propto g(x; \Theta)$ with

$$g(x; \Theta) = \sum_{p=1}^P \psi(x; \theta_p) + \sum_{(p,q) \in E} \lambda(\theta_p, \theta_q) \quad (3.12)$$

where each unary potential $\psi(x; \theta_p)$ is the response of a sliding window detector, and each pairwise score $\lambda(\theta_p, \theta_q)$ encodes a likelihood over the relative displacement between adjacent parts. We use the same learning algorithms and parametrization of each term in Eq 3.12 as in [88]. Here, parts and aspects are semantically defined, multiple aspects are handled using mixture models, and weight parameters for appearance and spatial terms are learned jointly using a structured SVM [71]. After training, we convert detection scores to probabilities $p(\Theta|x) \propto \exp(\gamma g(x; \Theta))$, where γ is a scaling parameter that is learned using cross-validation.

User Model

Readers interested in a computer-vision-only system with no human-in-the-loop can skip to Section 3.1.4. We assume that the probability of a set of user responses U^t can be expressed in terms of user responses that pertain to part click locations $U_{\Theta}^t \subseteq U^t$ and user responses that pertain to attribute questions $U_a^t \subseteq U^t$. We assume a user's

perception of the location of a part $\tilde{\theta}_p$ depends only on the ground truth location of that part θ_p , and a user’s perception of an attribute \tilde{a}_i depends only on the ground truth attribute a_i^c :

$$p(U^t | \mathbf{a}^c, \Theta, x) = \left(\prod_{p \in U_{\Theta}^t} p(\tilde{\theta}_p | \theta_p) \right) \left(\prod_{\tilde{a}_i \in U_a^t} p(\tilde{a}_i | a_i^c) \right). \quad (3.13)$$

We describe our methods for estimating $p(\tilde{\theta}_p | \theta_p)$ and $p(\tilde{a}_i | a_i^c)$ in the next two sections.

Modeling User Click Responses

Our interface for collecting part locations is shown in Figure 7.1. We represent a user click response as a triplet $\tilde{\theta}_p = \{\tilde{x}_p, \tilde{y}_p, \tilde{v}_p\}$, where $(\tilde{x}_p, \tilde{y}_p)$ is a point that the user clicks with the mouse and $\tilde{v}_p \in \{\text{visible}, \text{not visible}\}$ is a binary variable indicating presence/absence of the part.

Note that the user click response $\tilde{\theta}_p$ models only part location and visibility, whereas the true part location θ_p also includes scale and aspect. This is done in order to keep the user interface as intuitive as possible. On the other hand, incorporating scale and aspect in the true model is extremely important – the relative offsets and visibility of parts in *left side view* and *right side view* will be dramatically different. We model a distribution over user click responses as

$$p(\tilde{\theta}_p | \theta_p) = p(\tilde{x}_p, \tilde{y}_p | x_p, y_p, s_p) p(\tilde{v}_p | v_p) \quad (3.14)$$

where the relative part click locations are Gaussian distributed

$$\left(\frac{\tilde{x}_p - x_p}{s_p}, \frac{\tilde{y}_p - y_p}{s_p} \right) \sim \mathcal{N}(\tilde{\mu}_p, \tilde{\sigma}_p^2) \quad (3.15)$$

and each $p(\tilde{v}_p | v_p)$ is a separate binomial distribution for each possible value of v_p . The parameters of these distributions are estimated using a training set of pairs $(\theta_p, \tilde{\theta}_p)$. This model of user click responses results in a simple, intuitive user interface and still allows for a sophisticated and computationally efficient model of part localization (Section 3.1.4).

Attribute Question Responses

We use a model of attribute user responses similar to [10]. We estimate each $p(\tilde{a}_i | a_i)$ as a binomial distribution, with parameters learned using a training set of user

attribute responses collected from MTurk. As in [10], we allow users to qualify their responses with a certainty parameter *guessing*, *probably*, or *definitely*, and we incorporate a Beta prior to improve robustness when training data is sparse.

Inference

We describe the inference procedure for estimating per-class probabilities $p(c|U^t, x)$ (Eq 3.8), which involves evaluating $\int_{\Theta} p(\mathbf{a}^c, U^t, \Theta|x)d\Theta$. While this initially seems very difficult, we note that all user responses \tilde{a}_p^i and $\tilde{\theta}_p$ are observed values pertaining only to a single part, and attributes a^c are deterministic when conditioned on a particular choice of class c . If we run inference separately for each class c , all components of Eqs 3.11 and 3.13 can simply be mapped into the unary potential for a particular part. Evaluating Eq 3.8 exactly is computationally similar to evaluating a separate pictorial structure inference problem for each class.

On the other hand, when C is large, running C inference problems can be inefficient. In practice, we use a faster procedure which approximates the integral in Eq 3.9 as a sum over K strategically chosen sample points:

$$\begin{aligned} & \int_{\Theta} p(\mathbf{a}^c, U^t, \Theta|x)d\Theta \\ & \approx \sum_{k=1}^K p(U^t|\mathbf{a}^c, \Theta_k^t, x)p(\mathbf{a}^c|\Theta_k^t, x)p(\Theta_k^t|x) \\ & = p(U_a^t|\mathbf{a}^c) \sum_{k=1}^K p(\mathbf{a}^c|\Theta_k^t, x)p(U_{\Theta}^t|\Theta_k^t, x)p(\Theta_k^t|x). \end{aligned} \quad (3.16)$$

We select the sample set $\Theta_1^t \dots \Theta_K^t$ as the set of all local maxima in the probability distribution $p(U_{\Theta}^t|\Theta)p(\Theta|x)$. The set of local maxima can be found using standard methods for maximum likelihood inference on pictorial structures and then running non-maximal suppression, where probabilities for each user click response $p(\tilde{\theta}_p|\theta_p)$ are first mapped into a unary potential $\psi(x; \theta_p, \tilde{\theta}_p)$ (see Eq 3.12)

$$\psi(x; \theta_p, \tilde{\theta}_p) = \psi(x; \theta_p) + \log p(\tilde{\theta}_p|\theta_p). \quad (3.17)$$

The inference step takes time linear in the number of parts and pixel locations³ and is

³Maximum likelihood inference involves a bottom-up traversal of T , doing a distance transform operation [29] for each part in the tree (takes time $O(n)$ time in the number of pixels).

efficient enough to run in a fraction of a second with 13 parts, 11 aspects, and 4 scales. Inference is re-run each time we obtain a new user click response $\tilde{\theta}_p$, resulting in a new set of samples. Sampling assignments to part locations ensures that attribute detectors only have to be evaluated on K candidate assignments to part locations; this opens the door for more expensive categorization algorithms (such as kernelized methods) that do not have to be run in a sliding window fashion.

Selecting the Next Question

In this section, we introduce a common framework for predicting the informativeness of different heterogeneous types of user input (including binary questions and mouse click responses) that takes into account the expected level of human error, informativeness in a multitask setting, expected annotation time, and spatial relationships between different parts. Our method extends the expected information gain criterion described in [10].

Let $\text{IG}_t(q_j)$ be the expected information gain $\text{IG}(c; u_j|x, U^t)$ from asking a new question q_j :

$$\text{IG}_t(q_j) = \sum_{u_j \in \mathcal{A}_j} p(u_j|x, U^t) (\text{H}(U^t, u_j) - \text{H}(U^t)) \quad (3.18)$$

$$\text{H}(U^t) = - \sum_c p(c|x, U^t) \log p(c|x, U^t) \quad (3.19)$$

where $\text{H}(U^t)$ is shorthand for the conditional class entropy $\text{H}(c|x, U^t)$. Evaluating Eq 3.18 involves considering every possible user-supplied answer $u_j \in \mathcal{A}_j$ to that question, and recomputing class probabilities $p(c|x, U^t, u_j)$. For yes/no attribute questions (querying a variable \tilde{a}_i), this is computationally efficient because the number of possible answers is only two, and attribute response probabilities $p(U_a^t|\mathbf{a}^c)$ are assumed to be independent from ground truth part locations (see Eq 3.16).

Predicting Informativeness of Mouse Clicks In contrast, for part click questions the number of possible answers to each question is equal to the number of pixel locations, and computing class probabilities requires solving a new inference problem (Section 3.1.4) for each such location, which quickly becomes computationally intractible.

We use a similar approximation to the random sampling method described in Section 3.1.4. For a given part location question q_j , we wish to compute the expected entropy:

$$E_{\tilde{\theta}_p} [\text{H}(U^t, \tilde{\theta}_p)] = \sum_{\tilde{\theta}_p} p(\tilde{\theta}_p | x, U^t) \text{H}(U^t, \tilde{\theta}_p). \quad (3.20)$$

This can be done by drawing K samples $\tilde{\theta}_{p1}^t \dots \tilde{\theta}_{pK}^t$ from the distribution $p(\tilde{\theta}_p | x, U^t)$, then computing expected entropy

$$E_{\tilde{\theta}_p} [\text{H}(U^t, \tilde{\theta}_p)] \approx \quad (3.21)$$

$$- \sum_{k=1}^K p(\tilde{\theta}_p | x, U^t) \sum_c p(c | x, U^t, \tilde{\theta}_{pk}^t) \log p(c | x, U^t, \tilde{\theta}_{pk}^t).$$

In this case, each sample $\tilde{\theta}_{pk}^t$ is extracted from a sample Θ_k^t (Section 3.1.4) and each $p(c | x, U^t, \tilde{\theta}_{pk}^t)$ is approximated as a weighted average over samples $\Theta_1^t \dots \Theta_K^t$. The full question selection procedure is fast enough to run in a fraction of a second on a single CPU core when using 13 click questions and 312 binary questions.

Selecting Questions By Time

The expected information gain criterion (Eq 3.18) attempts to minimize the total number of questions asked. This is suboptimal as different types of questions tend to take more time to answer than others (*e.g.*, part click questions are usually faster than attribute questions). We include a simple adaptation that attempts to minimize the expected amount of human time spent. The information gain criterion $\text{IG}_t(q_j)$ encodes the expected number of bits of information gained by observing the random variable u_j . We assume that there is some unknown linear relationship between bits of information and reduction in human time. The best question to ask is then the one with the largest ratio of information gain relative to the expected time to answer it:

$$q_{j(t+1)}^* = \arg \max_{q_j} \frac{\text{IG}_t(q_j)}{\mathbb{E}[\text{time}(u_j)]} \quad (3.22)$$

where $\mathbb{E}[\text{time}(u_j)]$ is the expected amount of time required to answer a question q_j .

3.1.5 Datasets and Implementation Details

In this section we provide a brief overview of the datasets we used, methods used to construct visual questions, computer vision algorithms we tested, and parameter settings.

Birds-200 Dataset

Birds-200 is a dataset of 6033 images over 200 bird species, such as Myrtle Warblers, Pomarine Jaegers, and Black-footed Albatrosses – classes that cannot usually be identified by non-experts. In many cases, different bird species are nearly visually identical (see Fig. 3.11).

We assembled a set of 25 visual questions (list shown in Fig. 3.5), which encompass 288 binary attributes (*e.g.*, the question `HasBellyColor` can take on 15 different possible colors). The list of attributes was extracted from [whatbird.com](http://www.whatbird.com)⁴, a bird field guide website.

We collected “deterministic” class-attributes by parsing attributes from [whatbird.com](http://www.whatbird.com). Additionally, we collected data of how non-expert users respond to attribute questions via a Mechanical Turk interface. To minimize the effects of user subjectivity and error, our interface provides prototypical images of each possible attribute response. The reader is encouraged to look at the supplementary material for screenshots of the question answering user-interface and example images of the dataset.

Fig. 3.5 shows a visualization of the types of user response results we get on the Birds-200 dataset. It should be noted that the uncertainty of the user responses strongly correlates with the parts that are visible in an image as well as overall difficulty of the corresponding bird species.

When evaluating performance, test results are generated by randomly selecting a response returned by an MTurk user for the appropriate test image.

⁴<http://www.whatbird.com/>

Extended Birds-200 Dataset

For the part localized experiments, we extended the original CUB-200 dataset [86] to form CUB-200-2011 [79], which includes roughly 11,800 images, nearly double the previous total. Each image is annotated with 312 binary attribute labels and 15 part labels. We obtained a list of attributes from a bird field guide website [81] and selected the parts associated with those attributes for labeling. Five different MTurk workers provided part labels for each image by clicking on the image to designate the location or denoting part absence (Figure 7.1). One MTurk worker answered attribute questions for each image, specifying response certainty with options *guessing*, *probably*, and *definitely*. They were also given the option *not visible* if the associated part with the attribute was not present. At test time, we simulated user responses in a similar manner to [10], randomly selecting a stored response for each posed question. Instead of using bounding box annotations to crop objects, we used full uncropped images, resulting in a significantly more challenging dataset than CUB-200 [86].

Animals With Attributes

We also tested performance on the Animals With Attributes (AwA) [40], a dataset of 50 animal classes and 85 binary attributes. We consider this dataset less relevant than birds (because classes are recognizable by non-experts), and therefore do not focus as much on this dataset.

Implementation Details and Parameter Settings

Non-Localized Model: For both datasets, our computer vision algorithms are based on Andrea Vedaldi’s publicly available source code [72], which combines vector-quantized geometric blur and color/gray SIFT features using spatial pyramids, multiple kernel learning, and per-class 1-vs-all SVMs. We added features based on full image color histograms and vector-quantized color histograms. For each classifier we used Platt scaling [54] to learn parameters for $p(c|x)$ on a validation set. We used 15 training examples for each Birds-200 class and 30 training examples for each AwA class. Bird training and testing images are roughly cropped.

Additionally, we compare performance to a second computer vision algorithm based on attribute classifiers, which we train using the same features/training code, with positive and negative examples set using whatbird.com attribute labels. We combined attribute classifiers into per-class probabilities $p(c|x)$ using the method described in [40].

For estimating user response statistics on the Birds-200 dataset, we used $\alpha_{guess} = 64$, $\alpha_{prob} = 16$, $\alpha_{def} = 8$, and $\alpha_c = 8$ (see Section 3.1.3).

Part-Localized Model: For attribute detectors, we used simple linear classifiers based on histograms of vector-quantized SIFT and vector-quantized RGB features (each with 128 codewords) which were extracted from windows around the location of an associated part. We believe that significant improvements in classification performance could be gained by exploring more sophisticated features or learning algorithms.

As in [29], the unary scores of our part detector are implemented using HOG templates parametrized by a vector of linear appearance weights w_{v_p} for each part and aspect. The pairwise scores are quadratic functions over the displacement between (x_p, y_p) and (x_q, y_q) , parametrized by a vector of spatial weights w_{v_p, v_q} for each pose and pair of adjacent parts. For computational efficiency, we assume that the pose and scale parameters are defined on an object level, and thus inference simply involves running a separate sliding window detector for each scale and pose. The ground truth scale of each object is computed based on the size of the object’s bounding box.

Because our object parts are labeled only with visibility, we clustered images using k -means on the spatial x - and y - offsets of the part locations from their parent part locations, normalized with respect to image dimensions; this approach handles relative part locations in a manner most similar to how we model part relationships (Section 3.1.4). Examples of images grouped by their pose cluster are shown in Figure 7.5. Semantic labels were assigned post hoc by visual inspection. The clustering, while noisy, reveals some underlying pose information that can be discovered by part presence and locations.

3.1.6 Experiments

In this section, we provide experimental results and analysis of the hybrid-human computer classification paradigm. Due to space limitations, our discussion focuses on the Birds dataset. We include results (see Fig. 3.12) from which the user can verify that trends are similar on Birds-200 and AWA, and we include additional results on AWA in the supplementary material.

Measuring Performance

We use two main methodologies for measuring performance, which correspond to two different possible user-interfaces:

- **Method 1:** We ask the user exactly T questions, predict the class with highest probability, and measure the percent of the time that we are correct.
- **Method 2:** After asking each question, we present a small gallery of images of the highest probability class, and allow the user to stop the system early. We measure the average number of questions asked per test image.

For the second method, we assume that people are perfect verifiers, *e.g.*, they will stop the system if and only if they have been presented with the correct class. While this is not always possible in reality, there is some trade-off between classification accuracy and amount of human labor, and we believe that these two metrics collectively capture the most important considerations.

Results

In this section, we present our results and discuss some interesting trends toward understanding the visual 20 questions classification paradigm.

User Responses are Stochastic: In Fig. 3.8, we show the effects of different models of user responses without using any computer vision. When users are assumed to respond deterministically in accordance with the attributes from whatbird.com, performance rises quickly to 100% within 8 questions (roughly $\log_2(200)$). However, this

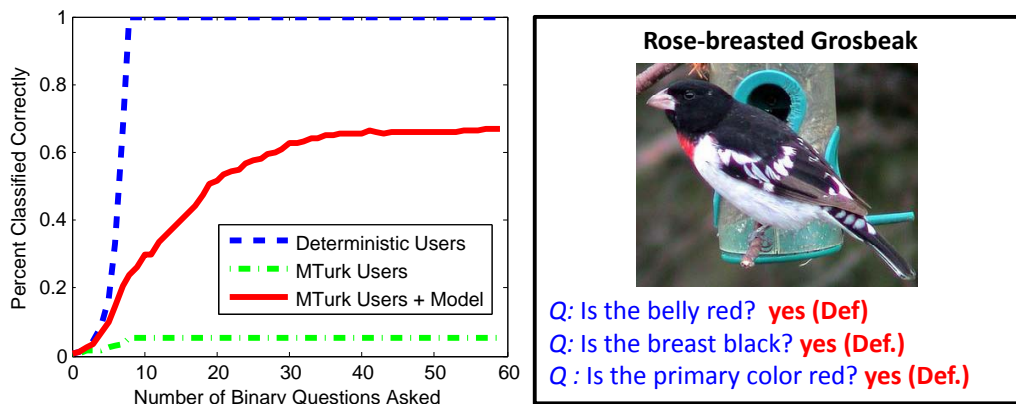


Figure 3.8: Different Models of User Responses: *Left:* Classification performance on Birds-200 (Method 1) without computer vision. Performance rises quickly (blue curve) if users respond deterministically according to whatbird.com attributes. MTurk users respond quite differently, resulting in low performance (green curve). A learned model of MTurk responses is much more robust (red curve). *Right:* A test image where users answer several questions incorrectly and our model still classifies the image correctly.

assumption is not realistic; when testing with responses from Mechanical Turk, performance saturates at around 5%. Low performance caused by subjective answers are unavoidable (*e.g.*, perception of the color brown vs. the color buff), and the probability of the correct class drops to zero after any inconsistent response. Although performance is 10 times better than random chance, it renders the system useless. This demonstrates a challenge for existing field guide websites. When our learned model of user responses (see Section 3.1.3) is incorporated, performance jumps to 66% due to the ability to tolerate a reasonable degree of error in user responses (see Fig. 3.8 for an example). Nevertheless, stochastic user responses increase the number of questions required to achieve a given accuracy level, and some images can never be classified correctly, even when asking all possible questions. In Section 3.1.6, we discuss the reasons why performance saturates at lower than 100% performance.

Computer Vision Reduces Manual Labor: The main benefit of computer vision occurs due to reduction in human labor (in terms of the number of questions a user has to answer). In Fig. 3.9, we see that computer vision reduces the average number of yes/no questions needed to identify the true bird species from 11.11 to 6.43 using responses from MTurk users. Without computer vision, the distribution of question counts is bell-

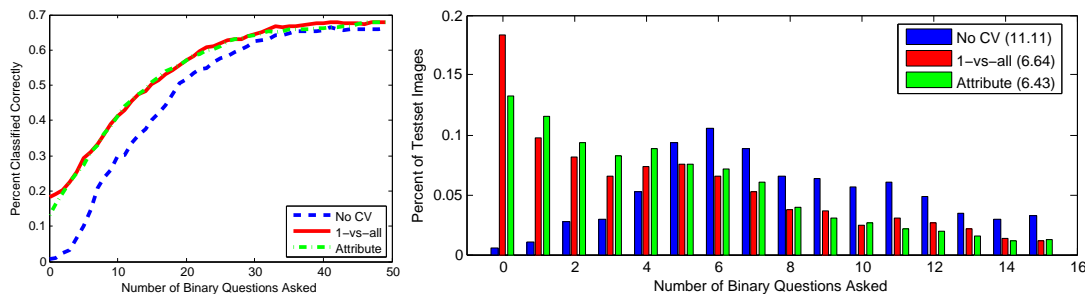


Figure 3.9: Performance on Birds-200 when using computer vision: Left Plot: comparison of classification accuracy (Method 1) with and without computer vision when using MTurk user responses. Two different computer vision algorithms are shown, one based on per-class 1-vs-all classifiers and another based on attribute classifiers. Right plot: the number of questions needed to identify the true class (Method 2) drops from 11.11 to 6.43 on average when incorporating computer vision.

shaped and centered around 6 questions. When computer vision is incorporated, the distribution peaks at 0 questions but is more heavy-tailed, which suggests that computer vision algorithms are often good at recognizing the “easy” test examples (examples that are sufficiently similar to the training data), but provide diminishing returns toward classifying the harder examples that are not sufficiently similar to training data. As a result, computer vision is more effective at reducing the average amount of time than reducing the time spent on the most difficult images.

User Responses Drive Up Performance: An alternative way of interpreting the results is that user responses drive up the accuracy of computer vision algorithms. In Fig. 3.9, we see that user responses improve overall performance from $\approx 19\%$ (using 0 questions) to $\approx 66\%$.

Computer Vision Improves Overall Performance: Even when users answer all questions, performance saturates at a higher level when using computer vision ($\approx 69\%$ vs. $\approx 66\%$, see Fig. 3.9). The left image in Fig. 3.10 shows an example of an image classified correctly using computer vision, which is not classified correctly without computer vision, even after asking 60 questions. In this example, some visually salient features like the long neck are not captured in our list of visual attribute questions. The features used by our vision algorithms also capture other cues (such as global texture statistics) that are not well-represented in our list of attributes (which capture mostly color and

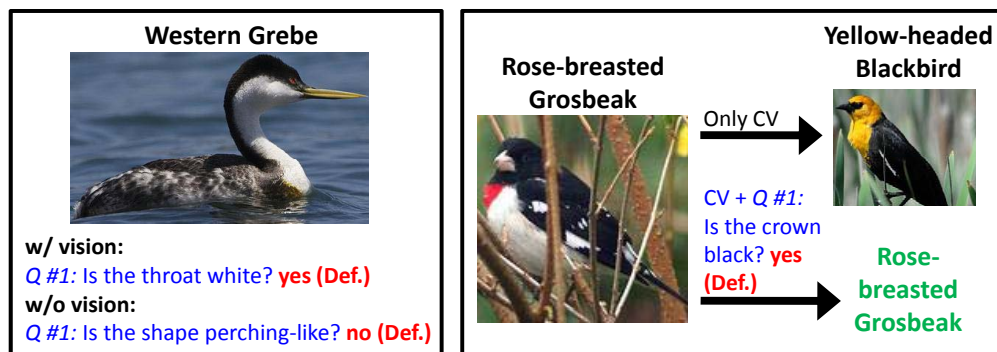


Figure 3.10: Examples where computer vision and user responses work together: *Left:* An image that is only classified correctly when computer vision is incorporated. Additionally, the computer vision based method selects the question `HasThroatColorWhite`, a different and more relevant question than when vision is not used. In the right image, the user response to `HasCrownColorBlack` helps correct computer vision when its initial prediction is wrong.

part-localized patterns).

Different Questions Are Asked With and Without Computer Vision: In general, the information gain criterion favors questions that 1) can be answered reliably, and 2) split the set of possible classes roughly in half. Questions like `HasShapePerchingLike`, which divide the classes fairly evenly, and `HasUnderpartsColorYellow`, which tends to be answered reliably, are commonly chosen.

When computer vision is incorporated, the likelihood of classes change and different questions are selected. In the left image of Fig. 3.10, we see an example where a different question is asked with and without computer vision, which allows the system to find the correct class using one question.

Recognition is Not Always Successful: According to the Cornell Ornithology Website⁵, the four keys to bird species recognition are 1) size and shape, 2) color and pattern, 3) behavior, and 4) habitat. Bird species classification is a difficult problem and is not always possible using a single image. One potential advantage of the visual 20 questions paradigm is that other contextual sources of information such as behavior and habitat can easily be incorporated as additional questions.

Fig. 3.11 illustrates some example failures. The most common failure conditions

⁵<http://www.allaboutbirds.org/NetCommunity/page.aspx?pid=1053>

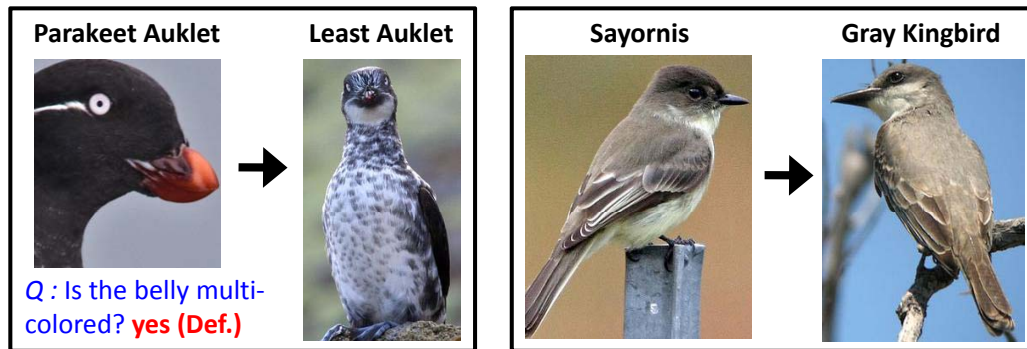


Figure 3.11: Images that are misclassified by our system: *Left:* The Parakeet Auklet image is misclassified due to a cropped image, which causes an incorrect answer to the belly pattern question (the Parakeet Auklet has a plain, white belly, see Fig. 3.3). *Right:* The Sayornis and Gray Kingbird are commonly confused due to visual similarity.

occur due to 1) classes that are nearly visually identical, 2) images of poor viewpoint or low resolution, such that some parts are not visible, 3) significant mistakes made by MTurkers, or 4) inadequacies in the set of attributes we used.

1-vs-all Vs. Attribute-Based Classification: In general, 1-vs-all classifiers slightly outperform attribute-based classifiers; however, they converge to similar performance as the number of question increases, as shown in Fig. 3.9 and 3.12. The features we use (kernelized and based on bag-of-words) may not be well suited to the types of attributes we are using, which tend to be localized and associated with a particular part. One potential advantage of attribute-based methods is computational scalability when the number of classes increases; whereas 1-vs-all methods always require C classifiers, the number of attribute classifiers can be varied in order to trade-off accuracy and computation time. The table below displays the average number of questions needed (Method 1) on the Birds dataset using different number of attribute classifiers (which were selected randomly):

200 (1-vs-all)	288 attr.	100 attr.	50 attr.	20 attr.	10 attr.
6.43	6.72	7.01	7.67	8.81	9.52

Question selection by time reduces human effort: By minimizing human effort with the time criterion, we are trading off between the expected information gain from a question response and the expected time to answer that question. Subsequently, we

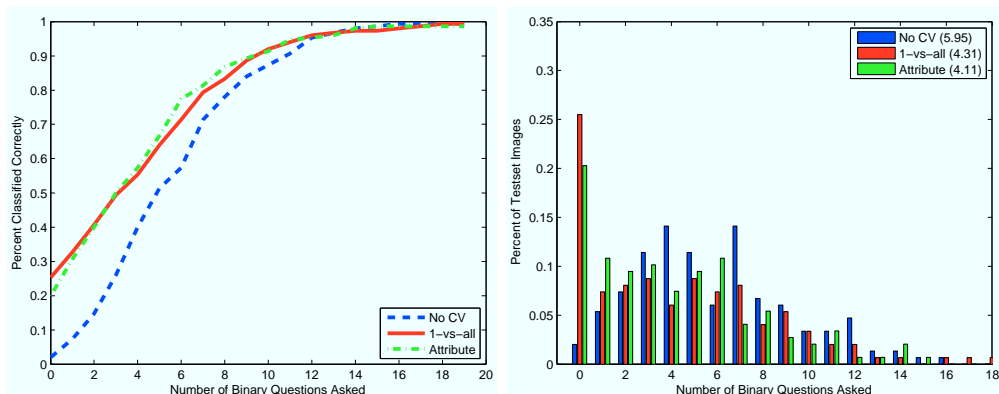


Figure 3.12: Performance on Animals With Attributes: Left Plot: Classification performance (Method 1), simulating user responses using soft class-attributes (see [40]). Right Plot: The required number of questions needed to identify the true class (Method 2) drops from 5.94 to 4.11 on average when incorporating computer vision.

are able to classify images in 36.6 seconds less on average using both binary and click questions than if we only take into account expected information gain; however, the margin in performance gain between using and not using click questions is reduced.

We note that the average time to answer a part click question is 3.01 ± 0.26 seconds, compared to 7.64 ± 5.38 seconds for an attribute question; in this respect, part questions are more likely to be asked first.

Part localization improves performance: There is a disparity in classification accuracy between evaluating attribute classifiers on ground truth locations (17.3%) versus predicted locations (10.3%); by using user responses to part click questions, we are able to overcome initial erroneous part detections and guide the system to the correct class. In Figure 3.13(a), we observe that by selecting the next question using our expected information gain criterion, average classification time using both types of user input versus only binary questions is reduced by 33.8 seconds on average. Compared to using no computer vision, we note an average reduction in human effort of over 40% (68.2 seconds).

Using the time criterion for selecting questions, the average classification time for a single image using both binary and click questions is 58.4 seconds. Asking binary questions only, the system takes an additional 20.4 seconds on average to correctly classify an image (Figure 3.13(b)). Using computer vision algorithms, we are able to

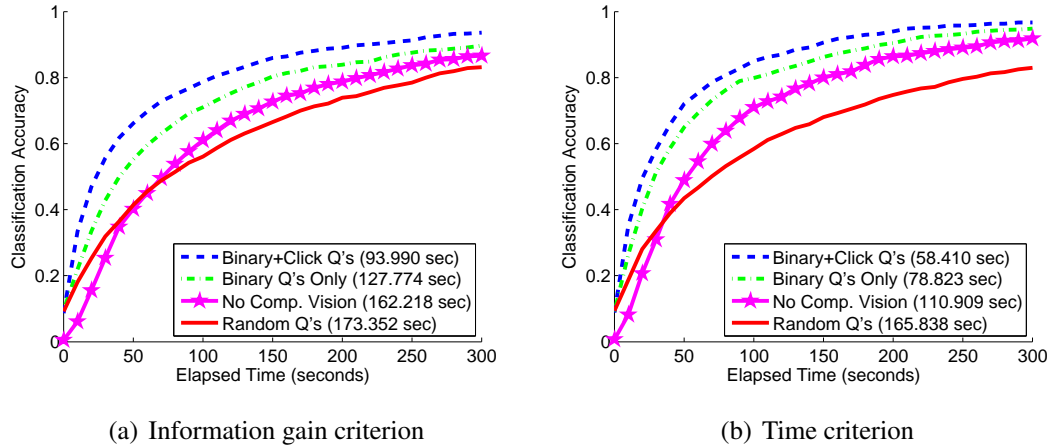


Figure 3.13: Classification accuracy as a function of time when 3.13(a) maximizing expected information gain; and 3.13(b) minimizing amount of human labor, measured in time. Performance is measured as the average number of seconds to correctly classify an image (described in Section 3.1.6).

consistently achieve higher average classification accuracy than using no computer vision at all, in the same period of time.

Qualitative Analysis of Part Click User Responses: Figure 3.14(a) presents an example in which the bird’s pose is estimated incorrectly. After posing one question and re-evaluating attribute detectors for updated part probability distributions, our model is able to correctly predict the class.

In Figure 3.14(b), we visualize the question-asking sequence and how the probability distribution of part locations over the image changes with user clicks. We note in Figure 3.14(c) that our pose clusters did not discover certain poses, especially frontal views, and the system is unable to estimate the pose with high certainty.

As previously discussed, part click questions take on average less time to answer. We observe that the system will tend to ask 2 or 3 part click questions near the beginning and then continue with primarily binary questions (*e.g.* Figure 3.14(d)). At this point, the remaining parts can often be inferred reliably through reasoning over the spatial model, and thus binary questions become more advantageous.

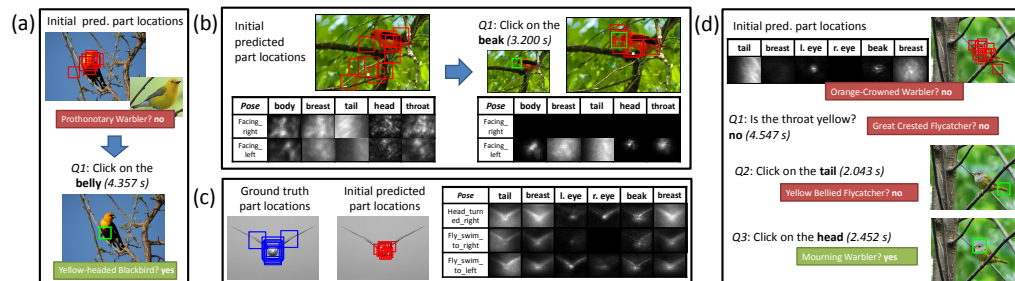


Figure 3.14: Four examples of the behavior of our system. 3.14(a): The system estimates the bird pose incorrectly but is able to localize the head and upper body region well, and the initial class prediction captures the color of the localized parts. The user’s response to the first system-selected part click question helps correct computer vision. 3.14(b): The bird is incorrectly detected, as shown in the probability maps displaying the likelihood of individual part locations for a subset of the possible poses (not visible to the user). The system selects “Click on the beak” as the first question to the user. After the user’s click, the other part location probabilities are updated and exhibit a shift towards improved localization and pose estimation. 3.14(c): Certain infrequent poses (e.g. frontal views) were not discovered by the initial off-line clustering (see Figure 7.5). The initial probability distributions of part locations over the image demonstrate the uncertainty in fitting the pose models. The system tends to fail on these unfamiliar poses. 3.14(d): The system will at times select both part click and binary questions to correctly classify images.

3.2 An Interactive Part Labeling Tool

In this section, we present an algorithm that computes and displays in realtime the maximum likelihood location of a deformable part model as the user drags different parts with the mouse, as visualized in Figure 3.15

3.2.1 Model and Notation

A full description of the model is contained in Section 7.2.3. We review the basic terminology here. Given an image x , let $\Theta = \theta_1 \dots \theta_P$ encode the position of each of P parts in the image. The location θ_p of a particular part p can be parameterized by an image location (x_p, y_p) , scale s_p , orientation r_p , and aspect v_p : $\theta_p = \{x_p, y_p, s_p, r_p, v_p\}$.

We assume a part tree model $T = (V, E)$ (see Figure 3.16(a)), such that the score $s(\Theta; x)$ of a particular part configuration Θ can be expressed as a sum over unary terms $\psi_p(\theta_p; x)$ for each part and pairwise terms $\lambda_{pq}(\theta_p, \theta_q)$ for each edge in the tree:

$$s(\Theta; x) = \sum_{p \in V} \psi_p(\theta_p; x) + \sum_{(p,q) \in E} \lambda_{pq}(\theta_p, \theta_q) \quad (3.23)$$

Here, $\psi_p(\theta_p; x)$ is a learned appearance score for part p (the response of a sliding window detector for part p) and $\lambda_{pq}(\theta_p, \theta_q)$ is a learned spatial score between pairs of parts. The maximum likelihood solution $\Theta^* = \max_{\Theta} s(\Theta; x)$ can be found efficiently using dynamic programming.

Our part detectors are based on sliding window HOG templates, and our spatial model is implemented as a quadratic function on the relative displacement between parts. The index v_p encodes the view/aspect of a part and is implemented using a mixture model, with different appearance templates and spatial parameters for each v_p [88].

3.2.2 Incorporating User Input

Let $U_t = \{\tilde{\theta}_{j(1)} \dots \tilde{\theta}_{j(t)}\}$ be a sequence of user input operations up to time step t . In this notation, $j(t)$ is the index of the part annotated by the user in time step t , and $\tilde{\theta}_{j(t)}$ is the user's label of the part location $\theta_{j(t)}$. If the user re-annotates the same part, it is assumed that the most recent annotation overrides all previous ones. The maximum

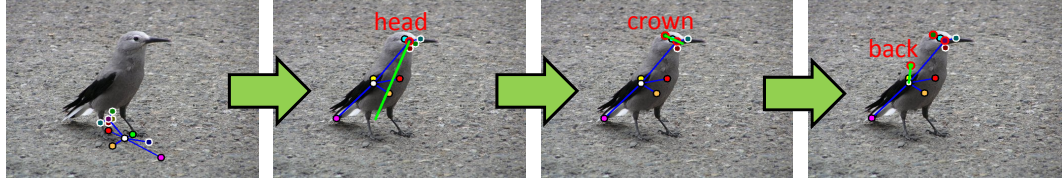


Figure 3.15: Visualization of Interactive Part Labeling Tool: The system displays in realtime the maximum likelihood location of all parts as the user drags different parts. The machine predicted bird location is initially entirely incorrect, but improves significantly after the user drags the location of the head. All 15 parts are correct after the user drags the head, crown, and back.

likelihood solution Θ_t^* that is consistent with U_t can be obtained by maximizing a modified score function $s^t(\Theta; x, U_t)$ which maps each user response $\tilde{\theta}_p$ into a unary potential $u_p(\theta_p; \tilde{\theta}_p)$:

$$s^t(\Theta; x, U_t) = s(\Theta; x) + \sum_{\tilde{\theta}_p \in U_t} u_p(\theta_p; \tilde{\theta}_p) \quad (3.24)$$

$$u_p(\theta_p; \tilde{\theta}_p) = \begin{cases} -\infty & \text{if } \theta_p \neq \tilde{\theta}_p \\ 0 & \text{otherwise} \end{cases} \quad (3.25)$$

Simple extensions include allowing imperfect user responses $u_p(\theta_p; \tilde{\theta}_p) \propto \log p(\theta_p | \tilde{\theta}_p)$, and partial annotations to a given part (e.g. the user labels x_p, y_p but not s_p, r_p, v_p).

3.2.3 Creating an Interactive User Interface

Dynamic programming is commonly used for maximum likelihood (ML) inference on pictorial structures with hierarchical structure. It computes cache tables which are indexable by pixel location and are accessed during a backtracking stage to extract the ML solution. In our case, interactively displaying the ML solution as the user drags the mouse simply involves indexing into a different starting pixel location during the backtracking stage, and therefore it is easily computable in realtime.

To make this work for our GUI, we require two changes to standard dynamic programming algorithms: 1) we run dynamic programming in both directions up and down the tree (this allows us to lookup the ML solution as the user drags any part as opposed to just the root of the part tree), and 2) we must update our cache tables

Algorithm 2 INTERACTIVEPARTLABELER

Input: An image x and model weights w
Output: Verified labels Θ_t^*

- 1: Compute part detection responses Ψ_p
 - 2: Precompute solution for any possible user response:
 - 3: Bottom-up traversal, evaluating Eq 3.28-3.29
 - 4: Top-down traversal, evaluating Eq 3.30-3.32
 - 5: **while** User unsatisfied with $\Theta_t^* = \max_{\theta_r} M_r^t[\theta_r]$ **do**
 - 6: As user drags $j(t)$, interactively show $M_{j(t)}^t[\tilde{\theta}_{j(t)}]$
 - 7: On mouse release, finalize solution $\tilde{\theta}_{j(t)}$:
 - 8: Update unary score $M_{j(t)}^{t+1}$ (Eq 3.33)
 - 9: Breadth first traversal from $j(t)$, evaluate Eq 3.34-3.37
 - 10: $t \leftarrow t + 1$
 - 11: **end while**
-

over time as we obtain additional user input (such that we can display the ML solution conditioned on all user annotations received so far). In the remainder of this section, we describe algorithms for implementing these two things efficiently. Due to space restrictions, discussion of the algorithm in this section is brief. We include more detailed derivation and proof of correctness in the supplementary material. The entire algorithm is summarized in Algorithm 2.

Let M_q^t denote an array storing the maximum likelihood solutions for part q at time step t after having received user annotations U_t . In our notation, $M_q^t[\theta_q]$ stores the score of the optimal solution conditioned on placing q at position θ_q :

$$\begin{aligned} M_p^t[\tilde{\theta}_p] &= \max_{\Theta} s^t(\Theta; x, U_t) \\ &\text{s.t. } \theta_p = \tilde{\theta}_p \end{aligned} \tag{3.26}$$

Our goal is to efficiently compute cache tables M_q^t for all parts q . We use the notation M_{qp}^t , where $p \in \text{neighbor}(q)$, to denote the table of sub-solutions over the subgraph which includes part q but deletes all parts and edges connected to q through p . For example, when $p = \text{parent}(q)$, M_{qp}^t stores the optimal solution over the sub-tree rooted at q . Let N_{pq}^t denote a similar concept which also factors in the spatial score between p

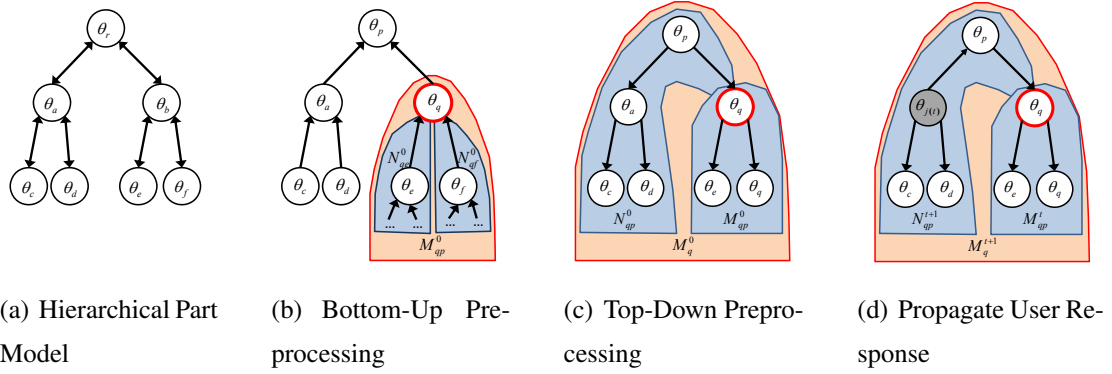


Figure 3.16: Visualization of Model and Algorithms: a) Object part location variables are assumed to have a hierarchical relationship. Dynamic programming is run in both directions up and down the tree. b-d) Visualization of the main algorithms used for the interactive interface. In each algorithm, part nodes are traversed in the order indicated by the arrows. When processing each node, solutions over the sub-graphs highlighted in blue are combined to form the solution highlighted in red. b) Using a standard dynamic programming algorithm, information is propagated from the child nodes up to the root using Eq 3.28-3.29, c) In a second top-down algorithm, information is passed from the root back down to the children using Eq 3.30-3.32, d) When a user response updates the variable θ_p , information is propagated using a breadth first traversal of the tree beginning at p using Eq 3.34-3.37.

and q :

$$N_{pq}^t[\tilde{\theta}_p] = \max_{\theta_q} (M_{qp}^t[\theta_q] + \lambda_{pq}(\theta_p, \theta_q)) \quad (3.27)$$

In other words, $N_{pq}^t[\tilde{\theta}_p]$ allows one to lookup the optimal location of θ_q when conditioned on a particular value $\tilde{\theta}_p$. Here, to avoid making the notation more complex, we have written N_{pq}^t as a score \max_{θ_q} ; however, in practice we must also store the location $\arg \max_{\theta_q}$, such that we can retrieve the solution later on during backtracking.

As in [28], we use a distance transform operation (which we denote by the operator \otimes) to densely compute N_{pq}^t in time linear in the number of pixel locations. Let Ψ_p and Λ_{pq} be shorthand for unary and pairwise score maps, such that $\Psi_p[\theta_p] = \psi_p(x, \theta_p)$ and $\Lambda_{pq}[\theta_q]$ is the cost associated with part q being at an offset of θ_q from p . In our notation, the standard dynamic programming algorithm for inference on pictorial structures

traverses the tree T bottom-up, using the recursive update step:

$$M_{qp}^0 = \Psi_q + \sum_{r \in \text{child}(q)} N_{qr}^0 \quad (3.28)$$

$$N_{pq}^0 = M_{qp}^0 \otimes \Lambda_{pq} \quad (3.29)$$

where Eq 3.28 is evaluated for all $q \in \text{child}(p)$. We run dynamic programming as our initial preprocessing step, then employ a second pass that processes each edge p, q in a top-down traversal of the tree:

$$M_{pq}^0 = M_p^0 - N_{pq}^0 \quad (3.30)$$

$$N_{qp}^0 = M_{pq}^0 \otimes \Lambda_{qp} \quad (3.31)$$

$$M_q^0 = M_{qp}^0 + N_{qp}^0 \quad (3.32)$$

This top-down pass computes M_{pq}^0 , N_{qp}^0 , and M_q^0 for all parent-child pairs p, q and relies on M_{qp}^0 and N_{pq}^0 being pre-computed (these were computed during the initial dynamic programming step). As the user moves the mouse to drag a part $j(t)$ to location $\tilde{\theta}_{j(t)}$, we can display in real-time the solution corresponding to $M_{j(t)}^t[\tilde{\theta}_{j(t)}]$. When the user releases the mouse to finalize a part location $\tilde{\theta}_{j(t)}$, we encode the user response into an updated unary potential $u_{j(t)}$ according to Eqn 3.25, which is used to update the ML solution for that part:

$$M_{j(t)}^{t+1}[\theta_{j(t)}] = M_{j(t)}^t + u_{j(t)}(\theta_{j(t)}; \tilde{\theta}_{j(t)}) \quad (3.33)$$

We then propagate this new information to other parts using a single pass, breadth-first traversal of the graph T , originating from the node $j(t)$. The update step is depicted in Fig 3.16(d):

$$M_{qp}^{t+1} = M_{qp}^t, \quad N_{pq}^{t+1} = N_{pq}^t \quad (3.34)$$

$$M_{pq}^{t+1} = M_p^{t+1} - N_{pq}^{t+1} \quad (3.35)$$

$$N_{qp}^{t+1} = M_{pq}^{t+1} \otimes \Lambda_{qp} \quad (3.36)$$

$$M_q^{t+1} = M_{qp}^{t+1} + N_{qp}^{t+1} \quad (3.37)$$

where q is any neighbor of p . This update is efficient in practice and involves computing one distance transform operation per edge in the part tree. Both the update step and precomputation steps take linear time in the number of parts, scales, aspects, and pixel locations.

Acknowledgements

Parts of Section 3.1 are based on the paper “Multiclass Recognition and Part Localization with Humans in the Loop” by C. Wah and S. Branson and P. Perona and S. Belongie [78]. The dissertation author contributed to developing the algorithm, experiments, and writing of the paper.

Section 3.2 is based on the paper “Strong Supervision From Weak Annotation: Interactive Training of Deformable Part Models” by S. Branson and P. Perona and S. Belongie [8]. The dissertation author developed the algorithm and experiments and wrote most of the paper.

Chapter 4

Interactive Learning Algorithms

In this chapter, we propose a simple method for large scale learning and annotation of structured models. The system interleaves interactive labeling (where the current model is used to semi-automate the labeling of a new example) and online learning (where a newly labeled example is used to update the current model parameters). This is scalable to large datasets and complex image models and is shown to have good theoretical and practical properties in terms of train time, optimality guarantees, and bounds on the amount of annotation effort per image. We demonstrate that the system can be used to efficiently and robustly train part and pose detectors on the CUB Birds-200—a challenging dataset of birds in unconstrained pose and environment.

4.1 Introduction

Over the last few years, there has been growing interest in structured learning methods for problems such as part-based detection, scene understanding, and segmentation. Part-based methods [29, 6, 22] have achieved state-of-the-art results on datasets such as VOC detection and have demonstrated increasingly practical computational properties. There is growing awareness in the field that more strongly localized models are a necessary ingredient toward solving object detection, and, ultimately, scene understanding. This line of research has been held back by the size of available training sets and by the fact that most datasets do not go beyond image-level and bounding-box-level annotations. Unfortunately, more detailed annotation of things such as part locations

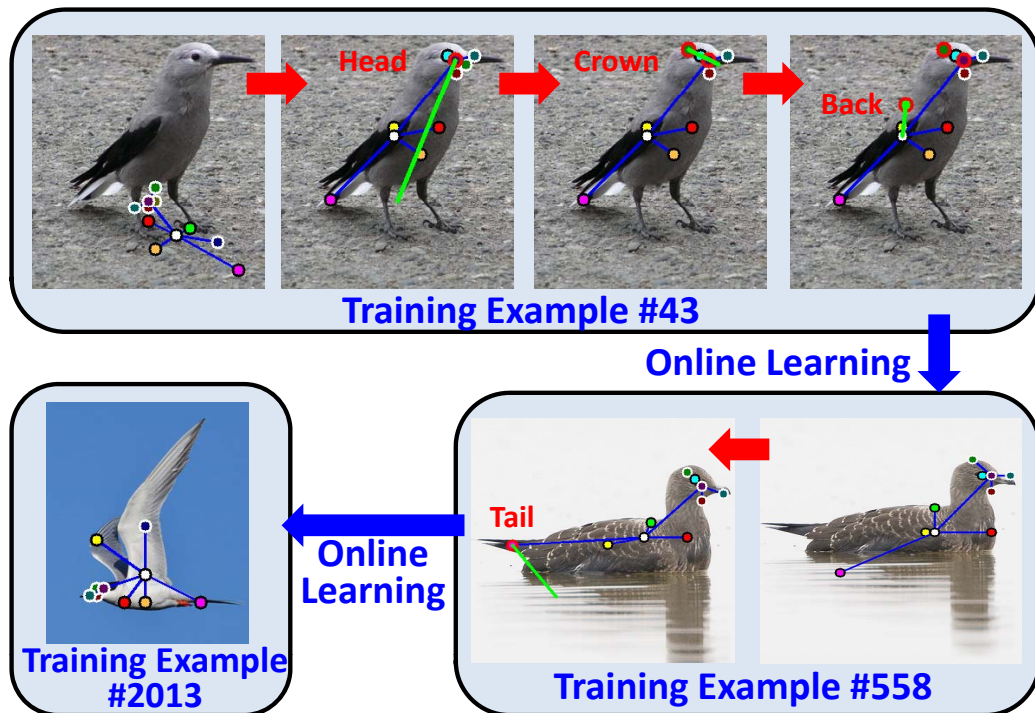


Figure 4.1: Interactive Labeling and Online Learning of Part Models: A part model is trained in online fashion, where annotation becomes increasingly automated as more images are labeled. The diagram shows how the interactive labeling interface changes on a particular test image as the size of the training set increases, with green lines representing parts that were dragged by the user.

and object poses can be expensive or logistically complicated to obtain.

Weakly supervised methods, where the level of annotation is less detailed than the underlying model, have shown great promise toward addressing this problem. Successful algorithms or applications include multiple instance learning [21], latent parts [29], latent structural SVMs [90], and expectation maximization on constellation models [84]. The solution found by these types of methods will usually be a local minimum of some non-convex objective function. Parameter learning of MRF/CRFs with latent/unobserved variables is in general an NP-hard problem. As a result, training can be slow and pinpointing the source of classification error—whether it’s due to optimization error, inappropriate model or feature space, or insufficient training data—is more of an art than a science.

Strongly supervised methods, where the level of annotation is the same as the

underlying model, are typically easier learning problems. A wide variety of strongly supervised learning algorithms and applications, such as binary classification, learning of sliding window detectors, parameter learning for CRFs, and structured prediction, can be formulated as convex optimization problems with polynomial time solutions. These algorithms have well understood theoretical properties with respect to computation time and generalization guarantees. The theoretical differences between strongly and weakly supervised algorithms means that the style and quality of annotation has a significant effect on the computational properties of training and the quality of the models learnt. Along this line of thought, Bourdev and Malik [6] have advocated "hyper-supervised" methods, arguing that researchers exaggerate the extent to which human annotation is the bottleneck to solving computer vision. They introduced a poselet model which requires more detailed labelings of parts and poses. The Lotus Hill dataset [89] of Zhu et al. echoes this sentiment.

In this chapter, we ask the question *is it possible to maintain the benefits of strongly supervised methods—computational tractability, performance guarantees, and scalability to more complex models—and the benefits of weakly supervised methods—reduced human annotation time—at the same time?* We argue that the answer is yes, using a combination of online learning and interactive labeling, which is depicted in Fig. 4.1. The basic idea is that a well functioning computer vision system should be able to predict all image labels with no human interaction, whereas an imperfect computer vision system (*e.g.*, one trained on insufficient training data) is still capable of accelerating the more mundane or obvious labeling tasks. Thus as we incrementally train a vision system, we should be able to increasingly reduce the amount of annotation per image.

General Framework: We propose the following framework for large scale training of computer vision systems:

1. Model the relationships between different variables using some structured model, such that runtime inference is computationally efficient
2. Ask a human to label a new image, using the current model to predict and display the maximum likelihood values of all variables as the user adjusts incorrect labels.

3. Update the learned model parameters using the newly labeled image
4. Repeat steps 2-3

We focus the discussion and experiments on annotation of deformable part models; however, the same basic methodologies should apply to a wide variety of other problems such as tracking, segmentation, and scene understanding.

Online Structured Learning: We employ online algorithms which optimize a structured SVM objective function [71], a convex optimization problem that has been applied to a wide variety of different problems in computer vision [5, 88, 52, 19, 64]. While the $\text{SVM}^{\text{struct}}$ solver is most commonly used, online optimization algorithms have been observed to be faster in practice [19, 88]. This result is supported theoretically by results relating to online learning algorithms for strongly convex loss functions [36, 33, 63, 57].

Theoretical Properties: Online algorithms have a few somewhat surprising theoretical properties that are useful in practice when applied to structured learning. First, asymptotic bounds for training time do not directly depend on the number of training images available. In practice this means that if one’s goal is to reach a solution within ϵ of the minimal achievable training error, the amount of processing per training image shrinks as the training set size increases.

Second, in the setting in which one keeps labeling new examples until one achieves ϵ -level *test error*, increasing the structural complexity of the model (*e.g.*, number of parts or alignment parameters) does not increase theoretical bounds on total annotation effort (if the feature space remains fixed and annotation effort is measured in terms of *total labels corrected*—the product of the number of training images required and correction operations per image).

Active Labeling: Interactive labeling—also called active labeling—has been applied to interactive image segmentation or matting [58, 87, 43], semi-automated video annotation [91, 77, 1], and active classification for class-attribute models [10, 47]. Active labeling interfaces use known relationships between variables in some structured model to reduce annotation time: the labels of neighboring pixels are correlated for segmentation methods, the position of an object in consecutive time frames are correlated for video an-

notation methods, and class and attribute variables are correlated for active classification methods. For our interface, the primary source of information for reducing annotation time is the spatial relationships between different parts.

Active Learning: Our work also has some similarities to the work of Vijayanarasimhan et al. [74, 75] relating to large scale annotation and active learning for structured objects. In active learning, computers intelligently decide which images and labels they want humans to annotate. In contrast, for interactive labeling methods, human annotators are the intelligent entity and decide which labels they want to correct. Active learning is a more ambitious learning problem with larger potential savings; however, in comparison to standard strongly supervised methods, it has higher computational complexity and fewer theoretical guarantees. In contrast, interactive labeling and online learning maintain the computational properties and theoretical guarantees of strongly supervised methods and may be applied to arbitrary structured prediction models.

4.2 Online Structured Learning

Our method jointly learns the appearance and spatial parameters of our deformable part model. We formulate the problem as a maximum margin structured learning problem (structured SVM [71]), which searches for the optimal vector of weights \mathbf{w}^* that minimizes the error function $F_n(\mathbf{w})$:

$$F_n(\mathbf{w}) = \frac{\lambda}{2} \|\mathbf{w}\|^2 + \frac{1}{n} \sum_{i=1}^n \ell_i(\mathbf{w}) \quad (4.1)$$

$$\ell_i(\mathbf{w}) = \max_y (\mathbf{w} \cdot \Phi(x_i, y) - \mathbf{w} \cdot \Phi(x_i, y_i) + \Delta(y_i, y)) \quad (4.2)$$

where $\{(x_1, y_1) \dots (x_n, y_n)\}$ is a training set of images and ground truth labels (for part detection, $y = \Theta$). $\Phi(x, y)$ is a vector of features extracted with respect to a particular prediction of part labels y (e.g., it concatenates HOG features extracted from around each part location and squared distances between adjacent parts). This criterion attempts to learn a set of weight parameters \mathbf{w} , such that the score extracted at the ground truth part locations $\mathbf{w} \cdot \Phi(x_i, y_i)$ is greater than the score of any other choice of part locations $\mathbf{w} \cdot \Phi(x_i, y)$ by at least $\Delta(y_i, y)$, a customizable loss function encoding the penalty of

Algorithm 3 ONLINEINTERACTIVEPARTLEARNER

- 1: Initialize $\mathbf{w}_0 \leftarrow \mathbf{0}$, $s \leftarrow 0$
 - 2: **for** $i = 1$ to n **do**
 - 3: Obtain new example x_i :
 $y_i \leftarrow \text{INTERACTIVEPARTLABELER}(\mathbf{w}_s, x_i)$
 - 4: Update weights using Eq 4.5 or 4.7
 - 5: $s \leftarrow s + 1$
 - 6: Optional: w/ spare CPU cycles, repeat lines 4-5
 - 7: **end for**
-

predicting part locations y when the true locations are y_i .

The structured hinge loss $\ell_i(\mathbf{w})$ is convex in \mathbf{w} , because it is the maximum of a set of affine functions. The gradient (or technically a sub-gradient) of ℓ_i can be computed by solving a problem similar to an inference problem:

$$\bar{y}_i = \max_y (\mathbf{w} \cdot \Phi(x_i, y) + \Delta(y_i, y)) \quad (4.3)$$

$$\nabla \ell_i = \Phi(x_i, \bar{y}_i) - \Phi(x_i, y_i) \quad (4.4)$$

Learning with strongly convex loss functions (this includes arbitrary convex loss function with L_2 regularization added such as Eq 4.1) has recently been extensively studied in online learning literature [36, 33, 63, 57]. [33, 36] show that for learning problems with λ -strongly convex loss functions (*e.g.*, the form of Eq 4.1), an online stochastic gradient descent (SGD) which streams in an example (x_s, y_s) and takes an update step

$$\mathbf{w}_s = \mathbf{w}_{s-1} - \frac{1}{\lambda s} (\lambda \mathbf{w}_{s-1} + \nabla \ell_s) \quad (4.5)$$

achieves at most logarithmic regret

$$\sum_{s=1}^S f_s(\mathbf{w}_s) - \min_{\mathbf{w}} \sum_{s=1}^S f_s(\mathbf{w}) \leq \frac{R^2 (\log S + 1)}{2\lambda} \quad (4.6)$$

where $f_s(\mathbf{w}) = \frac{\lambda}{2} \|\mathbf{w}\|^2 + \ell_s(\mathbf{w})$ and R is a bound on the magnitude of the gradient of $f_s(\mathbf{w})$. The regret is measured as the total loss incurred as one streams in new unseen training examples as compared to the minimum achievable loss over the entire training

set. It implies that even when using a simple optimization algorithm which takes only one gradient step per training example, average test error goes with $O(\frac{\log n}{n})$ —a faster statistical convergence rate than those implied by standard VC bounds for binary classification (if the loss one cares about some is some strongly convex loss function instead of 0/1 binary classification loss, bounds on generalization error go with $\tilde{O}(\frac{1}{n})$ instead of $O(\frac{1}{\sqrt{n}})$).

For structured SVMs (as well as for linear SVMs), R is related to a bound on the L_2 norm of the image of $\Phi(x, y)$ and is typically proportional to the dimensionality of the feature space. Regret bounds hold regardless of the order examples are processed and for any algorithm that improves the dual objective to Eq 4.1 by at least as much as SGD [36]. A variant of SGD is used by Pegasos [63], a popular online learning algorithm for linear SVMs. A slightly better update

$$\mathbf{w}_s = \frac{s-1}{s} \mathbf{w}_{s-1} - \min \left(\frac{1}{\lambda s}, \frac{\ell_s(\mathbf{w}_{s-1})}{\|\nabla \ell_s\|^2} \right) \nabla \ell_s \quad (4.7)$$

solves for the step size which maximally improves the dual objective in closed form and is similar to an online version of the update used by LIBLINEAR [25], a fast optimizer for linear SVMs.

By choosing S such that regret bounds in Eq 4.6 are less than ϵ , one can show that if one iterates for $S = \tilde{O}(\frac{R^2}{\lambda \epsilon})$ iterations and processes each training example an equal number of times, then the converged solution is guaranteed to be within ϵ of the minimal achievable training error. Similarly, if one attains $n = \tilde{O}(\frac{R^2}{\delta \lambda \epsilon})$ training examples, then with probability at least $1 - \delta$, the expected error on a random test example will be within ϵ of the minimum achievable model error $f(\mathbf{w}^*)$.

One important implication is that training time does not depend directly on the number of training examples. In practice this means that that the number of iterations of gradient descent one must run per training example shrinks as the training set size increases. Secondly, structured SVMs have the same empirical and statistical convergence properties as linear SVMs; the only difference is that for structured SVMs the time to compute the gradient (Eq 4.4) grows with inference time.

Combining Interactive Labeling: Incorporating interactive labeling is simple: every time we obtain a new training example, we use our our current model parameters \mathbf{w}_s



Figure 4.2: Typical Results on Birds-200, with blue dots denoting parts predicted by a deformable part model trained on a 1000 image training set, and red dots denoting parts that were corrected by a simulated user (as described in Section 4.3)

to accelerate the labeling process (see Algorithm 3). We use a loss function $\Delta(y_i, y)$ equal to the number of misclassified labels (*e.g.* number of incorrect parts in a given image). We assume a predicted part location is correct if its x, y location is within some sufficiently small radius from the ground truth location.

One motivation for using structured SVMs is that the structured hinge loss $\ell_i(\mathbf{w})$ is always at least as big as the custom loss function $\Delta(y_i, y)$ [71], as is the regularized error $f_i(\mathbf{w})$. Thus the total loss incurred during online learning $\sum_{s=1}^S f_s(\mathbf{w}_s)$ (which is bounded by regret bounds in Eq 4.6) is an upper bound on the total number of incorrect labels throughout the course of training. As a consequence, obtaining $n = \tilde{O}(\frac{R^2}{\delta\lambda\epsilon})$ training examples ensures not only that (with high probability) average test error will be no more than $f(\mathbf{w}^*) + \epsilon$ but also that the average number of labels corrected per training example will be no more than $f(\mathbf{w}^*) + \epsilon$. Intuitively, it becomes harder and harder to drive down generalization error as one adds more and more training examples, such that the majority of training examples are labeled with a similar level of error as that at final convergence. These results suggest that increasing the structural complexity of the model (*e.g.* adding more parts or mixture components) while fixing the dimensionality of the feature space $\Phi(x, y)$ does not necessarily increase the total amount of annotation effort during training.

Labeling Bias and Time Considerations: We emphasize though that these results are measured in terms of the *total number of labels corrected* during training and not directly in terms of human annotation time. We still assume that the user must verify correctness

of machine-predicted labels. Clearly in practice this will result in additional annotation time. Secondly, bounds on the number of corrected labels are less useful if $f(\mathbf{w}^*)$ is high (*e.g.*, the chosen feature space saturates and is not capable of getting good performance).

A second concern relates to the effect of interactive labeling on biasing user labels. For example, using an interactive part labeling tool will result in slightly different labeled pixel locations for some parts. It is our assumption that an annotator will submit a final label which he/she believes to be acceptable. Continuous variables such as part locations have some range of acceptability and are prone to fluctuation from annotator to annotator. Interactive labeling biases annotated locations within this range. We intend to study effects of labeling bias and annotation time in future work.

Diagnosing Sources of Error: Combining online learning and active labeling has a few nice practical properties which facilitate debugging when training is unsuccessful. Methodologies for diagnosing problems due to insufficient training data, insufficient computation time, bad model or feature space, and annotation error are described in the supplementary material.

4.3 Experiments

To demonstrate the practicality and effectiveness of our interactive labeling and learning system, we test performance on two different datasets using two different user interfaces: CUB-200-2011 [79], which allows users to simply click and drag the location of a particular part, and a dataset of synthetic birds, which also allows users to alter the scale, orientation, and aspect of each part. Results on the synthetic dataset are included in the supplementary material.

CUB-200-2011 [79] is an extended version of the Caltech-UCSD Birds 200 dataset [86] and contains 11,788 images of birds of 200 species. The dataset contains uncropped images of birds in the wild, including birds that are flying, perched, swimming, truncated and occluded. Each image was annotated by 5 different Mechanical Turk users by a simple x, y coordinate (*e.g.* users were asked to click on the center of each part) and associated with a non-semantic aspect.

Although our interface is practical and realtime, it requires background process-

ing to precompute lookup tables when the user releases the mouse. Thus the engineering challenges associated with mass-deploying our system on MTurk were beyond the scope of this paper. Since the dataset contains an exhaustive set of part click locations for each training image, we constructed a simulated user interface as follows:

1. The computer vision system updates its prediction of the most likely part locations
2. The simulated user selects and drags the part with maximum distance to his/her click response (normalized by a per part standard deviation)
3. If all part predictions are within 1.5 standard deviations from the user's click response, the session ends. Otherwise, steps 1-2 are repeated.

We processed training images in random order, ignoring bird species labels. The standard deviation of user click responses was computed separately for each part, using 5 different MTurk responses per image. Qualitatively, the simulated interface was fairly true to life (see Figure 4.2).

Since we are interested in understanding how total annotation time changes as we train our part detectors in an online fashion, we varied the training set size from 50 to 4000 images and used the remaining images as test data. The results of our experiments are summarized in Figure 4.3. Each curve in Figure 4.3a shows part prediction accuracy (measured as the number of parts within 1.5 standard deviations of a user click response) as a function of the number of parts corrected by the interactive interface.

Additionally, we measure the average number of parts that needed to be labeled until all part predictions were deemed acceptable. We see that of 13 possible parts, on average the user needed to label 6.6 parts when using only 50 training examples. This was reduced to 3.9 parts when the training set was increased to 4000 images. At this point, the benefits of adding more training images were small, and errors were mostly attributable to saturation of the model/feature space.

Figure 4.3b plots the same results, except that we measured performance as a function of the duration (in terms of human time spent) of the interactive interface. This was estimated using timing data for each part click response in the CUB-200-2011 dataset. The average duration of an interactive labeling session was 12.0 seconds when

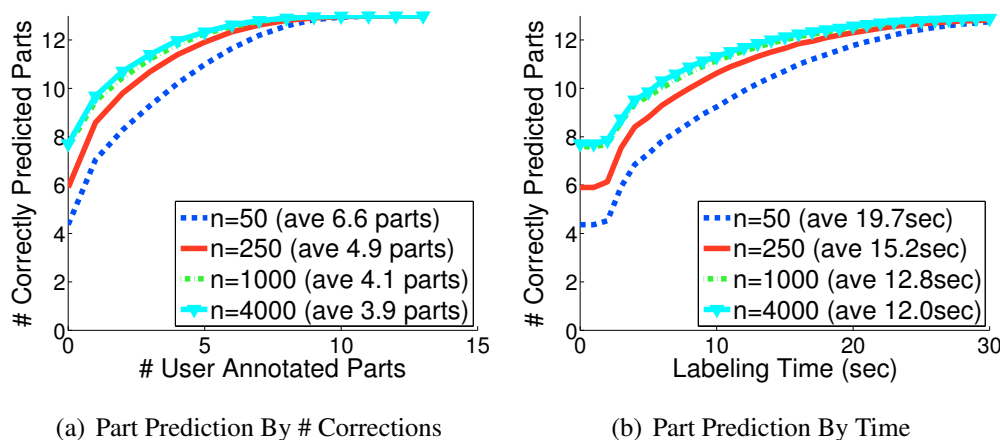


Figure 4.3: Results on Birds-200: (a) Average part prediction accuracy as a function of the number of annotated parts per image. Each curve shows performance for a different training set size, indicating annotation becomes progressively more automated as more images are labeled. The legend shows the average number of parts that needed to be labeled until all 13 were correct. An upward curving plot indicates interactive labeling is effective. (b) Part prediction accuracy as a function of labeling time per image.

using the detectors trained on the set of 4000 images and 19.7 seconds when trained on 50 images.

Computational Properties: The bird model we used consisted of 13 different parts, 11 aspects, and 4 scales. Total preprocessing time (which includes computing HOG features, evaluating sliding window detectors, and running dynamic programming) takes less than 1 second on a single 2.4GHz CPU. As the user drags a part, the predicted part locations are displayed as a simple lookup operation (which easily runs in realtime). Each time the user releases the mouse to finalize a part location, lookup tables are updated, which takes approximately .3 seconds. Total training time of the standalone on-line structured learning algorithm was approximately 3 hours on the 4000 image dataset on an 8-core computer when training, where training was stopped when it reached an approximation factor of $\epsilon = .02$ from the minimal achievable training error.

4.4 Conclusion

We proposed a framework for large scale annotation and learning of structured models that has excellent theoretical properties in terms of computation time and annotation effort. We introduced a novel interface for interactive labeling of deformable part models that is capable of updating and displaying the maximum likelihood location of each part in realtime as the user drags the mouse, and applied this model to our interactive, online learning framework. In future work, we hope to deploy our system on a larger scale and apply similar methodologies to other domains such as segmentation, tracking, and scene understanding.

Acknowledgements

This chapter is based on the paper “Strong Supervision From Weak Annotation: Interactive Training of Deformable Part Models” by S. Branson and P. Perona and S. Belongie [8]. The dissertation author developed the algorithm and experiments and wrote most of the paper.

Chapter 5

Methodologies For Diagnosing Errors in Learned Computer Vision Systems

5.1 Introduction

Suppose we have trained an object detection system and discover that test performance is lower than expected. How can we diagnose what went wrong? Is the problem due to insufficient training data, a bad model or feature space, annotation error, or insufficient computation time? If we can pinpoint the main bottleneck, how can we enact a change to fix it without recollecting an entirely new dataset or re-training from scratch?

These questions are fundamental problems in computer vision and machine learning that one will inevitably encounter when building a practical system. Although every good researcher will develop her own methodologies for solving these problems, these issues are seldom the focus of academic papers. In this paper, we introduce an integrated framework for training, collecting data, diagnosing different sources of test error, and correcting problems. It is designed to be computationally practical and scalable to large datasets, theoretically sound, and general enough to be applied to a wide variety of different learning problems, including multiclass classification, object detection, deformable part models, and attribute models. The general approach is summarized in the caption of Fig 5.1.

Development of learning-based computer vision systems can be broken down

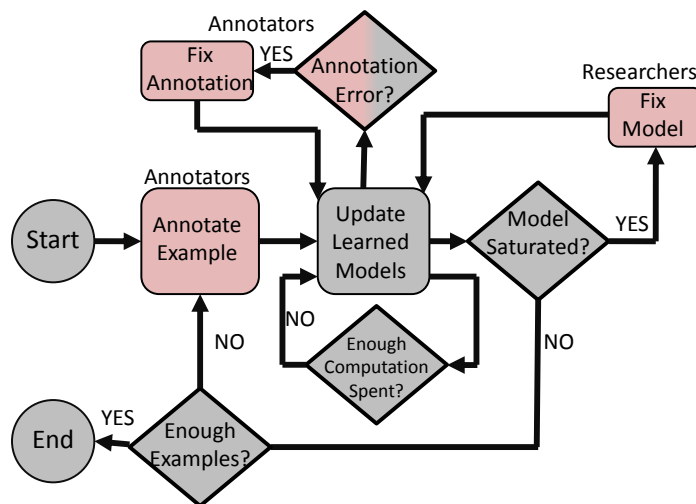


Figure 5.1: Proposed framework for annotation, training, and diagnosing different types of errors. Items in gray are executed by a computer, while items in pink are executed by a human. Items drawn using diamonds indicate a test or diagnosis technique. Tests to determine whether or not the model or feature space has saturated, sufficient training examples have been labeled, or sufficient computation time has been spent are automated by the computer system and can be communicated to the researcher/annotator via dynamically generated plots or figures. A test to determine if an example has been mislabeled is semi-automated by the computer system: it proposes examples that are likely to be mislabeled, which in turn must be verified by a human annotator. All ensuing changes such as newly labeled training examples, manually corrected labels, and changes to the model or feature space are implemented using online updates to the learned model without re-training the system.

into four major modules or disciplines—dataset collection, feature and model design, statistical machine learning, and optimization algorithms. Individually, each of these disciplines is fairly mature, with sophisticated algorithms and well developed theoretical analysis; however, we believe that methodologies to correctly combine them together into a cohesive system are still deficient and unsatisfactory. Each module has a number of different choices or parameters, and these choices have complex dependencies on other modules. For example, when collecting a dataset, one can choose what type of annotation to collect (*e.g.*, class labels, bounding boxes, part locations, segmentations) and how many images to label. The type of annotation determines which types of image models (*e.g.*, sliding window, pictorial structures, *etc.*) are computationally tractible.

The number of training examples that one should collect depends on the complexity of the feature space (*e.g.*, larger feature spaces requiring more training examples to avoid over-fitting and smaller feature spaces requiring fewer examples but being incapable of getting low training error). The type of image model impacts which types of features are appropriate (*e.g.*, a non-localized model necessitates translation invariant features like bag-of-words, whereas a sliding window or part-based model can use spatially localized features). Different types of optimization algorithms and learning objective functions are tractable for large datasets vs. small datasets. As a result, there is a severe limitation on how far we can go if we think about each module fully independently or treat other modules as black boxes. The dependencies that choices in different modules have on one another have significant effects on the asymptotic properties of both compute time and the number of training examples required to attain a given accuracy on test data—they are not mere details or simple parameter choices.

Moreover, we have a tendency to not prioritize research according to what is the biggest bottleneck toward solving the problem as a whole. For example, data collection can be laborious and is usually perceived as an unglamorous or a difficult area to publish academic papers. As a result, most researchers will depend primarily on fixed datasets that were collected by a different research group. Identifying a dataset as a bottleneck is a slow and cumbersome process in our current research culture: we observe general trends in dataset performance levels from paper deadline to paper deadline, which eventually taper off as researchers sweep over different possible choices of what features and learning algorithms. For example, performance levels on Caltech-101 have not increased very much in the last 5 years, and improvements on the VOC Detection Challenge have slowed down significantly in the last 3 years. As a result, people increasingly suspect that obtaining larger datasets or different types of annotations will be necessary if we truly want to solve computer vision problems. To be able to more quickly diagnose the bottleneck of learning-based computer vision systems, we argue that datasets should evolve at a finer granularity, and that procedures for data collection should be more integrated with procedures for choosing features and learning algorithms.

Our approach to this problem is limited to problems that can be formulated as a structured SVM problem; however, it could also be applied to other problems that

optimize a convex upper bound on some loss function. We show in Chapter 7 that this formulation encompasses many of the most popular and highest performing methods for multiclass classification [53, 42], object detection [29, 73], pose registration [88, 93], attribute learning [40, 26], tracking, segmentation, and action recognition. In Chapter 6, we demonstrate that training that this approach is computationally practical for large datasets. In this chapter, we add a few additional components:

1. We augment the online structured SVM learning algorithm from Chapter 6 to efficiently compute statistics to diagnose errors due to insufficient training data, bad model or feature space, or insufficient computation time, which can be interactively communicated to researchers and annotators
2. We introduce a procedure for semi-automatically identifying mislabeled, ambiguous, or difficult training examples, and introduce online update steps that allow annotators to remove examples, correct labels, or re-weight examples without re-training the system.
3. We introduce an algorithm for incrementally augmenting the feature space that can be interpreted as a boosting algorithm applied to max-margin structured prediction

5.2 Algorithm and Main Framework

5.2.1 Optimization Algorithm

Our framework is applicable to structured SVM learning; please refer to 1.2.1-1.2.2 for a description of the applicable notation and problem definition. The basic goal is to train a system to predict a structured output Y given a data example X (*e.g.*, Y could be an image segmentation, set of detected part locations, *etc.*).

In this chapter, we extend the optimization framework that is described in Chapter 6 to add different types of interactive feedback to annotators and researchers and online updates for changing the model or feature space or re-labeling examples. The goal of the optimization algorithm minimize the structured SVM training error $F_n(\mathbf{w}) =$

$\sum_{i=1}^n f(\mathbf{w}; X_i, Y_i)$, where $f(\mathbf{w}; X_i, Y_i)$ is a strongly convex loss function (see Eq 6.19) that is an upper bound on a customizable loss function $\Delta(Y, Y_i)$. The optimization algorithm works by sequentially updating \mathbf{w} with respect to a training example (X_i, Y_i) , where (X_i, Y_i) could be a newly labeled training example or one that has already been processed before. The update is based on choosing dual parameters α_i that maximize (or approximately maximizes) the dual objective

$$\max_{\alpha_1 \dots \alpha_n} \mathcal{D}_n(\alpha_1 \dots \alpha_n) = \min_{\mathbf{w}} F_n(\mathbf{w}) \quad (5.1)$$

where $\mathcal{D}_n(\alpha_1 \dots \alpha_n)$ can be written as (see [71])

$$\mathcal{D}_n(\alpha_1 \dots \alpha_n) = -\frac{1}{2\lambda n} \sum_{i,Y} \sum_{j,Y'} \alpha_i^Y \alpha_j^{Y'} \langle \mathbf{v}_i^Y, \mathbf{v}_j^{Y'} \rangle + \sum_{i,Y} \alpha_i^Y \Delta(Y, Y_i) \quad (5.2)$$

$$\text{s.t., } \forall_i \left(\forall_Y, \alpha_i^Y \geq 0, \quad \sum_Y \alpha_i^Y \leq 1 \right) \quad (5.3)$$

Here, each \mathbf{v}_i^Y is a vector

$$\mathbf{v}_i^Y = \Psi(X_i, Y) - \Psi(X_i, Y_i) \quad (5.4)$$

that is weighted by a corresponding scalar α_i^Y , with $\alpha_i^{Y_1}, \alpha_i^{Y_2}, \alpha_i^{Y_3}$ encoding a weight on every possible label Y for a training example i . For example, for sliding window based detection, each \mathbf{v}_i^Y is a vector of features extracted at a candidate bounding box location Y and is weighted by a dual parameter α_i^Y . Note that $\mathbf{v}_i^{\bar{Y}^t} = \nabla \ell(\mathbf{w}, Z_t)$. The relationship between the dual parameters α and primal parameters \mathbf{w}^t is

$$\mathbf{w}^t = -\frac{1}{\lambda t} \sum_{i=1}^t \mathbf{u}_i, \quad \mathbf{u}_i = \sum_Y \alpha_i^Y \mathbf{v}_i^Y \quad (5.5)$$

Pseudo code for the full algorithm is depicted in Algorithm 4.

5.2.2 Diagnostic Statistics

In this section, we show how the optimization algorithm presented in the previous section can be modified to allow efficiently computable empirical estimates of the minimum achievable model error, training error, and test error, which can be used to differentiate errors due to insufficient training data, bad model or feature space, or insufficient computation time spent.

Lower bound on minimum achievable training error:

We can compute a lower bound E_{model}^T on the minimum achievable training error:

$$E_{model}^T = \sum_{t=1}^T \Delta \mathcal{D}_t \quad (5.6)$$

$$\leq \min_{\mathbf{w}} F_T(\mathbf{w}) \quad (5.7)$$

The inequality follows from the weak duality theorem

$$\mathcal{D}_T(\alpha_1 \dots \alpha_T) \leq \min_{\mathbf{w}} F_T(\mathbf{w}) \quad (5.8)$$

where $\mathcal{D}_T(\alpha_1 \dots \alpha_T) = \sum_{t=1}^T \Delta \mathcal{D}_t$. Thus if $\frac{E_{model}^T}{T}$ is high, it implies that the model or feature space has definitely saturated and it is incapable of getting a lower average error per training example. Given a target test error level ϵ , we can signal the researcher to make a change to the model or feature space if $\frac{E_{model}^T}{T} \geq \epsilon$. For a dataset of size n , $\frac{E_{model}^T}{T}$ converges toward the minimum achievable training error as T increases: $\frac{E_{model}^T}{T} + O(\frac{\log T}{T}) = \frac{1}{n} \min_{\mathbf{w}} F_n(\mathbf{w})$ (see Thm 5.2.1).

Optimization error:

We can compute an online estimate of the training error E_{train}^T :

$$E_{train}^T = \sum_{t=1}^T f(\mathbf{w}^{t-1}; Z_t) \quad (5.9)$$

which is the loss measured in each iteration of Algorithm 4. The quantity

$$E_{regret}^T = E_{train}^T - E_{model}^T \quad (5.10)$$

is an estimate of the error we have incurred because we haven't spent enough computation time¹. $\frac{E_{regret}^T}{T}$ converges toward zero as T increases: $\frac{E_{regret}^T}{T} = O(\frac{\log T}{T})$ (see Thm 5.2.1). If $\frac{E_{regret}^T}{T} \geq \epsilon$ and each training example has been processed by Algorithm 4 an equal number of times, we can safely conclude that the amount of error due to insufficient computation time spent is less than ϵ and can safely sleep until annotators add a new training example (thereby saving CPU cycles).

¹it is actually an empirically measured upper bound on the online regret

Online estimate of test error:

We can compute an online estimate of the test error E_{test}^n :

$$E_{test}^n = \sum_{i=1}^n f(\mathbf{w}^{t_i-1}; Z_{t_i}) \quad (5.11)$$

where $t_i = S_i^1$ stores the iteration when example (X_i, Y_i) was first processed by Algorithm 4. In other words, E_{test}^n measures the loss on each example the first time it was encountered and before it was used to update the model parameters. This is one practical benefit of online learning algorithms (*e.g.*, we can efficiently estimate test error without necessitating expensive procedures like leave-one-out cross validation). If $E_{test}^n \leq \epsilon$, it implies that we have reached our target test performance and can terminate training and labeling new examples.

5.2.3 Theoretical Guarantees

We can use results from online learning theory to bound test error for customizable loss functions $\Delta(g(X_t; \mathbf{w}), Y)$, giving us worst case bounds on the number of training examples and amount of computation time we will need to attain a given level of test error. Since these bounds will usually be very pessimistic in practice, we also relate these bounds to the empirically measured statistics introduced in the previous section provide (which will be much tighter in practice).

Theorem 5.2.1 *Let $f(\mathbf{w}; Z)$ be the structured SVM objective defined in Eqn 1.7. Let $L = \sqrt{\lambda} + 2R$, where $\|\Psi(X, Y)\| \leq R$ is a bound on the image of Ψ . If Algorithm 4 is run for T iterations:*

1. **Generalization Error:** *With $T = n$ where training examples $Z_1 \dots Z_n$ are selected independently at random, then*

$$\mathbb{E}_Z[\Delta(g(X; \mathbf{w}^T), Y)] \leq \mathbb{E}_{Z_1 \dots Z_n} \left[\frac{E_{test}^n}{n} \right] \quad (5.12)$$

and with probability at least $1 - \delta$

$$\mathbb{E}_Z[\Delta(g(X; \mathbf{w}^T), Y)] \leq \min_{\mathbf{w}} \mathbb{E}_Z[f(\mathbf{w}; Z)] + \frac{L^2(\log(n) + 1)}{2\delta\lambda n} \quad (5.13)$$

2. **Optimization error:** Let $\bar{\mathbf{w}} = \frac{1}{T} \sum_{t=1}^T \mathbf{w}^{t-1}$. If Algorithm 4 is run for $T = mn$ iterations, passing over each example $m \geq 1$ times. The average training error can be bounded in relation to the minimum achievable training error:

$$\frac{1}{n} F_n(\bar{\mathbf{w}}) \leq \left(\frac{1}{n} \min_{\mathbf{w}} F_n(\mathbf{w}) \right) + \frac{E_{regret}^T}{T} \quad (5.14)$$

$$\leq \left(\frac{1}{n} \min_{\mathbf{w}} F_n(\mathbf{w}) \right) + \frac{L^2(\log(mn) + 1)}{2\lambda mn} \quad (5.15)$$

A proof is in Appendix 5.3. Similar results were first presented in [57]. One implication is that if we take only a single pass through the training set (setting $m = 1$), where train time equals test time, the optimization error (*e.g.*, error because we haven't spent enough computation time) converges at the same asymptotic rate as the generalization error (*e.g.*, error because we don't have enough training examples). Thus when computation time is a bottleneck, it is better to use as many training examples as possible (use a large value for n and a small value for m). A second implication is that the convergence rates for structured SVMs are the same as for linear SVMs, with the same constants involved [63], and therefore we don't sacrifice theoretical guarantees by using an application specific loss function $\Delta(g(X_t; \mathbf{w}), Y)$.

5.2.4 Non-Traditional Online Updates

In this section, we derive different types of online update steps, including removing examples, correcting mislabeled examples, re-weighting difficult examples, and altering the feature space. As a building block to these update steps, let $j < t$ be some iteration of Algorithm 4 with corresponding most violated label \bar{Y}_j and weight $\alpha_j^{\bar{Y}_j}$. Suppose we want to change $\alpha_j^{\bar{Y}_j}$ to some new value $\alpha_j^{\bar{Y}_j^*}$. Then the subsequent change in \mathcal{D}_t (Eq 5.3) and \mathbf{w}_t (Eq 5.5) are:

$$\Delta \mathcal{D}_t^\alpha = \mathcal{D}_t(\alpha_1 \dots \alpha_{j-1}, \alpha_j^{\bar{Y}_j^*}, \dots, \alpha_t) - \mathcal{D}_t(\alpha_1 \dots \alpha_{j-1}, \alpha_j^{\bar{Y}_j}, \dots, \alpha_t) \quad (5.16)$$

$$= (\alpha_j^{\bar{Y}_j^*} - \alpha_j^{\bar{Y}_j}) \left(\langle \mathbf{w}^t, \mathbf{v}_j^{\bar{Y}_j} \rangle + \Delta(\bar{Y}_j, Y_j) \right) - \frac{\left((\alpha_j^{\bar{Y}_j^*} - \alpha_j^{\bar{Y}_j}) \mathbf{v}_j^{\bar{Y}_j} \right)^2}{2\lambda t} \quad (5.17)$$

$$\mathbf{w}^t \leftarrow \mathbf{w}^t - \frac{\alpha_j^{\bar{Y}_j^*} - \alpha_j^{\bar{Y}_j}}{\lambda t} \mathbf{v}_j^{\bar{Y}_j} \quad (5.18)$$

This procedure is implemented in the function $\text{CHANGEALPHA}(j, \alpha_j^{\bar{Y}_j^*})$. Note that $\Delta \mathcal{D}_t^\alpha$ is maximized by setting $\alpha_j^{\bar{Y}_j^*}$ to:

$$\alpha_j^{\bar{Y}_j^*} = \min \left(1, \max \left(0, \alpha_j^{\bar{Y}_j} + \frac{\lambda t \left(\langle \mathbf{w}^t, \mathbf{v}_j^{\bar{Y}_j} \rangle + \Delta(\bar{Y}_j, Y_j) \right)}{\|\mathbf{v}_j^{\bar{Y}_j}\|^2} \right) \right) \quad (5.19)$$

Correcting Misabeled or Difficult Training Examples:

Since correcting mislabeled training examples fully automatically is unrealistically difficult, we propose a simple semi-automated procedure for identifying and correcting mislabeled examples. Here, our algorithm maintains a list of training examples that are sorted by their likelihood of being mislabeled. An annotator can browse this list as a sorted gallery of images and choose to re-label examples that appear to be incorrect. We assume that if an example repeatedly has high training error it is more likely to be mislabeled, inherently difficult, or ambiguous to label. We thus compute e_i as the average training error of X_i, Y_i over the course of Algorithm 4

$$e_i = \frac{1}{|S_i|} \sum_{j \in S_i} f(\mathbf{w}^{j-1}; Z_j) \quad (5.20)$$

and present the user with a gallery of images sorted by e_i . The annotator can perform two different kinds of operations: *label correction*, where a bad label Y_i is replaced by a good label Y_i^* , or *loss correction*, where the loss function for example i is changed from $\Delta(Y, Y_i)$ to some custom value $\Delta^*(Y, Y_i)$. The latter case is intended to handle examples that are inherently difficult or ambiguous to label. In other words, if an example is difficult to label even for a human, we can down play its contribution to our training error.

Both corrections can be performed using an online update that does not necessitate re-training. The update is implemented in $\text{RELABELEXAMPLE}(i, Y_i^*, \Delta^*)$ of Algorithm 5. The procedure begins by removing the old label by setting each $\alpha_j^{\bar{Y}_j}$ to 0. The old label is replaced with the new one, whereupon we solve for the new optimal value of $\alpha_j^{\bar{Y}_j}$ according to Eq 5.19.

Augmenting the Feature Space:

In Section 5.2.2, we introduced a test for detecting that the feature space has saturated, such that achieving lower training error is impossible. At this point, the researcher should choose to increase the complexity of the model or feature space. In this section, we introduce an algorithm for intelligently selecting a new feature to add from a pool of candidate features $\phi_1(Z)\dots\phi_F(Z)$. Suppose Algorithm 4 has been run for t iterations, such that $F_n(\mathbf{w})$ has been approximated by $\bar{F}_n^t(\mathbf{w})$, the loss over samples $\bar{Y}_1\dots\bar{Y}_t$:

$$\bar{F}_n^t(\mathbf{w}) = \frac{t\lambda}{2}\|\mathbf{w}\|^2 + \sum_{i=1}^t \bar{\ell}_i(\mathbf{w}) \quad (5.21)$$

$$\bar{\ell}_i(\mathbf{w}) = \max(0, \langle \mathbf{w}, \Psi(X_i, \bar{Y}_i) - \Psi(X_i, Y_i) \rangle + \Delta(\bar{Y}_i, Y_i)) \quad (5.22)$$

By Thm 5.2.1, $\bar{F}_n^t(\mathbf{w})$ becomes an arbitrarily good approximation as t increases:

$\frac{1}{n} \min_{\mathbf{w}} F_n(\mathbf{w}) = \frac{1}{t} \bar{F}_n^t(\bar{\mathbf{w}}) + O(\frac{\log t}{t})$. We can select a new feature to add $\phi_j(Z)$ using a boosting algorithm that selects the next feature based on a weighted training set:

$$j = \arg \max_j \left| \sum_{i=1}^t \beta_i (\phi_j(X_i, \bar{Y}_i) - \phi_j(X_i, Y_i)) \right| \quad (5.23)$$

where β_i is a per example weight:

$$\beta_i = 1[\bar{\ell}_i(\mathbf{w}) > 0] \quad (5.24)$$

This weighting scheme can be derived using either the functional gradient descent or coordinate descent view of boosting, where Eq 5.23 selects the feature j for which $|\frac{d\bar{F}_T}{dw_j}|$ is highest (the instantaneous change in \bar{F}_T is highest as w_j goes to 0):

$$\frac{d\bar{F}_T}{dw_j} = \sum_{i=1}^t \beta_i (\phi_j(X_i, \bar{Y}_i) - \phi_j(X_i, Y_i)) \quad (5.25)$$

Another possible weighting scheme is to directly use the current dual parameters as per-example weights:

$$\beta_i = \alpha_i^{\bar{Y}_i} \quad (5.26)$$

which corresponds to choosing the feature that maximally reduces the dual objective (Eq 5.3). This can be interpreted as choosing the feature that has potential to reduce the

minimum achievable training error by the most, since

$$\arg \min_{\mathbf{w}, w_j} \bar{F}_n^t([\mathbf{w}, w_j]) \geq \bar{F}_n^t(\mathbf{w}^t) + \Delta \mathcal{D}_t^j \quad (5.27)$$

where $\bar{F}_n^t([\mathbf{w}, w_j])$ is the training error when appending the feature ϕ_j to Ψ , and $\Delta \mathcal{D}_t^j$ is the change in Eq 5.3 when augmenting the feature space by ϕ_j

$$\Delta \mathcal{D}_t^j = -\frac{1}{2\lambda t} \left(\sum_{i=1}^t \alpha_i^{\bar{Y}_i} (\phi_j(X_i, \bar{Y}_i) - \phi_j(X_i, Y_i)) \right)^2 \quad (5.28)$$

5.3 Proof of Thm 5.2.1

Thm 5.2.1 can be derived from results from [36, 33, 63, 62], which have shown that strongly convex loss functions have fast statistical convergence rates and very efficient online optimization algorithms. These results can be applied to structured SVMs (which is a strongly convex loss function). A similar result is in [57]. We reproduce the results here to put things in terms of our notation and to prove validity of our empirically measured diagnostic statistics. First, we establish a few lemmas that will allow us to apply the results of [36] to structured SVMs:

Lemma 5.3.1 *The projection step in Line 6 of Algorithm 4 can only increase the dual objective*

Proof Line 6 of Algorithm 4 projects \mathbf{w}^t onto the L_2 ball $\|\mathbf{w}^t\|^2 \leq \frac{1}{\lambda}$. It checks if $\|\mathbf{w}^t\| > \frac{1}{\sqrt{\lambda}}$, and if so scales \mathbf{w}^t by

$$s \leftarrow \frac{1/\sqrt{\lambda}}{\|\mathbf{w}^t\|} \quad (5.29)$$

where $0 < s < 1$. The corresponding change in the dual objective is

$$\Delta \mathcal{D}_t^{proj} = \mathcal{D}_t(s\alpha) - \mathcal{D}_t(\alpha) \quad (5.30)$$

$$= \left[-\frac{t\lambda}{2} \|s\mathbf{w}^t\|^2 + s \sum_{i,Y} \alpha_i^Y \Delta(Y, Y_i) \right] - \mathcal{D}_t(\alpha) \quad (5.31)$$

$$= \left[-\frac{t\lambda}{2} \|s\mathbf{w}^t\|^2 + s \left(\frac{t\lambda}{2} \|\mathbf{w}^t\|^2 + \mathcal{D}_t(\alpha) \right) \right] - \mathcal{D}_t(\alpha) \quad (5.32)$$

$$= \left[-\frac{t\lambda}{2} \frac{s^2}{\lambda s^2} + s \left(\frac{t\lambda}{2} \frac{1}{\lambda s^2} + \mathcal{D}_t(\alpha) \right) \right] - \mathcal{D}_t(\alpha) \quad (5.33)$$

$$= -\frac{t}{2} + \frac{t}{2s} + (s-1)\mathcal{D}_t(\alpha) \quad (5.34)$$

$$= (1-s) \left(\frac{t}{2s} - \mathcal{D}_t(\alpha) \right) \quad (5.35)$$

$$\geq 0 \quad (5.36)$$

where the last line follows because $1-s > 0$ and $\mathcal{D}_t(\alpha) \leq \frac{t}{2s}$:

$$\mathcal{D}_t(\alpha) = -\frac{t\lambda}{2} \|\mathbf{w}^t\|^2 + \sum_{i,Y} \alpha_i^Y \Delta(Y, Y_i) \quad (5.37)$$

$$= -\frac{t\lambda}{2} \frac{1}{\lambda s^2} + \sum_{i,Y} \alpha_i^Y \Delta(Y, Y_i) \quad (5.38)$$

$$= -\frac{t}{2s^2} + \sum_{i,Y} \alpha_i^Y \Delta(Y, Y_i) \quad (5.39)$$

$$\leq -\frac{t}{2s^2} + t \leq -\frac{t}{2} + t \leq \frac{t}{2s} \quad (5.40)$$

where we have assumed $\Delta(Y, Y_i) \leq 1$.

Lemma 5.3.2 *The magnitude of the gradient of the structured SVM error $\nabla f(\mathbf{w}^{t-1}; Z_i)$ in Algorithm 4 is bounded $\|\nabla f(\mathbf{w}^{t-1}; Z_t)\| \leq \sqrt{\lambda} + 2R$, where $\|\Psi(X, Y)\| \leq R$ is a bound on the image of Ψ .*

Proof Since $\nabla \ell(\mathbf{w}^{t-1}; Z_t) = \Psi(X_t, \bar{Y}_t) - \Psi(X_t, Y_t)$ and $\|\Psi(X, Y)\| \leq R$ for all X, Y , it must be the case that $\|\nabla \ell(\mathbf{w}^{t-1}; Z_t)\| \leq 2R$. Line 6 of Algorithm 4 ensures that $\|\mathbf{w}^{t-1}\| \leq \frac{1}{\sqrt{\lambda}}$. Since $\nabla f(\mathbf{w}^{t-1}; Z_t) = \lambda \mathbf{w}^{t-1} + \nabla \ell(\mathbf{w}^{t-1}; Z_t)$, by the triangle inequality, $\|\nabla f(\mathbf{w}^{t-1}; Z_t)\| \leq \sqrt{\lambda} + 2R$.

Lemma 5.3.3 *Let $g(X; \mathbf{w}) = \arg \max_Y \langle \mathbf{w}, \Psi(X, Y) \rangle$ be the label predicted by model parameters \mathbf{w} . The loss associated with this prediction is upper-bounded by the structured hinge loss: $\ell(\mathbf{w}; Z) \geq \Delta(g(X; \mathbf{w}), Y)$*

Proof

$$\begin{aligned} \max_{Y'} (\langle \mathbf{w}, \Psi(X, Y') \rangle + \Delta(Y', Y)) &\geq \langle \mathbf{w}, \Psi(X, g(X; \mathbf{w})) \rangle + \Delta(g(X; \mathbf{w}), Y) \\ \max_{Y'} (\langle \mathbf{w}, \Psi(X, Y') \rangle + \Delta(Y', Y)) &\geq \langle \mathbf{w}, \Psi(X, Y) \rangle + \Delta(g(X; \mathbf{w}), Y) \\ \max_{Y'} (\langle \mathbf{w}, \Psi(X, Y') \rangle + \Delta(Y', Y)) - \langle \mathbf{w}, \Psi(X, Y) \rangle &\geq \Delta(g(X; \mathbf{w}), Y) \\ \ell(\mathbf{w}; Z) &\geq \Delta(g(X; \mathbf{w}), Y) \end{aligned}$$

Theorem 5.3.4 (Logarithmic Regret) *Let $f(\mathbf{w}; Z)$ be the structured SVM objective defined in Eqn 1.7. Let $L = \sqrt{\lambda} + 2R$, where $\|\Psi(X, Y)\| \leq R$ is a bound on the image of Ψ . If Algorithm 4 is run for T iterations, The average loss accumulated during online learning can be bounded in relation to the minimum achievable training error:*

$$\frac{1}{T} \sum_{t=1}^T \Delta(g(X_t; \mathbf{w}^{t-1}), Y_t) \leq \frac{1}{T} \min_{\mathbf{w}} F_T(\mathbf{w}) + \frac{E_{\text{regret}}^T}{T} \quad (5.41)$$

$$\leq \frac{1}{T} \min_{\mathbf{w}} F_T(\mathbf{w}) + \frac{L^2 (\log(T) + 1)}{2\lambda T} \quad (5.42)$$

Proof This proof below closely follows [36]. We begin with the definition of $\frac{E_{\text{regret}}^T}{T}$ from Eq 5.10:

$$\frac{E_{\text{regret}}^T}{T} = \frac{1}{T} \sum_{t=1}^T f(\mathbf{w}^{t-1}; Z_t) - \frac{1}{T} \sum_{t=1}^T \Delta \mathcal{D}_t \quad (5.43)$$

Rearranging terms and then applying the weak duality theorem:

$$\frac{1}{T} \sum_{t=1}^T f(\mathbf{w}^{t-1}; Z_t) = \frac{1}{T} \sum_{t=1}^T \Delta \mathcal{D}_t + \frac{E_{\text{regret}}^T}{T} \quad (5.44)$$

$$= \frac{1}{T} \mathcal{D}_T + \frac{E_{\text{regret}}^T}{T} \quad (5.45)$$

$$\leq \frac{1}{T} \min_{\mathbf{w}} F_T(\mathbf{w}) + \frac{E_{\text{regret}}^T}{T} \quad (5.46)$$

Applying Lemma 5.3.3, we can prove the first part of Thm 5.3.4:

$$\frac{1}{T} \sum_{t=1}^T \Delta(g(X_t; \mathbf{w}^{t-1}), Y_t) \leq \frac{1}{T} \min_{\mathbf{w}} F_T(\mathbf{w}) + \frac{E_{\text{regret}}^T}{T} \quad (5.47)$$

We now need only prove that $\frac{E_{regret}^T}{T} \leq \frac{L^2(\log(T)+1)}{2\lambda T}$. In the context of Algorithm 4, $\Delta\mathcal{D}_t$ satisfies

$$\Delta\mathcal{D}_t \geq \frac{\lambda(t-1)}{2t} \|\mathbf{w}^{t-1}\|^2 + \alpha_{\bar{Y}_t} \frac{t-1}{t} \langle \mathbf{w}^{t-1}, \mathbf{v}_t^{\bar{Y}_t} \rangle - \frac{(\alpha_{\bar{Y}_t})^2}{2\lambda t} \|\mathbf{v}_t^{\bar{Y}_t}\|^2 + \alpha_{\bar{Y}_t} \Delta(\bar{Y}_t, Y_t)$$

The inequality follows by taking the definition of $\Delta\mathcal{D}_t$ in Line 5 of Algorithm 1 and then applying Lemma 5.3.1 (the projection step in Line 6 can only increase $\Delta\mathcal{D}_t$). Since, $\alpha_{\bar{Y}_t}$ is chosen to maximize $\Delta\mathcal{D}_t$, the inequality still holds if we plugin a particular value $\alpha_{\bar{Y}_t} = 1$:

$$\begin{aligned} \Delta\mathcal{D}_t &\geq \frac{\lambda(t-1)}{2t} \|\mathbf{w}^{t-1}\|^2 + \frac{t-1}{t} \langle \mathbf{w}^{t-1}, \mathbf{v}_t^{\bar{Y}_t} \rangle - \frac{1}{2\lambda t} \|\mathbf{v}_t^{\bar{Y}_t}\|^2 + \Delta(\bar{Y}_t, Y_t) \\ &= \frac{\lambda}{2} \|\mathbf{w}^{t-1}\|^2 + \langle \mathbf{w}^{t-1}, \mathbf{v}_t^{\bar{Y}_t} \rangle + \Delta(\bar{Y}_t, Y_t) - \\ &\quad \frac{1}{2\lambda t} \|\mathbf{v}_t^{\bar{Y}_t}\|^2 - \frac{\lambda}{2t} \|\mathbf{w}^{t-1}\|^2 - \frac{1}{t} \langle \mathbf{w}^{t-1}, \mathbf{v}_t^{\bar{Y}_t} \rangle \\ &= f(\mathbf{w}^{t-1}; Z_t) - \frac{1}{2\lambda t} \|\mathbf{v}_t^{\bar{Y}_t}\|^2 - \frac{\lambda}{2t} \|\mathbf{w}^{t-1}\|^2 - \frac{1}{t} \langle \mathbf{w}^{t-1}, \mathbf{v}_t^{\bar{Y}_t} \rangle \\ &= f(\mathbf{w}^{t-1}; Z_t) - \frac{1}{2\lambda t} \|\mathbf{v}_t^{\bar{Y}_t} + \lambda \mathbf{w}^{t-1}\|^2 \\ &= f(\mathbf{w}^{t-1}; Z_t) - \frac{1}{2\lambda t} \|\nabla f(\mathbf{w}^{t-1}; Z_t)\|^2 \end{aligned}$$

Plugging this into Eq 5.43:

$$\begin{aligned} \frac{E_{regret}^T}{T} &\leq \frac{1}{T} \sum_{t=1}^T f(\mathbf{w}^{t-1}; Z_t) - \frac{1}{T} \sum_{t=1}^T \left(f(\mathbf{w}^{t-1}; Z_t) - \frac{1}{2\lambda t} \|\nabla f(\mathbf{w}^{t-1}; Z_t)\|^2 \right) \\ &= \frac{1}{T} \sum_{t=1}^T \frac{1}{2\lambda t} \|\nabla f(\mathbf{w}^{t-1}; Z_t)\|^2 \end{aligned}$$

Applying Lemma 5.3.2, it follows that

$$\frac{E_{regret}^T}{T} \leq \frac{1}{T} \sum_{t=1}^T \frac{1}{2\lambda t} L^2$$

Using the fact that $\sum_{t=1}^T \frac{1}{t} \leq \log(T) + 1$ completes the proof:

$$\frac{E_{regret}^T}{T} \leq \frac{L^2(\log(T) + 1)}{2\lambda T}$$

Thm 5.2.1.1 The proof can be directly taken from the proof of Theorems 2 and 3 in [63]

Thm 5.2.1.2 Since $f(\mathbf{w})$ is convex, by Jensen's inequality

$$\sum_{t=1}^T f(\bar{\mathbf{w}}; Z_t) \leq \sum_{t=1}^T f(\mathbf{w}^{t-1}; Z_t) \quad (5.48)$$

Since Algorithm 1 iterates over each example m times, $\frac{1}{n}F_n(\bar{\mathbf{w}}; Z_i) = \frac{1}{T} \sum_{t=1}^T f(\bar{\mathbf{w}}; Z_t)$.

Applying Thm 5.3.4 completes the proof:

$$\frac{1}{n}F_n(\bar{\mathbf{w}}) \leq \frac{1}{T} \sum_{t=1}^T f(\mathbf{w}^{t-1}; Z_t) \quad (5.49)$$

$$\leq \left(\frac{1}{n} \min_{\mathbf{w}} F_n(\mathbf{w}) \right) + \frac{E_{regret}^T}{T} \quad (5.50)$$

$$\leq \left(\frac{1}{n} \min_{\mathbf{w}} F_n(\mathbf{w}) \right) + \frac{L^2(\log(mn) + 1)}{2\lambda mn} \quad (5.51)$$

5.4 Online Dual Update Step

In this section, we derive the update step in Algorithm 5, which corresponds to solving for a step size $\alpha_t^{\bar{Y}_t}$ on the sub-gradient $\nabla \ell(\mathbf{w}^{t-1}; Z_t)$ which maximally increases the dual problem to the structured SVM error function.

In the context of Line 5 of Algorithm 4, we have already picked a training example Z_t and maximally violated label \bar{Y}_t and seek an assignment to the dual parameter $\alpha_t^{\bar{Y}_t}$ that maximizes the dual objective:

$$\begin{aligned} \Delta \mathcal{D}_t &= \mathcal{D}_t(\alpha_1 \dots \alpha_{t-1}, \alpha_t^{\bar{Y}_t}) - \mathcal{D}_{t-1}(\alpha_1 \dots \alpha_{t-1}) \\ &= \left[-\frac{\lambda t}{2} \|\mathbf{w}^t\|^2 + \sum_{i=1 \dots t, Y} \alpha_i^Y \Delta(Y, Y_i) \right] - \\ &\quad \left[-\frac{\lambda(t-1)}{2} \|\mathbf{w}^{t-1}\|^2 + \sum_{i=1 \dots t-1, Y} \alpha_i^Y \Delta(Y, Y_i) \right] \\ &= -\frac{t\lambda}{2} \|\mathbf{w}^t\|^2 + \frac{(t-1)\lambda}{2} \|\mathbf{w}^{t-1}\|^2 + \alpha_t^{\bar{Y}_t} \Delta(\bar{Y}_t, Y_t) \\ &= -\frac{\lambda t}{2} \left\| \frac{t-1}{t} \mathbf{w}^{t-1} - \frac{\alpha_t^{\bar{Y}_t}}{\lambda t} \mathbf{v}_t^{\bar{Y}_t} \right\|^2 + \frac{\lambda(t-1)}{2} \|\mathbf{w}^{t-1}\|^2 + \alpha_t^{\bar{Y}_t} \Delta(\bar{Y}_t, Y_t) \\ &= \frac{\lambda(t-1)}{2t} \|\mathbf{w}^{t-1}\|^2 + \alpha_t^{\bar{Y}_t} \frac{t-1}{t} \langle \mathbf{w}^{t-1}, \mathbf{v}_t^{\bar{Y}_t} \rangle - \frac{(\alpha_t^{\bar{Y}_t})^2}{2\lambda t} \|\mathbf{v}_t^{\bar{Y}_t}\|^2 + \alpha_t^{\bar{Y}_t} \Delta(\bar{Y}_t, Y_t) \end{aligned}$$

This is maximized by setting $\alpha_t^{\bar{Y}_t}$ to:

$$\alpha_t^{\bar{Y}_t} = \min \left(1, \max \left(0, \frac{\lambda(t-1) \langle \mathbf{w}^{t-1}, \mathbf{v}_t^{\bar{Y}_t} \rangle + \lambda t \Delta(\bar{Y}_t, Y_t)}{\|\mathbf{v}_t^{\bar{Y}_t}\|^2} \right) \right) \quad (5.52)$$

Acknowledgements

This chapter is based on work in progress by the author of this dissertation.

Algorithm 4 Online Structured Learning and Annotation

COMPUTER:

- 1: Initialize: $n \leftarrow 0, t \leftarrow 1, \mathbf{w}^0 \leftarrow \mathbf{0}, E_{train}^0 \leftarrow 0, E_{test}^0 \leftarrow 0, E_{model} \leftarrow 0$
- 2: **repeat**
- 3: Choose example $Z_t = (X_i, Y_i)$ processed the fewest times, $i = \arg \min_{j \in 1 \dots n} |S_j|$
- 4: Suffer loss $f(\mathbf{w}^{t-1}; Z_t) = \frac{\lambda}{2} \|\mathbf{w}^{t-1}\|^2 + \langle \mathbf{w}^{t-1}, \mathbf{v}_t^{\bar{Y}_t} \rangle + \Delta(\bar{Y}_t, Y_i)$, where

$$\bar{Y}_t \leftarrow \arg \max_Y (\langle \mathbf{w}^{t-1}, \Psi(X_i, Y) \rangle + \Delta(Y, Y_i))$$

$$\mathbf{v}_t^{\bar{Y}_t} \leftarrow \Psi(X_i, \bar{Y}_t) - \Psi(X_i, Y_i)$$
- 5: Update weights using Algorithm 7 or Eq 6.12 w/ projective step (Eq 6.15).
- 6: Compute Diagnostics:

$$E_{train}^t \leftarrow E_{train}^{t-1} + f(\mathbf{w}^{t-1}; Z_t)$$
 If $|S_i| = 0$, $E_{test}^n \leftarrow E_{test}^{n-1} + f(\mathbf{w}^{t-1}; Z_t)$

$$E_{model}^t \leftarrow E_{model}^{t-1} + \Delta \mathcal{D}_t$$
 If $\frac{E_{model}^t}{t} > \epsilon$, model/feature space has saturated
 If $\frac{E_{train}^t - E_{model}^t}{t} < \frac{\epsilon}{10}$ and $\forall_{i,j}, |S_i| = |S_j|$, optimiz. error is small (can sleep)
 If $\frac{E_{test}^n}{n} \leq \epsilon$, test error is small (can stop labeling data and training)
- 7: $S_i \leftarrow S_i \cup t, t \leftarrow t + 1$
- 8: **until** $\frac{E_{test}^n}{n} \leq \epsilon$

ANNOTATOR(S):

- 1: **repeat**
 - 2: Label new example (X_{n+1}, Y_{n+1})
 - 3: $S_{n+1} \leftarrow \emptyset, n \leftarrow n + 1$
 - 4: **until** $\frac{E_{test}^n}{n} \leq \epsilon$
-

Algorithm 5 Online Updates

CHANGEALPHA($j, \alpha_j^{\bar{Y}_j^*}$)

- 1: $d \leftarrow \alpha_j^{\bar{Y}_j^*} - \alpha_j^{\bar{Y}_j}$
- 2: $\Delta \mathcal{D}_t^\alpha \leftarrow d \left(\langle \mathbf{w}^t, \mathbf{v}_j^{\bar{Y}_j} \rangle + \Delta(\bar{Y}_j, Y_j) \right) - \frac{\left(d \mathbf{v}_j^{\bar{Y}_j} \right)^2}{2\lambda t}$
- 3: $\mathbf{w}^t \leftarrow \mathbf{w}^t - \frac{d}{\lambda t} \mathbf{v}_j^{\bar{Y}_j}$
- 4: $E_{model}^t \leftarrow E_{model}^t + \Delta \mathcal{D}_t^\alpha$
- 5: $\alpha_j^{\bar{Y}_j} \leftarrow \alpha_j^{\bar{Y}_j^*}$

REMOVEEXAMPLE(i)

- 1: **for** $j \in S_i$ **do**
- 2: **CHANGEALPHA**($j, 0$)
- 3: $w^t \leftarrow \frac{t-1}{t} w^t$
- 4: $t \leftarrow t - 1$
- 5: **end for**

RELABELEXAMPLE(i, Y_i^*, Δ^*)

- 1: **For** $j \in S_i$: **CHANGEALPHA**($j, 0$)
- 2: $Y_i \leftarrow Y_i^*$
- 3: $\Delta(Y, Y_i) \leftarrow \Delta^*$
- 4: **For** $j \in S_i$: **OPTIMIZEALPHA**(j)

OPTIMIZEALPHA(i)

- 1: **CHANGEALPHA** $\left(i, \min \left(1, \max \left(0, \frac{\alpha_i^{\bar{Y}_i} + \lambda t (\langle \mathbf{w}^t, \mathbf{v}_i^{\bar{Y}_i} \rangle + \Delta(\bar{Y}_i, Y_i))}{\|\mathbf{v}_i^{\bar{Y}_i}\|^2} \right) \right) \right)$

AUGMENTFEATURESPACE()

- 1: **For** each $i \in 1..t$, compute example weights β_i using Eq 5.24 or Eq 5.25
 - 2: **Choose best new feature**: $j \leftarrow \arg \max_j \left| \sum_{i=1}^t \beta_i (\phi_j(X_i, \bar{Y}_i) - \phi_j(X_i, Y_i)) \right|$
 - 3: **For** each $i \in 1..t$, **append new feature**: $\mathbf{v}_i^{\bar{Y}_i} \leftarrow [\mathbf{v}_i^{\bar{Y}_i}, \phi_j(X_i, \bar{Y}_i) - \phi_j(X_i, Y_i)]$
 - 4: **Append weight vector**:

$$w_j \leftarrow -\frac{1}{\lambda t} \sum_{i=1}^t \alpha_i^{\bar{Y}_i} (\phi_j(X_i, \bar{Y}_i) - \phi_j(X_i, Y_i))$$

$$\mathbf{w}^t \leftarrow [\mathbf{w}^t, w_j]$$
 - 5: $E_{model}^t \leftarrow E_{model}^t - \frac{\lambda t}{2} w_j^2$
 - 6: **For** each $i \in 1..t$: **OPTIMIZEALPHA**(i)
-

Chapter 6

Faster Customized Optimization Algorithms For Computer Vision Learning Problems

In this chapter, we introduce customized optimization algorithms for common learning problems in computer vision, including object detection, deformable part models, and cost sensitive multiclass classification. It is based on expanding sequential minimization optimization (SMO) and sub-gradient optimization techniques for structured SVMs, while better incorporating application specific knowledge. It is computationally scalable to very large datasets and complex structural representations. We show that structured learning is at least as fast—and often much faster—than methods based on binary classification for problems such as deformable part models, object detection, and multiclass classification, while achieving accuracies that are at least as good. We demonstrate fast train times on two challenging large-scale datasets for two very different problems: ImageNet for multiclass classification and CUB-200-2011 for deformable part model training. Our method is shown to be 10-50 times faster than SVM^{struct} for cost-sensitive multiclass classification while being about as fast as the fastest 1-vs-all methods for multiclass classification. For deformable part model training, it is shown to be 100-1000 times faster than methods based on SVM^{struct} or mining hard negatives and about 50 times faster than other largescale structured SVM solvers based on stochastic gradient descent. It has been integrated into a common optimization package

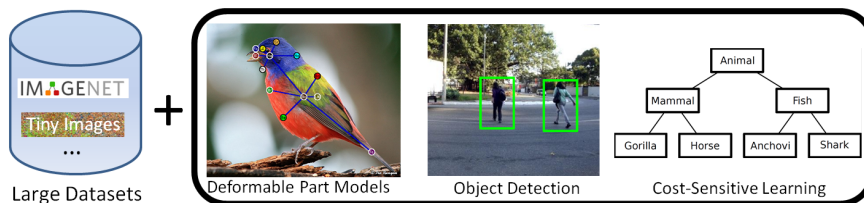


Figure 6.1: We introduce efficient optimization that make structured learning computationally tractable to very large datasets, significantly reducing train time for deformable part models, object detection, and cost-sensitive multiclass learning

with the active labeling techniques described in Chapter 4 and the interactive feedback mechanisms for diagnosing different sources of test error described in Chapter 5.

6.1 Introduction

Over the last twenty years, binary classification algorithms such as SVMs [13], boosting [30], random forests [11] have become very refined and tested. It is common practice to try to map computer vision problems into binary classification problems, which can then be solved using black box optimization algorithms. For example, sliding window object detection problems and deformable part models can be trained by drawing random bounding box locations or part location assignments as negative examples and multiclass classification problems can be solved by combining multiple 1-vs-rest binary classification problems. The prevalence of binary classification methods is perhaps more a consequence of convenience than anything else. It doesn't necessarily have inherently superior theoretical properties to other problem formulations in terms of computational properties or statistical convergence rates. Instead it helps preserve a more modular research decomposition between machine learning, optimization, and applied areas like computer vision.

Nevertheless, computer vision specific learning problems such as training object detectors and deformable part models have become so common place that it is worthwhile to customize optimization algorithms to these specific applications. This is especially true because these types of problems are inherently very computationally intensive and computation time is often the biggest bottleneck toward moving to larger training

sets or more sophisticated representations. Mapping such problems into binary classification problems can increase training time by discarding structural properties of the problem that are exploitable for faster optimization. For example, for multiclass classification, we know all output classes are mutually exclusive. For object detection, we know all possible bounding box locations occur on a 2D grid. For part-based detection, we know the spatial relationship between parts. The improved computational efficiency comes from concentrating processing on output labels (*e.g.*, bounding box locations) that have greater training error. The potential performance gains for structured learning become larger & larger as the problem becomes less and less like binary classification—it is a small gain for multiclass classification, a sizable gain for object detection, and a very significant gain for deformable part models.

At the same time, structured learning algorithms such as structured SVMs [71] are often associated with being slow to train in practice. For example, when applied to sliding window object detection, the algorithm used in SVM^{struct} [71] requires running a sliding window detector on each training image every time the detection model is updated. We argue that this slowness is a consequence of the particular optimization algorithms used rather than a fundamental property.

In this chapter, we propose two changes that allow structured SVMs to be at least as fast as their binary SVM counterparts for problems such as object detection, deformable part models, and multiclass classification. First, we apply ideas from online sub-gradient methods [63, 57] and sequential minimization algorithms [25, 38], which have been shown to be faster than the cutting plane algorithm used in the SVM^{struct} package. This improves train time significantly for medium to large-sized training sets. Second, we allow problem-specific knowledge to be injected into the optimization algorithm by incorporating a user-defined importance sampling function. Here, our optimization algorithm takes an update step with respect to a set of intelligently selected output labels (*e.g.*, a set of bounding boxes in a particular image that have high training error with respect to the current detector). This helps alleviate the slowness of cutting plane [71] and sub-gradient methods [57], which involve expensively firing the current object detector only to update the current model with respect to a single bounding box.

We demonstrate our results on ImageNet and CUB-200-2011, two challenging,

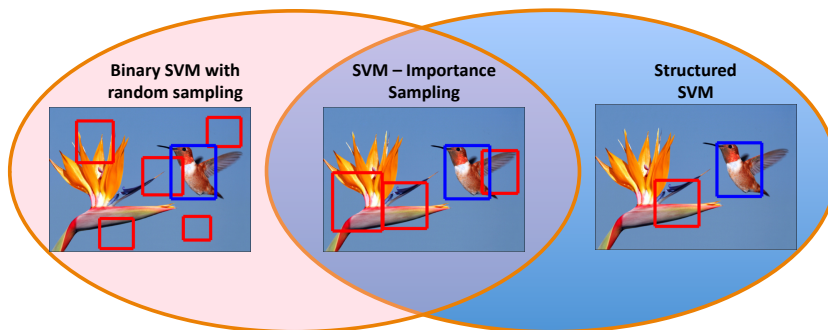


Figure 6.2: Three approaches for training a bird detector: Note that the example image is purposefully chosen as it contains two distinct sets of negative patches; easy from the background and hard from the ‘bird of paradise’ plant. (Left) Training a binary classifier requires a random sampling of negatives, which might be un-informative. (Right) A traditional structured SVM finds, in each iteration, the hardest sample to update the model, which is wasteful since multiple patches contain discriminative information. (Middle) The proposed method, SVM - Importance Sampling, incorporates domain knowledge in the importance sampling routine and inexpensively updates the model with respect to a set of samples in each sequential iteration.

large scale datasets for multiclass classification and deformable part model training, and demonstrate reductions in train time for deformable part model training and cost-sensitive multiclass SVM learning by a factor of ~ 1000 for deformable part model, and ~ 20 for cost-sensitive multiclass classification.

6.2 Background and Related Work

Structured SVMs:

Structured SVMs [66, 71] provide a method of training a system to predict a multidimensional structured output, such as a bounding box, set of part locations, or segmentation. They minimize a convex upper bound on a customizable loss function. Structured SVMs are a superset of SVM-based learning algorithms, which includes many of the most popular and highest performing algorithms for object detection [29, 15, 6], and multiclass classification [42, 18]. They have been successfully applied to many novel problems, including training object detectors using more appropriate loss functions [5], training multiclass SVMs with customizable class confusion costs [12], train-

ing deformable part models with multiple poses and occlusion reasoning [88, 8], training class-attribute structural models [83], and human action segmentation [64].

Binary Classification vs. Structured Prediction:

Problems such as multiclass classification and sliding window object detection are commonly solved by mapping problems into binary classification problems. While this has been successful in practice, there are many arguments for preserving the structural representation, including the ability to optimize more appropriate loss functions and greater generality and simplicity in constructing learning algorithms for new types of problems. We also argue that structured learning models—if one chooses the appropriate optimization algorithms—will be at least as fast to train for a given test accuracy and have at least as good generalization guarantees. It is not inherently easy to optimize 0/1 binary classification loss; this is why popular binary classification algorithms such as boosting, logistic regression, and SVMs optimize a convex upper bound on the 0/1 binary classification loss. Computational speed is thus a consequence of convexity and similar speeds are possible for other convex formulations.

The main theoretical problem with mapping structured problems into binary ones is that the training time can scale exponentially with dimensionality of the structured output space. By contrast, the train time of structured SVMs is known to be independent of the dimensionality of the structured output space when conditioned on other parameters (*e.g.*, regularization, prediction time, feature space magnitude) [57, 67, 71]. For example, object detection can be thought of as a structured prediction problem, where the goal is to predict a bounding box in a 5-dimensional output space: $\{x, y, \text{width}, \text{height}, \text{present}\}$. The space of negative windows per image is an exponential enumeration of this space (there might be $\approx 10^6$ negative windows per image). As a consequence, the error rate of a usable binary detector has to be less than $\epsilon = 10^{-6}$, and it is appropriate for the number of negative training examples per image to grow accordingly (since the number of training examples learning algorithms require usually scales with $\frac{1}{\epsilon}$ or worse). This may be reasonable for relatively simple structured output problems like object detection, but it becomes detrimental for more complicated problems like deformable part models. For example, for a part model with 8 parts, the number of

possible part configurations per image is $\approx 10^{6 \times 8}$. Thus structured learning algorithms have the potential to be significantly faster than their binary classification counterparts when the structural representation is complex.

Mining Hard Negative Examples:

Mining hard negatives is a popular method that is used to help binary classification algorithms get around this exponential scaling [15, 29]. In this method a learned binary classifier is used to extract a new set of negative examples with highest training error, which are added to the training set. The binary classifier is then re-trained after which the process is repeated. This procedure is similar to the cutting plane algorithm used by SVM^{struct} : re-training on hard negatives effectively alters the learning objective from a 0/1 loss toward something that resembles a structured SVM. We will show that this procedure in general has much slower convergence properties than our algorithm; nevertheless, the basic idea of using the current learned model to intelligently sample output labels has a similar flavor to our importance sampling algorithm, and thus a routine to mine hard negatives is usually sufficient to use directly in our optimization algorithm. We differ in terms of the optimization criterion used and the decision of when and how often to sample new output labels versus when to optimize over existing labels. These differences give our method greater understandability, faster train times, and higher performance at convergence.

Fast Solvers For Large Scale Linear SVMs:

Linear kernel SVMs have been shown to have dramatically better computational properties than non-linear kernel SVMs for both training and testing. Whereas non-linear kernel SVMs train in time that is at least quadratic in the number of training examples, linear SVMs train in linear time [35, 25] or in time that does not even depend on the size of the training set (at least in expectation) [63]. Maji et al. showed that many non-linear kernels can be approximated by linear kernels [45, 44]. Among linear SVM solvers, the sequential minimization method of LIBLINEAR [25] and the stochastic gradient descent method of Pegasos [63] have been shown to be significantly faster than the cutting plane algorithm of SVM^{perf} [35]. The main savings comes from updating the

model using more frequent online or sequential update steps as opposed to batch steps.

Solvers For Structured SVMs:

The basic optimization methods used by the above algorithms are general to many convex optimization problems. [57, 59, 8] apply online sub-gradient methods to structured SVMs, yielding a performance improvement over $\text{SVM}^{\text{struct}}$ that is similar to the difference between Pegasos and SVM^{perf} . Kakade and Shalev-Shwartz provided a template algorithm for developing fast optimization algorithms for novel strongly convex optimization problems and a theoretical framework for studying their statistical convergence properties [36]. Analysis of our algorithm is based on this template.

Multiclass SVMs:

Crammer and Singer introduced a multiclass SVM formulation [14] in which weights for each class are learned jointly instead of using multiple 1-vs-all binary classifiers. Keerthi et al. [38] developed a specialized multiclass SVM solver and showed that their algorithm is at least as fast as 1-vs-all methods, with a slight improvement in performance at convergence. When applied to multiclass SVMs, our algorithm is very similar. Beyond [38], our algorithm can be applied to cost-sensitive SVM learning.

6.3 Notation and Algorithm Summary

Structured Learning:

Let X be an input example and $Y = y_1 \dots y_O$ be a multidimensional structured output defining its ground truth label. For example, for deformable part models, X is an image and each $y_p \in Y$ defines the bounding box of a part in the image. A structured prediction function predicts the label with highest score:

$$g(X; \mathbf{w}) = \arg \max_Y \langle \mathbf{w}, \Psi(X, Y) \rangle \quad (6.1)$$

where $\langle \mathbf{w}, \Psi(X, Y) \rangle$ is a discriminative score measuring the goodness of a particular label Y , $\Psi(X, Y)$ is a feature space extracted with respect to label Y (e.g., features extracted around part locations $y_1 \dots y_O$), and \mathbf{w} is a learned vector of weights. Let $\Delta(Y, Y_i)$

be a customizable loss function associated with predicting label Y when the true label is Y_i . For example, $\Delta(Y, Y_i)$ can be a measure of the overlap between predicted part locations and their ground truth locations. Structured SVM learning minimizes the training error over a training set $D_n = Z_1 \dots Z_n$ of instance-label pairs $Z_i = (X_i, Y_i)$:

$$F_n(\mathbf{w}) = n \left(\frac{\lambda}{2} \|\mathbf{w}\|^2 + \frac{1}{n} \sum_{i=1}^n \epsilon_i \right) \quad (6.2)$$

$$\text{s.t.}, \forall_{i,Y}, \langle \mathbf{w}, \Psi(X_i, Y) \rangle + \Delta(Y, Y_i) \leq \langle \mathbf{w}, \Psi(X_i, Y_i) \rangle + \epsilon_i \quad (6.3)$$

where λ is a regularization constant. The objective adds slack variables ϵ_i that place an upper bound on $\Delta(Y, Y_i)$ that is convex in \mathbf{w} . An equivalent way of writing Eq 6.2 is

$$F_n(\mathbf{w}) = \sum_{i=1}^n f(\mathbf{w}; Z_i) \quad (6.4)$$

$$f(\mathbf{w}; Z_i) = \frac{\lambda}{2} \|\mathbf{w}\|^2 + \ell(\mathbf{w}, Z_i) \quad (6.5)$$

$$\ell(\mathbf{w}, Z_i) = \max_Y (\langle \mathbf{w}, \Psi(X_i, Y) \rangle + \Delta(Y, Y_i)) - \langle \mathbf{w}, \Psi(X_i, Y_i) \rangle \quad (6.6)$$

Dual Problem:

Eq 6.4 can be represented by its equivalent dual problem

$$\max_{\alpha_1 \dots \alpha_n} \mathcal{D}_n(\alpha_1 \dots \alpha_n) = \min_{\mathbf{w}} F_n(\mathbf{w}) \quad (6.7)$$

which we will show in the next section is useful for deriving optimization algorithms and theoretical guarantees. $\mathcal{D}_n(\alpha_1 \dots \alpha_n)$ can be written as (see [71])

$$\mathcal{D}_n(\alpha_1 \dots \alpha_n) = -\frac{1}{2\lambda n} \sum_{i,Y} \sum_{j,Y'} \alpha_i^Y \alpha_j^{Y'} \langle \mathbf{v}_i^Y, \mathbf{v}_j^{Y'} \rangle + \sum_{i,Y} \alpha_i^Y \Delta(Y, Y_i) \quad (6.8)$$

$$\text{s.t.}, \forall_i \left(\forall_Y, \alpha_i^Y \geq 0, \sum_Y \alpha_i^Y \leq 1 \right) \quad (6.9)$$

Here, each $\mathbf{v}_i^Y = \Psi(X_i, Y) - \Psi(X_i, Y_i)$ is a vector that is weighted by a corresponding scalar α_i^Y , which encodes a weight on every possible label Y for a training example i . For example, for sliding window based detection, each \mathbf{v}_i^Y is a vector of features extracted at a candidate bounding box location Y and is weighted by a dual parameter α_i^Y . The relationship between the dual parameters α and primal parameters \mathbf{w}^t is

$$\mathbf{w}^t = -\frac{1}{\lambda t} \sum_{i=1}^t \mathbf{u}_i, \quad \mathbf{u}_i = \sum_Y \alpha_i^Y \mathbf{v}_i^Y \quad (6.10)$$

6.4 Optimization Algorithms

Cutting Plane Algorithm:

SVM^{struct} optimizes Eq 6.2 using a cutting plane algorithm. In each iteration, the current model weights are used to solve for the value of Y for each example that maximally violates the constraint in Eq 6.2:

$$\bar{Y}_i = \arg \max_Y (\langle \mathbf{w}, \Psi(X_i, Y) \rangle + \Delta(Y, Y_i)) \quad (6.11)$$

Solving Eq 6.11 resembles a prediction problem (Eq 6.1), and the set of problems that are appropriate for structured SVMs is limited to problems and loss functions for which Eq 6.11 is efficiently solvable. The maximum violated labels \bar{Y}_i are combined into a constraint that is added to an SVM optimization problem. The cutting plane algorithm thus alternates between solving an SVM optimization problem and solving for maximum violated constraints. It takes $O(\frac{R^2 T}{\lambda \epsilon} n)$ time to converge to within ϵ of the minimum achievable training error [71], where T is the time to solve Eq 6.11, R is a bound on the magnitude of the feature space $\|\Psi(X, Y)\| \leq R$, and λ is the regularization constant.

Stochastic Gradient Descent:

[57, 8] optimize Eq 6.2 using a stochastic gradient descent (SGD) algorithm. SGD iterates over each example sequentially, taking an update step:

$$\mathbf{w}^t \leftarrow \mathbf{w}^{t-1} - \frac{1}{\lambda t} \nabla f(\mathbf{w}^{t-1}; Z_t) \quad (6.12)$$

where $\nabla f(\mathbf{w}^{t-1}; Z_t)$ is the sub-gradient of Eq 6.5, and is computed as

$$\nabla f(\mathbf{w}^{t-1}; Z_t) = \lambda \mathbf{w}^{t-1} + \nabla \ell(\mathbf{w}^{t-1}; Z_t) \quad (6.13)$$

$$\nabla \ell(\mathbf{w}^{t-1}; Z_t) = \Psi(X_t, \bar{Y}_t) - \Psi(X_t, Y_t) \quad (6.14)$$

where \bar{Y}_t is computed as in Eq 6.11. Adding a projective step similar to the one used in Pegasos [63]

$$\mathbf{w}^t \leftarrow \frac{\min(1/\sqrt{\lambda}, \|\mathbf{w}^t\|)}{\|\mathbf{w}^t\|} \mathbf{w}^t \quad (6.15)$$

ensures that this algorithm takes $\tilde{O}(\frac{R^2T}{\lambda\epsilon})$ time to converge to within ϵ of the minimum achievable training error, assuming one iterates over each training example an equal number of times (see supplementary material).

In the dual formulation of the problem, Eq 6.12 is equivalent to adding a new dual parameter $\mathbf{u}_t = \alpha_t^{\bar{Y}_t} \mathbf{v}_t^{\bar{Y}_t}$ with $\alpha_t^{\bar{Y}_t} = 1$ and $\mathbf{v}_t^{\bar{Y}_t} = \nabla \ell(\mathbf{w}^{t-1}; Z_t)$. Kakade and Shalev-Shwartz [36] proved that for any strongly convex loss function, each iteration of SGD will increase the dual objective $\mathcal{D}_t(\alpha_1 \dots \alpha_t) - \mathcal{D}_{t-1}(\alpha_1 \dots \alpha_{t-1})$ by a predictable amount, and this amount is sufficient to prove the above convergence rates. They observe that any algorithm that increases \mathcal{D}_t by at least as much in each timestep will converge at least as quickly.

Multi Sample Algorithm:

We introduce a sequential dual optimization algorithm that is based on this idea, where our goal is to increase \mathcal{D}_t by as much as possible using an inexpensive update. Similar to SGD, we sequentially update the model \mathbf{w}^t using a single training example Z_t ; however, we also employ an update that is more specialized for structured SVM optimization. In contrast to cutting plane and SGD algorithms which use only a single label \bar{Y}_t per image in each update step, we update using a set of intelligently selected labels $\bar{Y}_t^1, \dots, \bar{Y}_t^K$. For greatest efficiency, $\bar{Y}_t^1, \dots, \bar{Y}_t^K$ should typically be a sparse subset of all possible values for Y (e.g., if one tried to update the model with respect to all possible bounding boxes in the image, the update would be expensive). This subset of samples can be intelligently selected using the current model parameters \mathbf{w}^t and some application specific importance sampling routine. For correctness, \bar{Y}_t should be included in this set of samples.

Our optimization algorithm learns a weight $\alpha_t^{\bar{Y}_t^k}$ on each sample \bar{Y}_t^k by solving

$$\max_{\alpha_t^{\bar{Y}_t^1} \dots \alpha_t^{\bar{Y}_t^K}} \alpha_t = \mathcal{D}_t(\alpha_1 \dots \alpha_{t-1}, \alpha_t^{\bar{Y}_t^1} \dots \alpha_t^{\bar{Y}_t^K}) - \mathcal{D}_{t-1}(\alpha_1 \dots \alpha_{t-1}) \quad (6.16)$$

$$\text{s.t.}, \forall_i \left(\forall_Y, \alpha_i^Y \geq 0, \sum_Y \alpha_i^Y \leq 1 \right) \quad (6.17)$$

Algorithm 6

- 1: **for** $t = 1$ to T **do**
 - 2: Receive an example $Z_t \leftarrow (X_i, Y_i)$ where $i = t \bmod n$
 - 3: $\bar{Y}_t^1 \dots \bar{Y}_t^K \leftarrow \text{IMPORTANCESAMPLE}(X_t, Y_t, \mathbf{w}^{t-1})$
 - 4: Solve for $\alpha_t^{\bar{Y}_t^1} \dots \alpha_t^{\bar{Y}_t^K}$ using Eq 6.23
 - 5: Update \mathbf{w}^t using Eq 6.18
 - 6: Bound \mathbf{w}^t using Eq 6.15
 - 7: **end for**
-

and then updates \mathbf{w}^t according to:

$$\mathbf{w}^t \leftarrow \frac{t-1}{t} \mathbf{w}^{t-1} - \frac{1}{\lambda t} \sum_{k=1}^K \alpha_t^{\bar{Y}_t^k} \mathbf{v}_t^{\bar{Y}_t^k} \quad (6.18)$$

Details of the implementation of Eq 6.23 are included in Section 6.6. Pseudo-code for the algorithm is provided in Algorithm 6.

6.5 Notation

Recall that $\max_{\alpha} \mathcal{D}_t(\alpha) = \min_{\mathbf{w}} F_T(\mathbf{w})$, where $F_T(\mathbf{w})$ is the structured SVM training error and is defined as

$$F_T(\mathbf{w}) = \sum_{t=1}^T f(\mathbf{w}; Z_t), \quad f(\mathbf{w}; Z_t) = \frac{\lambda}{2} \|\mathbf{w}\|^2 + \ell(\mathbf{w}, Z_t) \quad (6.19)$$

$$\ell(\mathbf{w}, Z_t) = \max_Y (\langle \mathbf{w}, \Psi(X_t, Y) \rangle + \Delta(Y, Y_t)) - \langle \mathbf{w}, \Psi(X_t, Y_t) \rangle \quad (6.20)$$

and $\mathcal{D}_t(\alpha)$ is the equivalent dual problem and is defined as

$$\mathcal{D}_t(\alpha) = -\frac{\lambda t}{2} \|\mathbf{w}^t\|^2 + \sum_{i,Y} \alpha_i^Y \Delta(Y, Y_i) \quad (6.21)$$

$$\mathbf{w}^t = -\frac{1}{\lambda t} \sum_{i=1}^t \mathbf{u}_i, \quad \mathbf{u}_i = \sum_Y \alpha_i^Y \mathbf{v}_i^Y, \quad \mathbf{v}_i^Y = \Psi(X_i, Y) - \Psi(X_i, Y_i) \quad (6.22)$$

6.6 Multi-Sample Update

In this section, derive an update step that occurs over a set of samples $\bar{Y}_i = \bar{Y}_i^1, \bar{Y}_i^2, \dots, \bar{Y}_i^K$. For convergence guarantees (see Theorem 5.2.1), we assume that this set

includes the subgradient as the first sample: $\mathbf{v}_t^{\bar{Y}_t^1} = \nabla \ell(\mathbf{w}^{t-1}; Z_t)$. The update solves the optimization problem (exactly or approximately):

$$\alpha_t = \max_{\alpha_t^{\bar{Y}_t^1} \dots \alpha_t^{\bar{Y}_t^K}} \mathcal{D}_t(\alpha_1 \dots \alpha_{t-1}, \alpha_t^{\bar{Y}_t^1} \dots \alpha_t^{\bar{Y}_t^K}) - \mathcal{D}_{t-1}(\alpha_1 \dots \alpha_{t-1}) \quad (6.23)$$

$$\text{s.t.}, \forall_i \left(\forall_j, \alpha_i^{\bar{Y}_i^j} \geq 0, \sum_{j=1}^K \alpha_i^{\bar{Y}_i^j} \leq 1 \right) \quad (6.24)$$

Pseudo-code for the algorithm is shown in Algorithm 6. In this algorithm, each sample \bar{Y}_i^j is iterated over once. We define an iterative procedure, such that each $\alpha_i^{\bar{Y}_i^j}$ is updated in order $j = 1 \dots K$. In each iteration, we solve (in closed form) for the optimal value to set $\alpha_t^{\bar{Y}_i^j}$ which maximizes $\Delta \mathcal{D}_{t,j}^{expand}$:

$$\begin{aligned} \Delta \mathcal{D}_{t,j}^{expand} &= \mathcal{D}_t(\alpha_1 \dots \alpha_{t-1}, \alpha_t^{\bar{Y}_i^j}) - \mathcal{D}_t(\alpha_1 \dots \alpha_{t-1}, 0) \\ &= \frac{-\lambda t}{2} \left\| \mathbf{w}^t - \frac{\alpha_t^{\bar{Y}_i^j}}{\lambda t} \mathbf{v}_t^{\bar{Y}_i^j} \right\|^2 + \frac{\lambda t}{2} \|\mathbf{w}^t\|^2 + \alpha_t^{\bar{Y}_i^j} \Delta(\bar{Y}_i^j, Y_t) \\ &= \alpha_t^{\bar{Y}_i^j} \left(\langle \mathbf{w}^{t,j-1}, \mathbf{v}_t^{\bar{Y}_i^j} \rangle + \Delta(\bar{Y}_i^j, Y_t) \right) - \frac{(\alpha_t^{\bar{Y}_i^j})^2}{2\lambda t} \|\mathbf{v}_t^{\bar{Y}_i^j}\|^2 \end{aligned}$$

This is maximized (by setting the derivative equal to 0) by choosing

$$\alpha_t^{\bar{Y}_i^j} = \frac{\lambda t \left(\langle \mathbf{w}^{t,j-1}, \mathbf{v}_t^{\bar{Y}_i^j} \rangle + \Delta(\bar{Y}_i^j, Y_t) \right)}{\|\mathbf{v}_t^{\bar{Y}_i^j}\|^2}$$

Unfortunately, due to the constraint $\sum_{j=1}^K \alpha_i^{\bar{Y}_i^j} \leq 1$ in Eq 6.23, this update becomes invalid once $\sum_{j=1}^K \alpha_i^{\bar{Y}_i^j} = 1$. We therefore consider an alternate ‘‘swap’’ move, which works by setting $\alpha_i^{\bar{Y}_i^j}$ while simultaneously scaling all parameters $\alpha_i^{\bar{Y}_i^1} \dots \alpha_i^{\bar{Y}_i^{j-1}}$ by $s = 1 - \alpha_i^{\bar{Y}_i^j}$. This swap move preserves the constraint $\sum_{j=1}^K \alpha_i^{\bar{Y}_i^j} = 1$. The subsequent change in the dual objective is:

$$\begin{aligned}
& \Delta \mathcal{D}_{t,j}^{swap} \\
&= \mathcal{D}_t(\alpha_1 \dots \alpha_{t-1}, s\alpha_t^{\bar{Y}_t^1}, \dots, s\alpha_t^{\bar{Y}_t^{j-1}}, \alpha_t^{\bar{Y}_t^j}) - \mathcal{D}_t(\alpha_1 \dots \alpha_{t-1}, s\alpha_t^{\bar{Y}_t^1}, \dots, s\alpha_t^{\bar{Y}_t^{j-1}}, 0) \\
&= \frac{-\lambda t}{2} \left\| \mathbf{w}^t - \frac{(s-1)\mathbf{u}_t^{j-1} + \alpha_t^{\bar{Y}_t^j} \mathbf{v}_t^{\bar{Y}_t^j}}{\lambda t} \right\|^2 + \frac{\lambda t}{2} \|\mathbf{w}^t\|^2 + (s-1)D_t^{j-1} \\
&\quad + \alpha_t^{\bar{Y}_t^j} \Delta(\bar{Y}_t^j, Y_t) \\
&= \frac{-\lambda t}{2} \left\| \mathbf{w}^t - \frac{-\alpha_t^{\bar{Y}_t^j} \mathbf{u}_t^{j-1} + \alpha_t^{\bar{Y}_t^j} \mathbf{v}_t^{\bar{Y}_t^j}}{\lambda t} \right\|^2 + \frac{\lambda t}{2} \|\mathbf{w}^t\|^2 - \alpha_t^{\bar{Y}_t^j} D_t^{j-1} + \alpha_t^{\bar{Y}_t^j} \Delta(\bar{Y}_t^j, Y_t) \\
&= \alpha_t^{\bar{Y}_t^j} \left[\left(\langle \mathbf{w}^t, \mathbf{v}_t^{\bar{Y}_t^j} \rangle + \Delta(\bar{Y}_t^j, Y_t) \right) - \left(\langle \mathbf{w}^t, \mathbf{u}_t^{j-1} \rangle + D_t^{j-1} \right) \right] \\
&\quad + \frac{\left(\alpha_t^{\bar{Y}_t^j} \right)^2}{2\lambda t} \left\| \mathbf{u}_t^{j-1} + \mathbf{v}_t^{\bar{Y}_t^j} \right\|^2
\end{aligned}$$

where D_t^{j-1} is shorthand for

$$D_t^{j-1} = \sum_{k=1}^{j-1} \alpha_t^{\bar{Y}_t^k} \Delta(\bar{Y}_t^k, Y_t) \quad (6.25)$$

and can be interpreted as a (weighted) average loss over all samples. The value of $\alpha_t^{\bar{Y}_t^j}$ which maximizes $\Delta \mathcal{D}_{t,j}^{swap}$ is:

$$\alpha_t^{\bar{Y}_t^j} = \frac{\lambda t \left[\left(\langle \mathbf{w}^t, \mathbf{v}_t^{\bar{Y}_t^j} \rangle + \Delta(\bar{Y}_t^j, Y_t) \right) - \left(\langle \mathbf{w}^t, \mathbf{u}_t^{j-1} \rangle + D_t^{j-1} \right) \right]}{\left\| \mathbf{u}_t^{j-1} + \mathbf{v}_t^{\bar{Y}_t^j} \right\|^2} \quad (6.26)$$

We can compute $\alpha_t^{\bar{Y}_t^j}$ in $O(d)$ time, where d is the dimensionality of the feature space Ψ , if we maintain updated values for \mathbf{w}_t^j , \mathbf{u}_t^j , and D_t^j . The appropriate updates if we were to scale α_t by s and then set $\alpha_t^{\bar{Y}_t^j}$ are:

$$\mathbf{u}_t^j \leftarrow s \mathbf{u}_t^{j-1} + \alpha_t^{\bar{Y}_t^j} \mathbf{v}_t^{\bar{Y}_t^j} \quad (6.27)$$

$$\mathbf{w}^{t,j} \leftarrow \mathbf{w}^{t,j-1} - \frac{(s-1)\mathbf{u}_t^{j-1} + \alpha_t^{\bar{Y}_t^j} \mathbf{v}_t^{\bar{Y}_t^j}}{\lambda t} \quad (6.28)$$

$$D_t^j \leftarrow s D_t^{j-1} + \alpha_t^{\bar{Y}_t^j} \Delta(\bar{Y}_t^j, Y_t) \quad (6.29)$$

Algorithm 7 MULTISAMPLEUPDATE

Input: New example X_t, Y_t , current weights \mathbf{w}^{t-1}
Output: New weights $\mathbf{w}^{t,K}$

- 1: $\bar{Y}_t^1 \dots \bar{Y}_t^K \leftarrow \text{IMPORTANCESAMPLE}(X_t, Y_t, \mathbf{w}^{t-1})$
 - 2: Initialize $\mathbf{w}^{t,0} \leftarrow \frac{t-1}{t} \mathbf{w}^{t-1}$, $\mathbf{u}_t^0 \leftarrow 0$, $D_t^0 \leftarrow 0$, $\alpha_t \leftarrow 0$
 - 3: **for** $j = 1$ to K **do**
 - 4: **if** $\alpha_t < 1$ **then**
 - 5: $\alpha_t^{\bar{Y}_t^j} \leftarrow \min \left(1 - \alpha_t, \max \left(0, \frac{\lambda t \left(\langle \mathbf{w}^{t,j-1}, \mathbf{v}_t^{\bar{Y}_t^j} \rangle + \Delta(\bar{Y}_t^j, Y_t) \right)}{\|\mathbf{v}_t^{\bar{Y}_t^j}\|^2} \right) \right)$
 - 6: $s \leftarrow 1$
 - 7: $\alpha_t \leftarrow \alpha_t + \alpha_t^{\bar{Y}_t^j}$
 - 8: **else**
 - 9: $\alpha_t^{\bar{Y}_t^j} \leftarrow \min \left(1, \max \left(0, \frac{\lambda t \left[\left(\langle \mathbf{w}^{t,j-1}, \mathbf{v}_t^{\bar{Y}_t^j} \rangle + \Delta(\bar{Y}_t^j, Y_t) \right) - \left(\langle \mathbf{w}^{t,j-1}, \mathbf{u}_t^{j-1} \rangle + D_t^{j-1} \right) \right]}{\|\mathbf{u}_t^{j-1} + \mathbf{v}_t^{\bar{Y}_t^j}\|^2} \right) \right)$
 - 10: $s \leftarrow 1 - \alpha_t^{\bar{Y}_t^j}$
 - 11: **end if**
 - 12: $\mathbf{u}_t^j \leftarrow s \mathbf{u}_t^{j-1} + \alpha_t^{\bar{Y}_t^j} \mathbf{v}_t^{\bar{Y}_t^j}$
 - 13: $\mathbf{w}^{t,j} \leftarrow \mathbf{w}^{t,j-1} - \frac{(s-1)\mathbf{u}_t^{j-1} + \alpha_t^{\bar{Y}_t^j} \mathbf{v}_t^{\bar{Y}_t^j}}{\lambda t}$
 - 14: $D_t^j \leftarrow s D_t^{j-1} + \alpha_t^{\bar{Y}_t^j} \Delta(\bar{Y}_t^j, Y_t)$
 - 15: **end for**
 - 16: Optionally repeat steps 3-16 multiple times
-

6.7 Applications

In this section, we show how Algorithm 6 (described in Section 6.6) can be applied to a variety of popular learning problems, including cost-sensitive multiclass SVMs, sliding window object detection, and deformable part models. In particular, we define an importance sampling routine for each method that can be used in conjunction with Algorithm 1.

6.7.1 Cost-Sensitive Multiclass SVMs

Given a training set of image-label pairs $(X_1, Y_1), \dots, (X_n, Y_n)$ where $Y_i \in 1 \dots C$ is a class label, a cost-sensitive multiclass SVM solves the optimization problem:

$$F_n^C(\mathbf{w}) = n \left(\frac{\lambda}{2} \|\mathbf{w}\|^2 + \frac{1}{n} \sum_{i=1}^n \epsilon_i \right) \quad (6.30)$$

$$\text{s.t.}, \forall_{i,c}, \langle \mathbf{w}_c, \phi(X_i) \rangle + \Delta^C(c, Y_i) \leq \langle \mathbf{w}_{Y_i}, \phi(X_i) \rangle + \epsilon_i \quad (6.31)$$

where $\phi(X)$ is a feature vector of length d , each \mathbf{w}_c is a vector of weights for class c , $\mathbf{w} = [\mathbf{w}_1, \dots, \mathbf{w}_C]$ concatenates the weights for all classes, and $\Delta^C(c, Y_i)$ is a confusion cost associated with predicting class c when the true label is Y_i . As in [71], this problem is solvable using a structured SVM, where $\Psi(X, Y)$ concatenates features for each class:

$$\Psi^C(X, Y) = [\psi_1(X, Y) \dots \psi_C(X, Y)] \quad (6.32)$$

$$\psi_c(X, Y) = \begin{cases} \phi(X) & \text{if } Y = c \\ \mathbf{0} & \text{otherwise} \end{cases} \quad (6.33)$$

For this problem, we choose a dense sample set that includes every possible class label $\bar{Y}_t \leftarrow \{1, \dots, C\}$. This is computationally efficient because the multi-sample update step (Eq 6.23) takes $O(dC)$ time, which is the same amount of time it takes to solve for \bar{Y}_i or compute $\nabla \ell(\mathbf{w}, Z_t)$ (the main computations for cutting plane and SGD algorithms). A similar algorithm was presented in [38].

6.7.2 Sliding Window Object Detection

A sliding window object detector can be trained using a structured SVM [5], where $Y = \{x, y, \text{scale}\}$ encodes a bounding box and $\Psi^B(X, Y)$ is a vector of features extracted at Y . Let $\Delta^B(Y, Y_i)$ be an arbitrary loss function associated with predicting bounding box Y when the true bounding box is Y_i . The sliding window detector can be trained by optimizing the structured SVM objective (Eq 6.2). Let Δ_i^B encode all values of $\Delta^B(Y, Y_i)$ into an array, such that $\Delta_i^B[Y] = \Delta^B(Y, Y_i)$. Similarly, let \mathbf{M}_i be an array of sliding window responses, such that $\mathbf{M}_i[Y] = \langle \mathbf{w}, \Psi(X_i, Y) \rangle$, and let $\mathbf{L}_i = \mathbf{M}_i + \Delta_i^B$. Note that

$$\bar{Y}_i = \arg \max_Y (\langle \mathbf{w}, \Psi(X_i, Y) \rangle + \Delta(Y, Y_i)) = \arg \max_{Y'} \mathbf{L}_i[Y'] \quad (6.34)$$

We choose a sparse sample set of bounding boxes $\bar{Y}_t^1 \dots \bar{Y}_t^K$, that are the result of running non-maximal suppression on the response map \mathbf{L}_i . The motivation behind this sampling technique is to suppress overlapping bounding boxes that will tend to be redundant due to similar feature values.

6.7.3 Deformable Part Model Based Detection

A Felzenszwalb-like deformable part model can be trained using a structured SVM, where $Y = y_1, \dots, y_P$ encodes a set of part locations, each of which is represented using a 4-tuple $y_p = \{x_p, y_p, \text{scale}_p, \text{aspect}_p\}$, where aspect_p defines a mixture component index (*e.g.*, it toggles between different appearance and spatial models for side view, frontal view, *etc.*), and parts can have tree-structured dependencies with spring costs connecting parent and child parts. Details of the mapping into structured SVMs are presented in [88, 8]. These methods concatenate appearance features and spatial offsets for all part-aspect pairs into a structured feature space $\Psi^P(X, Y)$. The label \bar{Y}_i can be efficiently solved for using standard dynamic programming algorithms for pictorial structure inference. For our importance sampling routine, we choose a sparse sample set of part location assignments $\bar{Y}_t^1 \dots \bar{Y}_t^K$, that are the result of running non-maximal suppression on the response map of the root of the part tree after dynamic programming. This sampling routine is described in [78].

6.8 Experiments

We demonstrate generality and computational speed on two challenging, large-scale datasets: ImageNet for multiclass classification and CUB-200-2011 for pose registration and part-based detection.

6.8.1 Multiclass SVM

ImageNet is a large image data base introduced by Deng *et al.* in 2009 [18]. In this paper we use the training set from the 1k category ImageNet subset published as part of the Pascal Visual Object Challenge (VOC) 2010, and also a 200 category subset.

We use the SIFT-based bag-of-words feature representation (with 1k visual words) given as part of the Pascal VOC challenge, and consider the hierarchical loss, $C_{i,j}$, proposed by Jia *et al.* [17], where $C_{i,j} = 0$, when $i = j$, and $C_{i,j} = h(i, j)$, the height of the lowest common ancestor otherwise. The proposed method, SVM-IS, is compared to the Liblinear implementation of the Crammer and Singer multiclass formulation, Liblinear 1-vs-all, Pegasos 1-vs-all and SVM^{struct} . While the structured SVM formulations allow for a direct minimization of the hierarchical loss, the other methods need to be made cost sensitive using some heuristic; here we deploy the framework proposed by Jia *et al.* [17]. They note that one can obtain posterior probability estimates by first learning the parameters of a sigmoid function using a hold out set, and then using the learned sigmoid function to transform the SVM decision function values to posterior probabilities [55]. Once the posteriors are thus estimated, the unseen images are classified by the optimal Bayes decision rule: $f(x) = \arg \min_{i=1, \dots, K} K \sum_{j=1}^K C_{i,j} \hat{p}_j(x)$, where $\hat{p}_j(x)$ is the probability that query image x belong to class j . Note that this heuristic is cumbersome to implement, and requires additional train time to learn the sigmoid parameters (this extra time is emphnot included in the figures). We run a three fold cross validation splitting the data; 60% train data, 20% test, and 10% each for sigma calibration and validation sets. We use the validation set to find optimal value of the regularization parameter, $\lambda = \{10^{-3} \dots 10^{-8}\}$ for each solver. For the structured SVM formulations the sigma calibration set was included in the training directly. The results are shown in fig. 6.3, and we note that the Structural SVMs converge at a lower cost and that SVM-IS converges significantly faster than SVM^{struct} .

6.8.2 Part-Based Detection

CUB-200-2011 [80] is a dataset of 11,788 images of birds from 200 different bird species. Images are uncropped and in unconstrained poses in the wild, with a high degree of variability in shape and appearance. 15 different parts are labelled in each image by an x, y coordinate and a binary variable specifying whether or not each part is visible. We measure performance as the number of correctly predicted part locations, the same metric used in [8], where a predicted part location is considered to be correct if we correctly predict its visibility and its x,y-location is within some radius of the ground

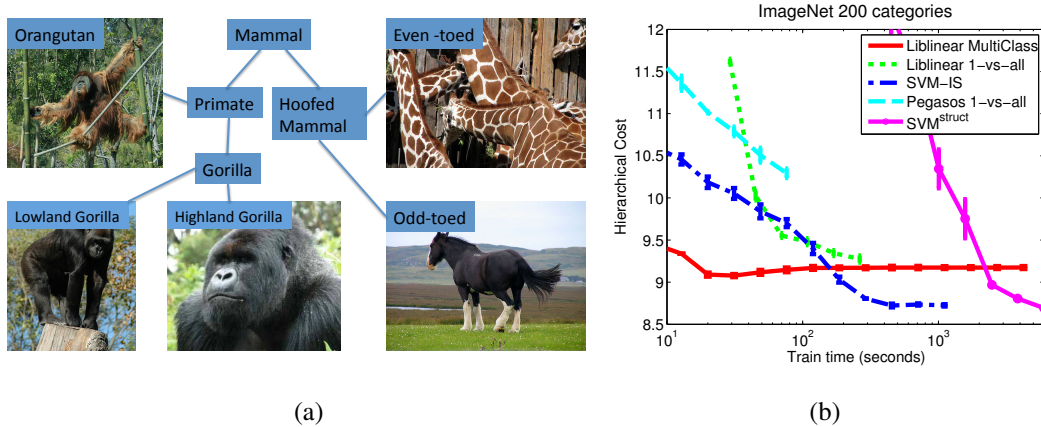


Figure 6.3: Results on ImageNet Rand 200: ImageNet has a strong hierarchy as illustrated in fig. 6.3(a), which motivates the hierarchical loss function – clearly, the misclassification between the gorilla sub-species is less severe than that of horse - giraffe. Figure 6.3(b) show that the structural SVM formulations better optimize this loss, even after post processing as described in sec. 6.8.1 is performed on the non - structural SVMs. Note also that the proposed method, SVM-IS, converges significantly faster than SVM^{struct} .

truth location (defined as within 1.5 standard deviations of where MTurker responses for that part).

We use 12 different aspects (mixture components) for each of the 15 parts, and also predict part visibility, which is modelled as a mixture component with no appearance features. Each mixture component is represented by a 7X7 HOG template and spring costs between neighboring parts. We allow the head and body to have different poses/aspects (*e.g.* the head can be in left side-view while the body is in frontal view). The dimensionality of the feature space is $\Psi(X_i, Y)$ 81,902. Since the model is very complex, the feature space is high-dimensional, and the training set size is moderately large (5,894 images were used for training and 5,894 for testing), the dataset is very computationally challenging.

We compare to two variants of SVM^{struct} [71], an SGD solver that can be seen as an extension of Pegasos to structured SVMs and is used in [8, 57], and a method based on mining hard negatives, where on each round we augment a binary training set with one hard negative per image. Results are shown in Fig 6.4. We can see that our solver significantly outperforms other methods.

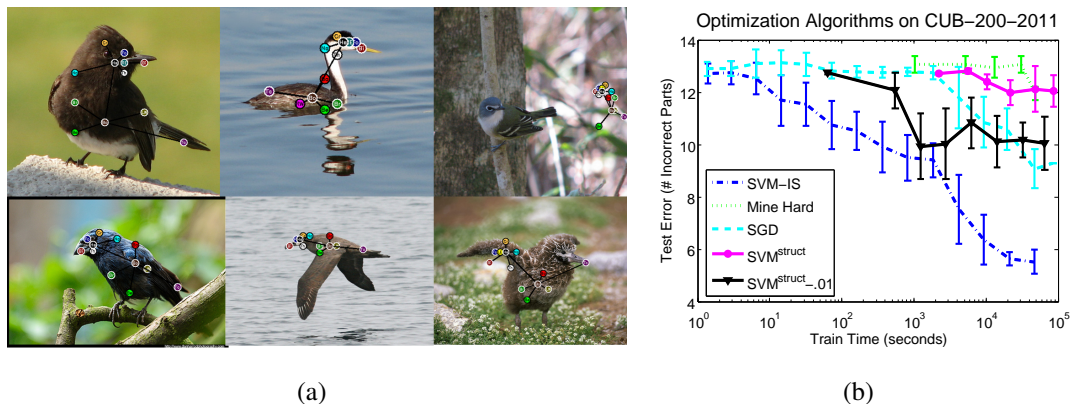


Figure 6.4: Results on CUB-200-2011: (a) Typical part-detection results on CUB-200-2011, with failure cases in the right column. Note that our detector works across extreme variations in pose. (b) Our method SVM-IS converges significantly faster than all other methods. Batch algorithms (SVM^{struct} and mining hard negatives) make little progress in the allotted time (the entire plot shown represents only 25 passes through the training set). Our method yields roughly 50 times speedup over the already fast SGD solver [8]. Timing results are on a single core 2.7GHz Intel Xeon CPU.

6.8.3 Analysis of Results

Batch vs. Online Algorithms:

Methods that employ updates in batches the size of the training set are slower by an asymptotic factor that scales with training set size. This includes SVM^{struct} (shown as a magenta curve in Fig 6.3 and Fig 6.4), which takes roughly 30 times longer to converge in the ImageNet experiment and 100-1000 times longer in the CUB-200-2011 experiment. The method based on mining hard negatives also processes the dataset in batch and is slow to make progress. We also experiment with a variant of SVM^{struct} that is plotted as a black curve in Fig 6.4. This corresponds to a cutting plane algorithm that extracts small batches (1% of the training set) that are added as constraints an SVM optimization problem. The method initially progresses much more quickly than the default SVM^{struct} algorithm, but eventually grinds to a halt as the set of constraints becomes large and SVM optimization slows.

Single sample vs. multi sample:

The multi-sample update reduces training time by a factor of approximately 50 compared to the already fast SGD solver in the CUB-200-2011 experiment. A similar result is visible for the ImageNet SVM experiment, where the LIBLINEAR-multiclass optimizer is slightly faster than the LIBLINEAR-1-vs-all optimizer.

6.9 Conclusion

We introduced a fast structured SVM solver that is shown to be significantly faster than existing methods based on SVM^{struct} , mining hard negatives, or Stochastic Gradient Descent. It reduces train time by a factor of 20-1000 for cost-sensitive multiclass learning and deformable part model training on Imagenet and CUB-200-2011. In future work, we plan on running more extensive object detection experiments. We would also like to apply our algorithm to larger datasets, stronger alignment models, and other types of problems like tracking, segmentation, and attribute vocabulary learning.

Acknowledgements

This chapter is based on work with Oscar Beijbom and Serge Belongie that will be submitted to CVPR 2013. The dissertation author contributed to developing the algorithms and experiments and writing the paper.

Chapter 7

Example Applications

In this chapter, I show how several different computer vision application problems can be plugged into the structured learning framework described in this paper. In general, an application can be supported if there exists an efficient solver for the following 3 related problems:

$$\text{Prediction : } Y^* = \arg \max_Y \langle \mathbf{w}, \Psi(X, Y) \rangle \quad (7.1)$$

$$\text{Max Violated Constraint : } \bar{Y} = \arg \max_Y (\langle \mathbf{w}, \Psi(X, Y) \rangle + \Delta(Y, Y_i)) \quad (7.2)$$

$$\text{Interactive Prediction : } Y_{\tilde{Y}}^* = \arg \max_Y \langle \mathbf{w}, \Psi(X, Y) \rangle \quad (7.3)$$

$$\text{s.t. } \forall_{\tilde{y}_i \in \tilde{Y}}, \tilde{y}_i = y_i$$

where solving Eq 7.1 is used for structured prediction at test time, solving Eq 7.2 will facilitate efficient structured learning, and Eq 7.3 will allow for interactive interfaces that can reduce human annotation time during training or offer new applications. For many or most applications, existence of an efficient prediction algorithm will imply existence of an efficient algorithm for the other two, as we will see in the examples below. The purpose of this section is to show how a variety of popular computer vision problems can be mapped into a common problem formulation that is supported by our software package (see <http://vision.ucsd.edu/visipedia/code>). As a consequence of being mapped into this formulation, one obtains a number of practical benefits:

- Fast optimization algorithms that are scalable to large datasets and often faster than the optimization packages that are most commonly used in computer vision

(see Chapter 6).

- An active learning technique that employs interactive labeling and online learning (see Chapter 4), while being computationally practical for complex image representations and large datasets.
- Feedback mechanisms for diagnosing different sources of test error (*e.g.*, insufficient number of training examples, a bad model or feature space, mislabeled training examples, or insufficient computation time), and correcting the result in online fashion without necessitating retraining (*e.g.*, adding training examples, changing the feature space, relabeling examples, or spending more computation time)

7.1 Cost-Sensitive Multiclass SVMs:

Given a training set of image-label pairs $(X_1, Y_1), \dots, (X_n, Y_n)$ where each $Y_i \in 1 \dots C$ is a class label, a cost-sensitive multiclass SVM solves the optimization problem:

$$F_n^C(\mathbf{w}) = n \left(\frac{\lambda}{2} \|\mathbf{w}\|^2 + \frac{1}{n} \sum_{i=1}^n \epsilon_i \right) \quad (7.4)$$

$$\text{s.t.}, \forall_{i,c}, \langle \mathbf{w}_c, \phi(X_i) \rangle + \Delta^C(c, Y_i) \leq \langle \mathbf{w}_{Y_i}, \phi(X_i) \rangle + \epsilon_i \quad (7.5)$$

where $\phi(X)$ is a feature vector of length d (*e.g.* SIFT, color histogram, bag-of-words descriptor, fisher encoding, *etc.*), each \mathbf{w}_c is a weight vector of length d for class c , $\mathbf{w} = [\mathbf{w}_1, \dots, \mathbf{w}_C]$ concatenates the weights for all classes, and $\Delta^C(c, Y_i)$ is a confusion cost associated with predicting class c when the true label is Y_i . For example, $\Delta^C(c, Y_i)$ could encode a taxonomical loss (given a class taxonomy, $\Delta^C(c, Y_i)$ is the distance from node c to node Y_i on a class taxonomy tree). For a non-cost-sensitive multiclass SVM $\Delta^C(c, Y_i) = 1[c \neq Y_i]$. As in [71], this problem is solvable using a structured SVM, where $\Psi(X, Y)$ concatenates features for each class:

$$\Psi^C(X, Y) = [\psi_1(X, Y) \dots \psi_C(X, Y)] \quad (7.6)$$

$$\psi_c(X, Y) = \begin{cases} \phi(X) & \text{if } Y = c \\ \mathbf{0} & \text{otherwise} \end{cases} \quad (7.7)$$

This is a cost-sensitive version of a Crammer-Singer multiclass SVM [14]. It is slightly different than 1-vs-all SVM training, since parameters for each class are trained jointly. Computation of Eq 7.1-7.2 involves computing $\langle \mathbf{w}_c, \phi(X_i) \rangle$ for each class and takes $O(dC)$, where C is the number of classes and d is the number of non-zero features. An interactive interface is not applicable since the structured output space is one-dimensional.

7.2 Alignment Models

The use of stronger alignment models has become an increasingly popular and important in computer vision for a few reasons

1. **Alignment parameters are a desired output of many computer vision applications** (*e.g.*, the location of the object in the image, the pose of the object, the relative positions of different parts of the object, *etc.*)
2. **Better representation of statistical properties of objects:** Statistical properties of an object and its appearance in most feature spaces is often most naturally expressed as a function of the location of the object in the image. Excluding alignment from the model corresponds to making stronger conditional independence assumptions (*e.g.*, Figure 7.6a vs. Figure 7.6c), and it appears to be difficult or impossible to invent a good feature space that is invariant to object deformation.
3. **Saturation of research in object recognition methods that don't model alignment:** Research in object recognition that doesn't model alignment (and instead rely on features like bag of words) has not led to much quantitative improvement in performance in the last 5 years, while still being far from achieving usable accuracy levels for most applications. By contrast, methods that incorporate stronger alignment models are improving more quickly and work well enough to be used in some applications [93, 88].

7.2.1 Sliding Window Object Detection

A sliding window object detector can be trained using a structured SVM [5], where $Y = \{x, y, s\}$ encodes a bounding box of fixed aspect ratio that is centered at (x, y) and has scale s . A feature space $\Psi^B(X, Y)$ is extracted with respect to this bounding box (*e.g.*, extract features like HOG or color histograms with respect to a coordinate system centered at (x, y) and axis length defined by s).

Let $\Delta^B(Y, Y_i)$ be an arbitrary loss function associated with predicting bounding box Y when the true bounding box is Y_i . Let $\mathbf{\Delta}^B$ encode all such values into an array, such that $\mathbf{\Delta}^B[Y] = \Delta^B(Y, Y_i)$. Similarly, let \mathbf{M} be an array of sliding window responses, such that $\mathbf{M}[Y] = \langle \mathbf{w}, \Psi(X, Y) \rangle$ (fast algorithms such as convolutional methods exist to densely compute \mathbf{M} for certain types of feature spaces). Note that the most violated label \bar{Y} can be found efficiently by adding the loss into the sliding window response

$$\bar{Y} = \arg \max_Y (\mathbf{M} + \mathbf{\Delta}^B)[Y], \quad (7.8)$$

enabling efficient computation of $\nabla f(\mathbf{w}; Z_i)$.

Loss Functions For Object Detection

Loss functions that we consider in this dissertation include the area of intersection over area of union criterion used in the VOC Pascal detection challenge:

$$\Delta_{voc}^B(Y, Y_i) = 1 - \frac{\text{AREA}(Y \cap Y_i)}{\text{AREA}(Y \cup Y_i)} \quad (7.9)$$

and a loss that compares performance to human users:

$$\Delta_{mturk}^B(Y, Y_i) = \sqrt{(Y - Y_i)^T \Sigma^{-1} (Y - Y_i)} \quad (7.10)$$

where Σ is a 3X3 covariance matrix obtained by polling bounding box labels from multiple mechanical turk users per image and comparing them to ground truth. We also consider thresholded versions of the above loss functions, such that the loss is always 0 or 1.

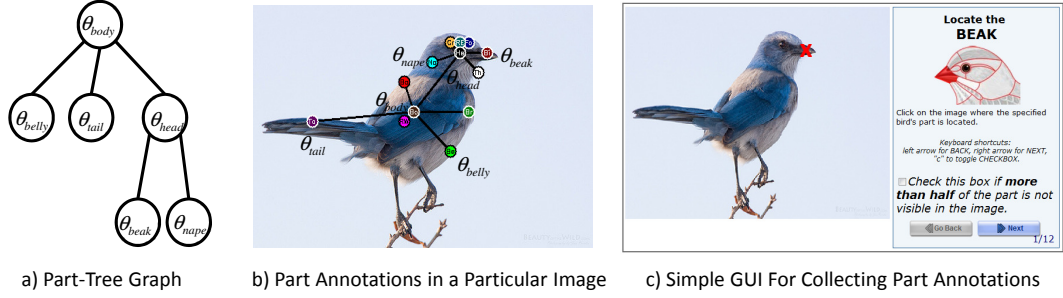


Figure 7.1: Part-Based Model: a) Spatial relationships between different parts is represented by a tree-structured graph (only a subset of this tree is shown for visualization purposes). b) Annotated part locations in a particular image. c) A step-by-by GUI for collecting part annotations where a user annotates only one part at a time via a simple mouse click.

7.2.2 Deformable Part Models

Many or most objects (*e.g.*, articulated objects like humans) deform with more degrees of freedom than is modeled by a simple bounding box. Part-based methods represent the location of an object as a set of part locations $Y = \{\theta_1, \dots, \theta_P\}$. Each part location can be represented as a bounding box $\theta_p = \{x_p, y_p, s_p\}$ (we consider other possible representations in the next section).

Specialized algorithms exist for exact inference on a certain class of deformable part models while handling a number of challenging factors (*e.g.*, sliding window detectors for each part, per-part occlusion reasoning, mixtures of different poses or views of the object). We assume that parts can have spatial dependencies that are representable by a tree-structured graph $T = (V, E)$ (see Figure 7.1, where a part q can be connected to its parent part p by a spring cost (an arbitrary quadratic or linear function of the scale-normalized offset between parent and child):

$$\lambda_{pq}(\theta_p, \theta_q) = \langle \mathbf{w}_{pq}, \omega_{pq}(\theta_p, \theta_q) \rangle \quad (7.11)$$

$$\omega_{pq}(\theta_p, \theta_q) = [\dot{x}_{pq}, \dot{x}_{pq}^2, \dot{y}_{pq}, \dot{y}_{pq}^2] \quad (7.12)$$

$$\dot{x}_{pq} = \frac{x_p + \mu_q^x - x_q}{s_p}, \quad \dot{y}_{pq} = \frac{y_p + \mu_q^y - y_q}{s_p} \quad (7.13)$$

where \mathbf{w}_{pq} is a learned vector of weights parameterizing the spring cost between parts p and q . The parameters μ_q^x are an optional ideal spatial offset between parent and child (the offset could effectively be learned up to a constant by changing the learned weight

on \hat{x}_{pq}). Let $\varphi_p(\theta_p; X)$ be a vector of appearance features extracted at part location θ_p , and let \mathbf{w}_p be a learned vector of weights on these appearance features, such that the appearance score for part p at each location b_p is:

$$\psi_p(\theta_p; X) = \langle \mathbf{w}_p, \varphi_p(\theta_p; X) \rangle \quad (7.14)$$

Structured prediction on a deformable part model solves the following problem:

$$Y^* = \arg \max_{\theta_1 \dots \theta_P} \left(\sum_{p \in V} \psi_p(\theta_p; X) + \sum_{(p,q) \in E} \lambda_{pq}(\theta_p, \theta_q) \right) \quad (7.15)$$

By simple examination of equations 7.11-7.16, we see that this is equivalent to the structured prediction formulation

$$Y^* = \arg \max_Y \langle \mathbf{w}, \Psi^P(X, Y) \rangle \quad (7.16)$$

if we define $\Psi(X, Y)$ as the concatenation of all appearance features $\varphi_p(\theta_p; X)$ for $p \in V$ and spatial features $\lambda_{pq}(\theta_p, \theta_q)$ for $(p, q) \in E$:

$$\Psi^P(X, Y) = [\varphi_1, \varphi_2, \dots, \varphi_P, \omega_{\text{par}(1),1}, \omega_{\text{par}(2),2}, \dots, \omega_{\text{par}(P-1),P-1}] \quad (7.17)$$

$$\mathbf{w} = [\mathbf{w}_1, \mathbf{w}_2, \dots, \mathbf{w}_P, \mathbf{w}_{\text{par}(1),1}, \mathbf{w}_{\text{par}(2),2}, \dots, \mathbf{w}_{\text{par}(P-1),P-1}] \quad (7.18)$$

if we assume part P is the root of the tree. For tree structured graphs, the optimal set of part locations Y^* can be solved in $O(PI)$ time using dynamic programming, where P is the number of parts and I is the number of pixel locations times the number of scales in the image, using the algorithm shown in Algorithm 8. This algorithm makes use of a modified distance transform function of Felzenszwalb et al. [28], which densely solves the following optimization problem

$$O_q[\theta_p] = \arg \max_{\theta_q} (M_q[\theta_q] + \lambda_{pq}(\theta_p, \theta_q)) \quad (7.19)$$

$$N_q[\theta_p] = \max_{\theta_q} (M_q[\theta_q] + \lambda_{pq}(\theta_p, \theta_q)) \quad (7.20)$$

in time linear in the number of possible locations of θ_p (whereas a naive algorithm would take quadratic time).

Algorithm 8 DPMINFERENCE($X, p, \mathbf{w}, \Delta, \mathbf{U}$)

- 1: $r \leftarrow \text{root}(T)$
- 2: OPTIMIZE SUBTREE($X, r, \mathbf{w}, \Delta, \mathbf{U}$), assume $\mathbf{M}_1 \dots \mathbf{M}_P, \mathbf{O}_1 \dots \mathbf{O}_P, \theta_1 \dots \theta_P$ are global
- 3: $\theta_r \leftarrow \arg \max \mathbf{M}_r[\theta_r]$
- 4: BACKTRACK(r, θ_r)
- 5: **return** $\theta_1 \dots \theta_P$

OPTIMIZE SUBTREE($X, p, \mathbf{w}, \Delta, \mathbf{U}$)

- 1: Compute detection responses \mathbf{M}_p (Eq 7.14 at all pixel locations and scales)
- 2: If $\Delta \neq \emptyset$ (finding the most violated constraint): $\mathbf{M}_p \leftarrow \mathbf{M}_p + \Delta_p^B$
- 3: If $\mathbf{U} \neq \emptyset$ (performing interactive labeling): $\mathbf{M}_p \leftarrow \mathbf{M}_p + \mathbf{U}_p$
- 4: **for all** $q \in \text{children}(p)$ **do**
- 5: $\mathbf{M}_q \leftarrow \text{OPTIMIZE SUBTREE}(X, q, \mathbf{w}, \Delta, \mathbf{U})$
- 6: $\mathbf{N}_q, \mathbf{O}_q \leftarrow \text{MODIFIED DISTANCE TRANSFORM}(\mathbf{M}_q, \mathbf{w}_{pq})$ (Eq 7.19)
- 7: $\mathbf{M}_p \leftarrow \mathbf{M}_p + \mathbf{N}_q$
- 8: **end for**
- 9: **return** \mathbf{M}_p

BACKTRACK(p)

- 1: **for all** $q \in \text{children}(p)$ **do**
 - 2: $\theta_q \leftarrow \mathbf{O}_q[\theta_p]$
 - 3: BACKTRACK(q, θ_q)
 - 4: **end for**
-

Loss Functions For Part Localization

We consider simple loss functions consisting of the sum loss over each part

$$\Delta^P(Y', Y_i) = \sum_{p=1}^P \Delta_p^B(\theta'_p, \theta_p) \quad (7.21)$$

where Δ_p^B is one of the loss functions defined in Section 7.2.1. As in Section 7.2.1, let us assume that this loss function has been densely precomputed into an array $\Delta_p^B[\theta_p]$. It

is easy to see that the most violated label \bar{Y} can be found by solving

$$\bar{Y} = \arg \max_{\theta_1 \dots \theta_P} \left(\sum_{p \in V} (\psi_p(\theta_p; X) + \Delta_p^B[\theta_p]) + \sum_{(p,q) \in E} \lambda_{pq}(\theta_p, \theta_q) \right) \quad (7.22)$$

$$= \arg \max_Y (\langle \mathbf{w}, \Psi(X, Y) \rangle + \Delta^P(Y, Y_i)) \quad (7.23)$$

which is solvable using Algorithm 8 if we simply add the part loss Δ_p^B to the part detection score M_p .

Interactive Part Prediction

Suppose the user has provided a partial annotation \tilde{Y} to a subset of part locations. Let $\tilde{\theta}_i$ denote the user's annotation to part i . We can see that the highest scoring user-consistent set of part locations $Y_{\tilde{Y}}^*$ (Eq 7.3) can be expressed as:

$$Y_{\tilde{Y}}^* = \arg \max_{\theta_1 \dots \theta_P} \left(\sum_{p \in V} (\psi_p(\theta_p; X) + \mathbf{U}_p[\theta_p]) + \sum_{(p,q) \in E} \lambda_{pq}(\theta_p, \theta_q) \right) \quad (7.24)$$

$$\mathbf{U}_p[\theta_p] = \begin{cases} -\infty & \text{if } \tilde{\theta}_p \text{ is labeled and } \theta_p \neq \tilde{\theta}_p \\ 0 & \text{otherwise} \end{cases} \quad (7.25)$$

where \mathbf{U} is a precomputed array that is $-\infty$ at all locations that don't agree with the user's label. This is solvable using Algorithm 8 if we simply add \mathbf{U}_p to the part detection score M_p .

7.2.3 Pose Mixture Models and Occlusion Reasoning

The model described in the previous section breaks if different parts are occluded or undergo significant changes in pose (consider for example looking at the side or back of a person's head, where a person's eyes, mouth and nose could become non-visible). This problem can be fixed by using a more complex representation of part location $\theta_p = \{x_p, y_p, s_p, r_p, v_p\}$, where v_p is a discrete pose or mixture component for part p . We have also added an optional orientation r_p of part p . The pose parameter v_p could correspond to values like *side view left*, *side view right*, *frontal view*, *not visible*, etc.. In this case we consider an augmented definition of our spatial cost between parent and

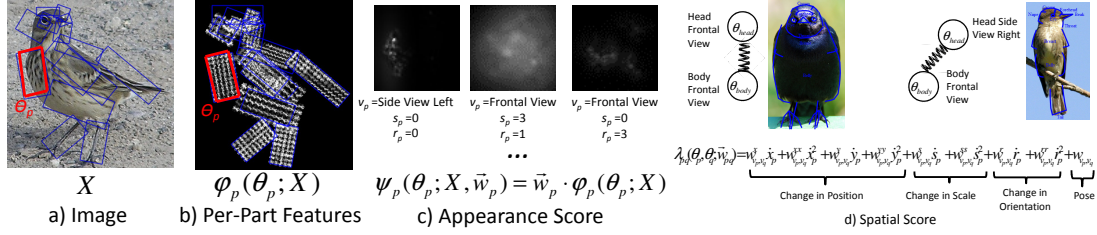


Figure 7.2: Components of a Part Based Model: a) Each part in an image X is represent by a location $\theta_p = \{x_p, y_p, s_p, r_p, v_p\}$ (x-y location, scale, rotation, pose). b) Appearance features (in this case HOG features) are extracted with respect to a part location and stored into a vector $\varphi_p(\theta_p; X)$. c) A sliding window part detector computes a detection score $\psi_p(\theta_p; X)$ for every possible value of θ_p . d) Spatial features $\lambda_{pq}(\theta_p, \theta_q)$ occur over every parent-child part, with a quadratic cost penalizing changes in location, scale, and orientation. Each possible pair of poses for child and parent parts has a different spring cost and different pose transition score

child parts, where the spatial score between parent and child parts p and q in poses v_p and v_q can be written as:

$$\lambda_{pq}^{v_p, v_q}(\theta_p, \theta_q) = \langle \mathbf{w}_{pq}^{v_p, v_q}, \omega_{pq}^{v_p, v_q}(\theta_p, \theta_q) \rangle \quad (7.26)$$

$$\omega_{pq}^{v_p, v_q} = [\dot{x}_{pq}^2, \dot{y}_{pq}^2, \dot{s}_{pq}^2, \dot{r}_{pq}^2, 1] \quad (7.27)$$

This spatial score (as for Eq 7.11) is a function of the x, y displacements $\dot{x}_{pq}, \dot{y}_{pq}$ between p and q as well as the change in scale and orientation $\dot{s}_{pq} = s_p - s_q$ and $\dot{r}_{pq} = r_p - r_q$. Additionally, it adds a transition score (the weight applied to the element 1, the last element of $\omega_{pq}^{v_p, v_q}$), which is a learned likelihood of observing the pose pair v_p and v_q . A separate vector of weights $\mathbf{w}_{pq}^{v_p, v_q}$ is learned for all possible pose pairs v_p and v_q , and the spatial score $\lambda_{pq}(\theta_p, \theta_q)$ is set to the applicable value of $\lambda_{pq}^{v_p, v_q}$:

$$\begin{aligned} \lambda_{pq}(\theta_p, \theta_q) &= \langle \mathbf{w}_{pq}, \omega_{pq}(\theta_p, \theta_q) \rangle \\ \omega_{pq}(\theta_p, \theta_q) &= [\pi_{1,1}, \pi_{2,1}, \dots, \pi_{Q_p,1}, \pi_{1,2}, \pi_{2,2}, \dots, \pi_{Q_p,2}, \dots, \pi_{1,Q_q}, \pi_{2,Q_q}, \dots, \pi_{Q_p,Q_q}] \\ \pi_{i,j}(\theta_p, \theta_q) &= \begin{cases} [1] & \text{if } v_p = i \text{ and } v_q = j \text{ and } v_q = \text{not visible} \\ \omega_{pq}^{v_p, v_q} & \text{else if } v_p = i \text{ and } v_q = j \\ \mathbf{0} & \text{otherwise} \end{cases} \quad (7.28) \end{aligned}$$

Here, a special pose $v_q = \text{not visible}$ is used to handle the case when q is occluded or self-occluded. In this case, the values x_q, y_q, s_q, r_q have no meaning, and the spatial score $\lambda_{pq}^{v_p, v_q}(\theta_p, \theta_q)$ is simply the learned likelihood weight that q is not visible given v_p .

Each visible pose j for a given part p is associated with a set of appearance features $\varphi_{p_j}(\theta_p; X)$ that parameterized by weights \mathbf{w}_{p_j} . For a non-visible pose j , $\varphi_{p_j}(\theta_p; X)$ is an empty (zero dimensional) vector. The appearance score of part p with Q poses is the appearance score of the appropriate pose v_p :

$$\psi_p(\theta_p; X) = \langle \mathbf{w}_p, \varphi_p(\theta_p; X) \rangle \quad (7.29)$$

$$\varphi_p(\theta_p; X) = [\varphi_{p_1}(\theta_p; X) \dots \varphi_{p_Q}(\theta_p; X)] \quad (7.30)$$

$$\mathbf{w}_p = [\mathbf{w}_{p_1} \dots \mathbf{w}_{p_Q}] \quad (7.31)$$

$$\varphi_{p_j}(\theta_p; X) = \begin{cases} \emptyset & \text{if } j \text{ is not visible} \\ \varphi(\theta_p; X) & \text{else if } v_p = j \\ \mathbf{0} & \text{otherwise} \end{cases} \quad (7.32)$$

where $\varphi(\theta_p; X)$ is a vector of features extracted at location θ_p . The structured feature space concatenates appearance features φ_p for all parts p and spatial features ω_{pq} for all parent-child parts p and q , as for Eq 7.17.

Structured prediction and learning works using the same deformable part model inference algorithm that was defined in the previous section (Algorithm 8). The only change is the solver for Eq 7.20 maximizes over a larger number of parameters:

$$N_q[\theta_p] = \max_{x_p, y_p, s_p, r_p, v_p} (M_q[\theta_q] + \lambda_{pq}(\theta_p, \theta_q)) \quad (7.33)$$

Dense computation of $N_q[\theta_p]$ for all values of θ_p takes time quadratic in the number of pose mixture components and linear in the number of pixel locations, scales, and orientations using distance transforms [28] (Assume $M_q[x_q, y_q, s_q, r_q, v_q]$ has been computed in a previous step. Loop through all allowable pose pairs v_p and v_q , then sequentially run a distance transform operation for each dimension x, y, s, r . For orientation, the boundaries of $N_q[\dots r_q \dots]$ must be padded by π radians to handle the looping nature of orientation).

7.2.4 Annotation Interfaces For Part-Based Models

Incorporating pose mixture models is important for obtaining good practical performance for part-based models; however, it introduces new challenges for collecting

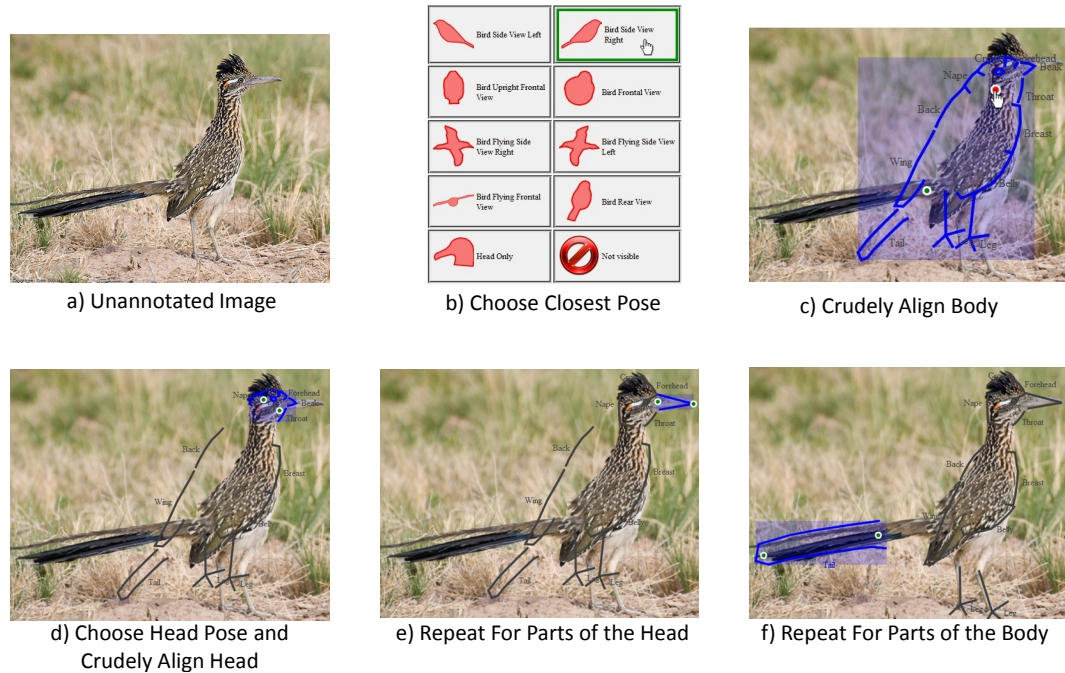


Figure 7.3: Hyper-Supervised Part Annotation Given a Predefined Bird Model: In contrast to the GUI in Figure 7.1, the user can label the scale, orientation, and pose of each part, in addition to the pixel location. Annotation occurs as a course-to-fine step-by-step procedure. The model can be used to align birds that vary significantly in shape and pose.

annotations since pose can be difficult or ambiguous to label. We consider a few possible solutions in this section.

A Tool For Collecting Detailed Part Annotations

A tool for collecting part and pose annotations in the style of Section 7.2.3 is shown in Fig 7.3, and a corresponding expert tool for defining part models (*e.g.*, decomposing an object into a set of parts and poses) is shown in Fig 7.4.

A Simpler Annotation Interface With Pose Clustering Methods

Obtaining pose annotations as described in the previous section can be expensive. In this section, we instead assume that we have collected only an x, y location and visible/non-visible tag $\tilde{v} \in \{0, 1\}$ for each part using one of the interfaces shown in Figure 7.1b-c). The scale s and pose v of each part is left unlabeled and is inferred

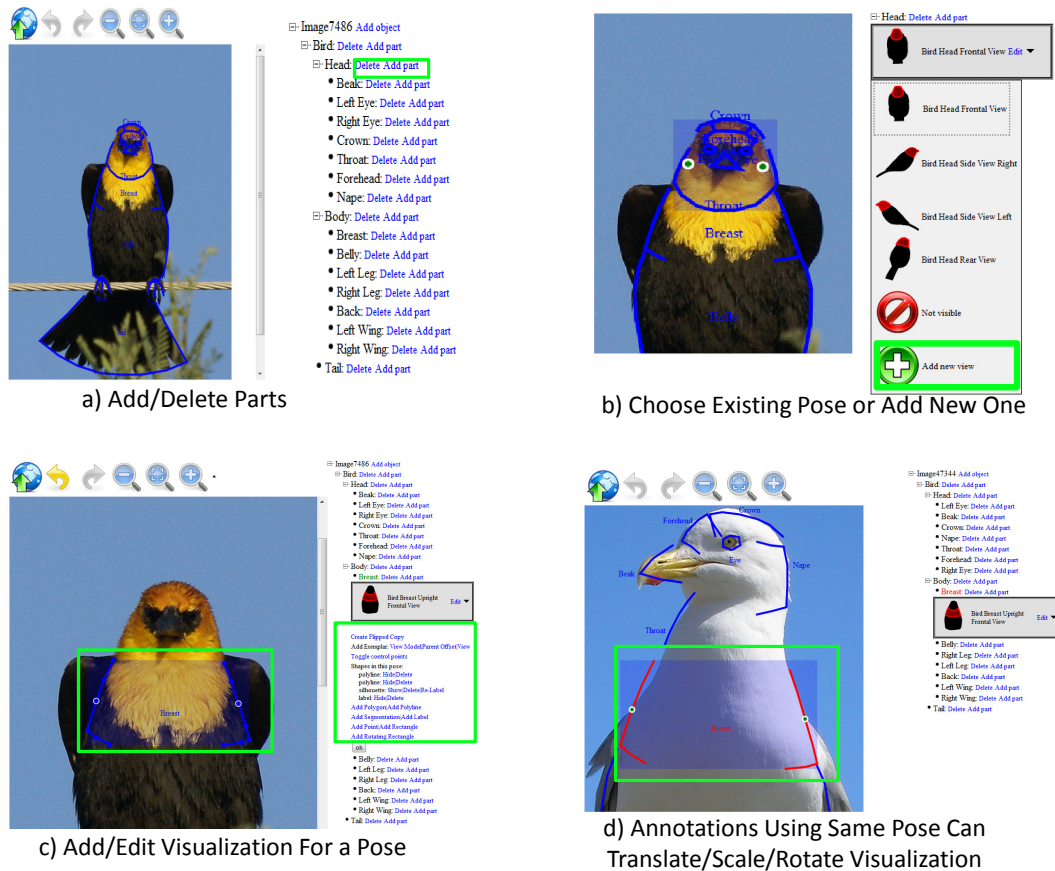


Figure 7.4: GUI For Building Custom Part Models: A web-based GUI for defining a custom part model for new categories. Models created with this tool can be used to annotate images as in Figure 7.3 a) The user can add new parts using the tree control on the right. b) When annotating a new image, the user can select a pose of a part from a set of previously defined poses. If no previously defined pose matches the current image, the user can create a new pose. c) When creating a new pose (in this case a frontal view of the breast), the user can draw a custom vector graphics visualization of that pose. d) When annotating a different image, a user can select the same frontal view pose of the breast, adjusting the location of 2 control points to rotate/scale/translate the pose visualization.

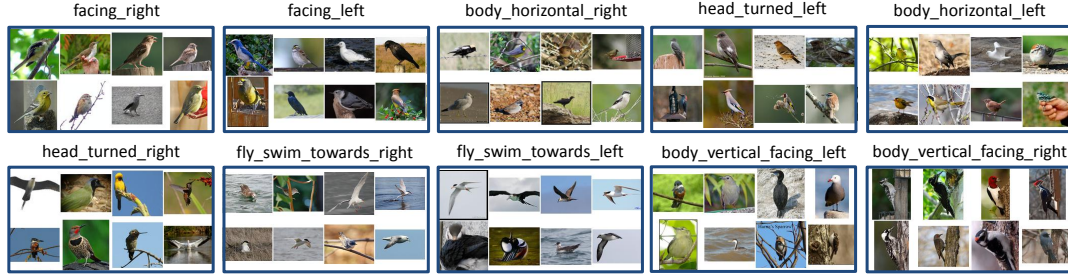


Figure 7.5: Pose Clusters For the Part *Bird Body*: Images in our dataset are clustered by k -means on the spatial offsets of part locations from parent part locations. Semantic labels of clusters were manually assigned by visual inspection. Left/right orientation is in reference to the image.

using unsupervised methods.

Setting Orientation and Scale: In this case, we ignore orientation, setting $r = 0$, such that handling rotation an object or part will have to be absorbed by the pose mixture model. Additionally, we assume a single scale associated with the object, such that all parts of the same object will be assigned the same scale. We choose s as being proportional to the width of the bounding box of the object.

Clustering Pose Using Relative Part Offsets: For each non-leaf part p with A_q child parts q_1, q_2, \dots, q_{A_q} , we consider a feature space that appends the location offsets and visibility flags of each child part:

$$f_p = [\dot{x}_{pq_1}, \dot{y}_{pq_1}, \gamma \tilde{v}_{q_1}, \dot{x}_{pq_2}, \dot{y}_{pq_2}, \gamma \tilde{v}_{q_2}, \dots, \dot{x}_{pq_{A_q}}, \dot{y}_{pq_{A_q}}, \gamma \tilde{v}_{q_{A_q}}] \quad (7.34)$$

where \dot{x}_{pq_1} and \dot{y}_{pq_1} are defined as for Eq 7.11, and γ is a parameter that trades off the relative contribution of part visibility and spatial offset to the pose clustering cost. Given a dataset of labeled part locations, a set of pose mixture components is learned by k -means clustering on this feature space. Part p as well as all visible child parts $q_1 \dots q_{A_q}$ are assigned to the same cluster center.

Clustering Pose Using Segmentations: Here, we assume that training images have also been annotated with a figure-ground segmentation defining which pixels are inside the object. We extract a patch from this segmentation centered about each visible part location (x_p, y_p) . Patches across each training image are clustered independently for each

leaf part p . Non-leaf parts are instead clustered using relative part offsets as described in the previous section.

7.3 Model Sharing Methods

In this section we consider a few possible models for transfer learning and multi-task learning using shared part and attribute models.

7.3.1 Joint Learning of Attribute Classifiers

Attributes are semantically interpretable mid-level features that are typically shared among multiple tasks or multiclass detectors. For example, attributes such as *striped belly* or *blue tail* could be shared among multiple bird species classifiers. There are a few reasons why attribute-based methods have recently become popular in computer vision:

1. **Predicting the attributes themselves may be a desired output of computer vision:** Computer vision research over the past decade has focused primarily on a handful of object classes like faces or airplanes (nouns in basic-level categories), while neglecting detecting attributes (typically adjectives), actions (typically verbs), and subordinate categories. There is some criticism that this particular choice of emphasis has biased computer vision research.
2. **Transfer learning and multitask learning:** Attributes serve as an intermediate layer that is shared by multiple classes or tasks. Model sharing can reduce computation time by sharing processing between different tasks. It can also reduce the number of training examples per class that are needed to obtain a given test error by reducing the total number of model parameters (since weights on different image-level features are shared among classes).
3. **New ways to communicate with humans:** Attributes are another avenue for obtaining supervised annotation from humans, for communicating for interactive applications, and for machines to provide humans with additional feedback about the inner workings or failures of computer vision algorithms.

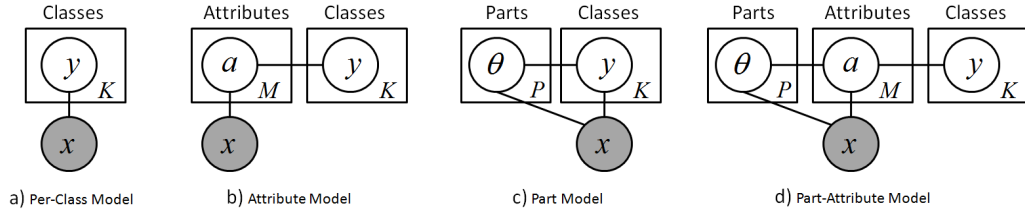


Figure 7.6: Graphical Models For Alignment and Model Sharing: a) Simple multiclass classification methods train independent classifiers for each of C classes, and featurize an entire image without modeling object location. b) Attribute-based methods employ a layer of M attribute detectors, which are functions of low-level image features and shared among classes. c) Localized models employ class detectors that can be complex functions of the object’s location in the image. d) Part-attribute methods can employ both part and attribute detectors that are shared among classes.

The most widely used attribute-based model is the direct class attribute model from Lampert et al. [40], visualized in Fig 7.6b. Here, a set of weights \mathbf{w}_a on low-level image features $\phi(X)$ is learned for each attribute $a \in 1 \dots M$ independently

$$s_a(X) = \langle \mathbf{w}_a, \phi(X) \rangle \quad (7.35)$$

The set of attributes is shared among all classes via an expert-defined vector of attribute memberships $\mathbf{a}^c = a_1^c, \dots, a_M^c$, where $-1 \leq a_1^c \leq 1$ for each class $c \in 1 \dots C$, such that the classification score of class c can be expressed as

$$h_c(X) = \sum_{i=1}^A a_i^c s_a(X) \quad (7.36)$$

In this section, we describe a method that uses this same direct class attribute prediction model, but learns the attribute weights for all attributes jointly while optimizing multiclass classification accuracy. The problem can be solved using the same loss function as the cost-sensitive multiclass SVM

$$\forall_{i,c}, h_c(X_i) + \Delta^C(c, Y_i) \leq h_{Y_i}(X_i) + \epsilon_i \quad (7.37)$$

The only difference is that the goal is to learn attribute weight vectors $\mathbf{w} = [\mathbf{w}_1, \dots, \mathbf{w}_A]$. Examining equations 7.35-7.36, we see that

$$\max_{c \in 1 \dots C} h_c(X) = \max_{Y \in 1 \dots C} \langle \mathbf{w}, \Psi^A(X, Y) \rangle \quad (7.38)$$

if we define a structured feature space $\Psi^A(X, Y)$ that simply replicates $\phi(X)$ for each attribute i and weights it by its class-attribute membership a_i^c :

$$\Psi^A(X, Y) = [a_1^Y \phi(X), \dots, a_M^Y \phi(X)] \quad (7.39)$$

7.3.2 Part Sharing Models

In part-sharing, a common set of part detectors is shared among multiple classes (for example, if our goal is to detect multiple bird species, we can share part detectors among bird species). This can be used to add localization information to extend either the multiclass classification model from Section 7.1 (depicted by the graphical model in Fig 7.6c).

In the former case, we can now think of the representation of an object label $Y = \{c, \theta_1, \dots, \theta_P\}$ as a class c and a set of P part locations, where $\theta = \theta_1 \dots \theta_P$ is as defined in Section 7.2.3. We assume that each class c has a set of a vector of appearance weights \mathbf{w}_c on part localized features $\Psi^P(X, \theta)$ (see Eq 7.17). All classes also share the same set of part detectors via a vector of appearance weights \mathbf{w}^P on $\Psi^P(X, \theta)$, such that the detection score for class c is

$$g_c(X, Y; \mathbf{w}) = \langle \mathbf{w}_c, \Psi^P(X, \theta) \rangle + \langle \mathbf{w}^P, \Psi^P(X, \theta) \rangle \quad (7.40)$$

The predicted class and part locations with the highest score is:

$$Y^* = \arg \max_{\theta, c} g_c(X, Y; \mathbf{w}) \quad (7.41)$$

$$= \arg \max_Y \langle \mathbf{w}, \Psi(X, Y) \rangle \quad (7.42)$$

$$\mathbf{w} = [\mathbf{w}^P, \mathbf{w}_1, \mathbf{w}_2, \dots, \mathbf{w}_C] \quad (7.43)$$

$$\Psi(X, Y) = [\Psi^P(X, Y), \Psi_1^P(X, Y), \Psi_2^P(X, Y), \dots, \Psi_C^P(X, Y)] \quad (7.44)$$

$$\Psi_i^P(X, Y) = \begin{cases} \Psi^P(X, Y) & \text{if } i = c \\ \mathbf{0} & \text{otherwise} \end{cases} \quad (7.45)$$

This can be computed simply by running a separate prediction problem (Algorithm 8) for each class:

$$Y^* = \arg \max_c \text{DPMINFERENCE}(X, \mathbf{w}_c + \mathbf{w}^P) \quad (7.46)$$

Loss Functions For Part Sharing Models

We consider loss functions that are some weighted combination of multiclass classification loss $\Delta^C(c, Y_i)$ (see Section 7.1) and part localization loss $\Delta^P(\theta, Y_i)$:

$$\Delta(Y, Y_i) = \Delta^C(c, Y_i) + \gamma_P \Delta^P(\theta, Y_i) \quad (7.47)$$

where γ_P is a constant that is used to tradeoff how much we care about part localization accuracy as compared to classification accuracy. The most violated label can also be solved using Algorithm 8:

$$\bar{Y} = \arg \max_c \text{DPMINFERENCE}(X, \mathbf{w}_c + \mathbf{w}^P, \Delta^C(c, Y_i) + \Delta^P) \quad (7.48)$$

Interactive Labeling For Part Sharing Models

In interactive labeling, a user can specify a partial labeling \tilde{Y} , which can contain a partial labeling $\tilde{\theta}$ to a subset of part locations, to the object class \tilde{c} , or to some subset of the two. This is once again solvable using Algorithm 8:

$$Y_{\tilde{Y}}^* = \arg \max_c \text{DPMINFERENCE}(X, \mathbf{w}_c + \mathbf{w}^P, \emptyset, \mathbf{U}^c) \quad (7.49)$$

$$\mathbf{U}_p^c[\theta_p] = \begin{cases} -\infty & \text{if } \tilde{\theta}_p \text{ is labeled and } \theta_p \neq \tilde{\theta}_p \text{ or } \tilde{c} \text{ is labeled and } c \neq \tilde{c} \\ 0 & \text{otherwise} \end{cases} \quad (7.50)$$

7.3.3 Part-Attribute Sharing Models

We can also share both part and attribute detectors among different classes, extending the class-attribute model from Section 7.3.1 to include part localization information (depicted by the graphical model in Fig 7.6d). This makes intuitive sense, since many or most attributes tend to be associated with a part (*e.g.* striped belly, long tail, *etc.*) or an object (*e.g.* red car, gull-shaped bird, *etc.*) as opposed to an image. Thus whereas most research in attribute-based methods tend to predict image-level attributes that extract features from the entire image (as in Section 7.3.1), a model that is capable of associating attributes with their associated part locations is capable of both providing a more realistic output and incorporating more relevant appearance features.

In this case, we represent an object label $Y = \{c, \mathbf{a}, \theta\}$ as a class c , a vector of M attributes $\mathbf{a} = a_1 \dots a_M$, and a set of P part locations $\theta = \theta_1 \dots \theta_P$ (defined in Section 7.2.3). A vocabulary of M attribute detectors and P part detectors is shared among C classes. For simplicity, we assume each attribute a is associated by a part $p(a)$, such that the detection score for an attribute at location θ is

$$s_a(X, \theta) = \langle \mathbf{w}_a, \Psi^P(X, \theta) \rangle \quad (7.51)$$

where $\Psi^P(X, \theta)$ is a vector of appearance features extracted at locations θ (see Section 7.2.3). The set of attributes is shared among all classes via an expert-defined vector of attribute memberships $\mathbf{a}^c = a_1^c, \dots, a_M^c$, where $-1 \leq a_i^c \leq 1$ for each class $c \in 1 \dots C$, such that the detection score of class c can be expressed as

$$h_c(X, \theta) = \langle \mathbf{w}^P, \Psi^P(X, \theta) \rangle + \sum_{i=1}^A a_i^c s_a(X, \theta) \quad (7.52)$$

where $\langle \mathbf{w}^P, \Psi^P(X, \theta) \rangle$ is the part detection score at θ (see Section 7.2.3). The predicted label Y^* is the class c and set of part locations θ that maximizes this score (note that since class-attribute memberships are assumed to be deterministic, the corresponding set of predicted attributes will be \mathbf{a}^c):

$$\begin{aligned} Y^* &= \arg \max_{c, \theta} h_c(X, \theta) \\ &= \arg \max_Y \langle \mathbf{w}, \Psi(X, Y) \rangle \\ \mathbf{w} &= [\mathbf{w}^P, \mathbf{w}_{a_1}, \mathbf{w}_{a_2}, \dots, \mathbf{w}_{a_M}] \\ \Psi(X, Y) &= [\Psi^P(X, Y), a_1^c \varphi_{p(a_1)}(\theta_{p(a_1)}; X), \dots, a_M^c \varphi_{p(a_M)}(\theta_{p(a_M)}; X)] \end{aligned} \quad (7.53)$$

This is solvable by running Algorithm 8 separately for each class c , where the part detection weights \mathbf{w}_p for part p and class c

$$\mathbf{w}_p = \mathbf{w}_p^P + \sum_{a_i, p(a_i)=p} a_i^c \mathbf{w}_{a_i} \quad (7.54)$$

Incorporation of loss functions and interactive labeling works the same way as described in Section 7.3.2.

7.3.4 Approximate Inference Algorithms and Extensions

The prediction algorithms described in Section 7.3.2-7.3.3 take $O(CPI)$ time, where C is the number of classes, P is the number of parts, and I is the number of pixel locations per image. A faster, approximate inference technique that takes $O(C + PI)$ time is described in Section 3.1. Section 3.1 also contains various other extensions including more sophisticated models of imperfect user interaction and active prediction algorithms.

7.4 Tracking

In this section, we describe a method for simultaneous detection and tracking of a particular type of object such as a face or a human body. Let X be a video sequence consisting of B image frames, and $Y = \{y_1, y_2, \dots, y_B\}$ denote the position of the object in each frame, where $y_t = \{x_t, y_t, s_t\}$ is the bounding box of the object (as defined in Section 7.2.1 in frame t).

Let $\Psi_t^B(X, y_t)$ be a vector of appearance features extracted from the bounding box y_t in frame t and let \mathbf{w}_a be a vector of appearance weights, such that the detection score for an object in frame t is

$$\psi(y_t; X_t) = \langle \mathbf{w}_t, \Psi_t^B(X, y_t) \rangle \quad (7.55)$$

Let $\lambda(y_{t-1}, y_t)$ be a temporal score measuring the relative likelihood of an object moving from location y_{t-1} at frame $t - 1$ to location y_t at frame t :

$$\lambda(y_{t-1}, y_t) = \langle \mathbf{w}_b, \omega(y_{t-1}, y_t) \rangle \quad (7.56)$$

$$\omega(y_{t-1}, y_t) = [\dot{x}, \dot{x}^2, \dot{y}, \dot{y}^2] \quad (7.57)$$

$$\dot{x} = \frac{x_{t-1} + \mu^x - x_t}{s_{t-1}}, \quad \dot{y} = \frac{y_{t-1} + \mu^y - y_t}{s_{t-1}} \quad (7.58)$$

Note that Eq 7.56 has identical form to the spring cost we used for our deformable part model (Eq 7.11). The highest scoring set of object locations across the entire video sequence is:

$$Y^* = \arg \max_{y_1 \dots y_B} \left(\sum_{t=1 \dots B} \psi(y_t; X_t) + \sum_{t=2 \dots B} \lambda(y_{t-1}, y_t) \right) \quad (7.59)$$

This can be solved using the same algorithm that we used for deformable part model inference (Algorithm 8), if we define a tree $T = (V, E)$, with one node for each time frame $1 \dots B$ and an edge between each consecutive time frame.

Optimizing Loss Functions For Tracking

We consider loss functions consisting of the sum loss over each time frame

$$\Delta^T(Y', Y_i) = \sum_{t=1}^B \Delta(y'_t, y_{ti}) \quad (7.60)$$

where $\Delta(y'_t, y_{ti})$ is the loss associated with placing an object at location y'_t when the true location is y_{ti} (one of the loss functions defined in Section 7.2.1). As in Section 7.2.1, let us assume that this loss function has been densely precomputed into an array $\Delta_t[y_t]$. The most violated label \bar{Y} can be found by solving

$$\bar{Y} = \arg \max_{y_1 \dots y_B} \left(\sum_{t=1 \dots B} (\psi(y_t; X_t) + \Delta_t[y_t]) + \sum_{t=2 \dots B} \lambda(y_{t-1}, y_t) \right) \quad (7.61)$$

$$= \arg \max_Y (\langle \mathbf{w}, \Psi(X, Y) \rangle + \Delta(Y, Y_i)) \quad (7.62)$$

which is solvable using Algorithm 8 if we simply add the loss Δ_t to the frame detection score \mathbf{M}_t .

Interactive Prediction For Object Tracking:

Suppose the user has provided a partial annotation \tilde{Y} to the location of the object in a subset of the time frames. Let $\tilde{\theta}_t$ denote the user's annotation to frame t . We can see that the highest scoring user-consistent tracking $Y_{\tilde{Y}}^*$ (Eq 7.3) can be expressed as:

$$\bar{Y} = \arg \max_{y_1 \dots y_B} \left(\sum_{t=1 \dots B} (\psi(y_t; X_t) + \mathbf{U}_t[y_t]) + \sum_{t=2 \dots B} \lambda(y_{t-1}, y_t) \right) \quad (7.63)$$

$$\mathbf{U}_t[y_t] = \begin{cases} -\infty & \text{if } \tilde{y}_p \text{ is labeled and } y_t \neq \tilde{y}_t \\ 0 & \text{otherwise} \end{cases} \quad (7.64)$$

where \mathbf{U}_t is a precomputed array that is $-\infty$ at all locations that don't agree with the user's label. This is solvable using Algorithm 8 if we simply add \mathbf{U}_t to the frame detection score \mathbf{M}_t .

Articulated Tracking

We could also consider a more sophisticated tracker, where the goal is to track the position of each of P parts of an object over B time frames. Here, we assume a more complex representation of the the position of the object in each time frame $y_t = \{\theta_1^t \dots \theta_P^t\}$, where $\theta_p^t = \{x_p^t, y_p^t, s_p^t, r_p^t, v_p^t\}$ encodes the x,y location, scale, orientation, and pose of part p in the t -th time frame (as defined in Section 7.2.3). The highest scoring set of object part locations across the entire video sequence is obtained by summing the deformable part model score (Eq 7.15) for each time frame:

$$Y^* = \arg \max_{y_1 \dots y_B} \left(\sum_{t=1}^B \left(\sum_{p \in V} \psi_p(\theta_p^t; X_t) + \sum_{(p,q) \in E} \lambda_{pq}(\theta_p^t, \theta_q^t) \right) + \sum_{t=2}^B \lambda(\theta_r^{t-1}, \theta_r^t) \right) \quad (7.65)$$

Here, the temporal score is assumed to be a function of the position of only the root part r across time frames. Solving Eq 7.65 can be done using Algorithm 8, if we define a tree $T' = (V, E)$ that replicates our part tree (see Section 7.2.2) B times, adding an edge between the root part over consecutive time frames: $\theta_r^{t-1} \theta_r^t$. Optimization time is linear (but multiplicative) in the number of parts, time frames, and pixel locations. Incorporation of interactive prediction and optimization of different loss functions is straightforward.

7.5 Segmentation

Let X be an image and $Y = s_1 \dots s_I$ be a figure-ground segmentation, such that $s_i \in \{0, 1\}$ indicates whether or not the i -th pixel is inside the foreground or not. Let $\varphi_i(X)$ be a vector of appearance features extracted from a patch that is centered around the i -th pixel, and let \mathbf{w}_u be a learned vector of weights on these appearance features, such that

$$\Omega_i(s_i; X) = \langle \mathbf{w}_u, s_i \varphi_i(X) \rangle \quad (7.66)$$

is a unary score representing the relative likelihood that the i -th pixel is labeled as s_i . Let i and j be neighboring pixels, with $\Gamma_{ij}(s_i, s_j; X)$ be a pairwise score that penalizes neighboring pixels i and j from having differing labels, *e.g.*, a contrast-sensitive Potts

model:

$$\Gamma_{ij}(s_i, s_j; X) = w_b \rho_{ij}(s_i, s_j; X) \quad (7.67)$$

$$\rho_{ij}(s_i, s_j; X) = 1[s_i \neq s_j] \exp\{-(X_i - X_j)^2\} \quad (7.68)$$

where w_b is a learned weight on the relative importance of the pairwise score.

The total segmentation score sums together unary and pairwise potentials:

$$Y^* = \arg \max_{s_1 \dots s_I} \sum_{i=1}^I \Omega_i(s_i; X) + \sum_{i,j \in \text{neighbors}(i)} \Gamma_{ij}(s_i, s_j; X) \quad (7.69)$$

$$= \arg \max_Y \langle \mathbf{w}, \Psi^S(X, Y) \rangle \quad (7.70)$$

$$\mathbf{w} = [w_u, w_b] \quad (7.71)$$

$$\Psi^S(X, Y) = \left[\left(\sum_{i=1}^I s_i \varphi_i(X) \right), \left(\sum_{i,j \in \text{neighbors}(i)} \rho_{ij}(s_i, s_j; X) \right) \right] \quad (7.72)$$

The optimal segmentation Y^* is efficiently solvable using a max-flow min-cut algorithm [7].

Loss Functions For Segmentation

Let $\Delta^S(Y', Y)$ be the number of incorrect pixels in a predicted segmentation Y' :

$$\Delta^S(Y', Y) = \sum_{t=1}^I 1[s'_t \neq s_t] \quad (7.73)$$

The most violated label \bar{Y} can be found by solving

$$\bar{Y} = \arg \max_{s'_1 \dots s'_I} \sum_{i=1}^I (\Omega_i(s'_i; X) + 1[s'_i \neq s_i]) + \sum_{i,j \in \text{neighbors}(i)} \Gamma_{ij}(s'_i, s'_j; X) \quad (7.74)$$

which is solvable using the same min-cut max-flow algorithm.

Interactive Prediction For Segmentation:

Suppose the user has provided a partial annotation \tilde{Y} , *e.g.* a series of brush strokes defining subsets of pixels that are part of the foreground and background. Let \tilde{s}_i denote

the user's annotation to the i -th pixel. We can see that the highest scoring user-consistent segmentation $Y_{\bar{Y}}^*$ (Eq 7.3) can be expressed as:

$$\bar{Y} = \arg \max_{s_1 \dots s_I} \sum_{i=1}^I (\Omega_i(s_i; X) + U_i(s_i)) + \sum_{i,j \in \text{neighbors}(i)} \Gamma_{ij}(s_i, s_j; X) \quad (7.75)$$

$$U_i(s_i) = \begin{cases} -\infty & \text{if } \tilde{s}_i \text{ is labeled and } s_i \neq \tilde{s}_i \\ 0 & \text{otherwise} \end{cases} \quad (7.76)$$

which is solvable using the same min-cut max-flow algorithm.

7.6 Behavior and Action Recognition

In this section, we introduce an algorithm for representing, segmenting and classifying actions or behaviors (*e.g.*, specify the exact time periods in a video sequence when the mouse is grooming, running, sleeping, *etc.*) using more sophisticated temporally localized models. Let X be a video sequence, and Y be a segmentation of X into actions. In our experiments, we assume an object tracker (*e.g.*, the tracker described in Section 7.4) has been run on the video sequence X , such that we have an estimated location and orientation of an object and its parts in each time frame.

Our behavior detector analyzes the motion of these part positions over time and uses them to predict behaviors. Let $x_t = \{\theta_1^t \dots \theta_P^t\}$ denote the tracked location of each part in time frame t (see Section 7.4), which we refer to in this section as *per frame features*. Our goal is to segment the video sequence into a sequence of predicted actions $Y = \{y_1, y_2, \dots, y_m\}$, where $y_j = \{(b_j, e_j, c_j)\}$ is the j th interval in the segmentation of x , b_j and e_j mark the beginning and end of the interval and c_j is the class label of the activity that the boundary encloses. We refer to these labeled intervals as *bouts* of actions, and we assume a full segmentation of the video sequence, such that $b_1 = 1$, $b_S = T$, and $b_t = e_{t-1}$.

In order to more appropriately represent statistical patterns of temporal motion during an action, the algorithm relies on *bout features*, defined as $\psi(X, b, e)$. These can be arbitrary functions over the set of per-frame features $x_1 \dots x_t$, localized with respect to the beginning and end times of the action b and e . These bout features are multiplied by

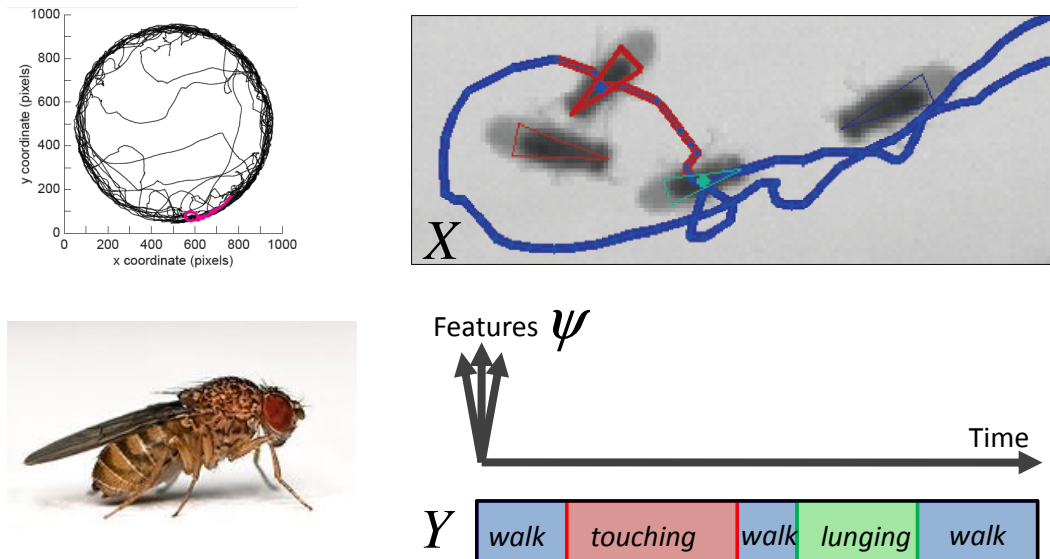


Figure 7.7: Overview of Behavior Detection in Flies: In this example, 50 flies are tracked in a circular arena. Features are computed as statistics or transformations of tracked trajectories in candidate time-localized behavior bouts. The system predicts a segmentation of a tracked trajectory into the set of predicted behaviors with the highest score.

a vector of learned model weights, computing a score measuring the likelihood that an action occurs at a particular time in the video sequence. These are combined with transition scores that encode the likelihood of two consecutive actions. Collectively, these define a *score function* measuring the likelihood of a segmentation of a video sequence into actions. Inference solves for the segmentation of a video that maximizes this score and is efficiently computable using dynamic programming. Training minimizes a convex upperbound on a customizable *loss function* that measures how much a predicted segmentation disagrees with the ground truth segmentation.

7.6.1 Behavior Bout features

A bout feature $\psi_k(X, b, e)$ is a function of the raw input X and an interval $[b, e]$. It is an abstracted version of the feature space, where statistical patterns existing in X may be more easily discovered or represented using knowledge of the beginning and end of each bout. We define a variety of different types of basic feature expansion operations that are used to synthesize bout-level features from frame-level features:

Bout Statistic Features: For each per-frame feature, we take the minimum, maximum, sum, mean and standard deviation of the feature over the frames spanned by $[b, e]$.

Histogram Features: For each per-frame feature, we take the histogram of the feature over the frames spanned by $[b, e]$.

Temporal Region Features: We split the interval $[b, e]$ into N smaller intervals. For each of the smaller intervals $[b_n, e_n]$ we extract bout statistic features from the per-frame features over the interval. This can be useful for capturing actions of complex structure, where the per-frame feature statistics during part of the action differ from the per-frame feature statistics during another part of the action. For example an action of drinking water may consist of a person reaching for a cup, grabbing the cup, lifting it towards its mouth, and so on, where each action has very different statistics for the velocity of the person’s hand.

7.6.2 Scoring and Predicting Behaviors

We define a function $f(X, Y)$ which measures how well Y segments X into actions. The score function is represented in terms of scores for individual bouts and transition costs, summing over each bout in the segmentation:

$$f(X, Y) = \sum_{(b_j, e_j, c_j) \in Y} \langle \mathbf{w}_{c_j}, \psi(x, b_j, e_j) \rangle - \lambda_{c_{j-1}}(c_j) - \gamma_{c_j}(b_j, e_j) \quad (7.77)$$

where \mathbf{w}_{c_j} are the weights used to calculate the score for a bout of class c_j , $\lambda_{c_{j-1}}(c_j)$ is the cost of moving from action c_{j-1} to c_j , and $\gamma_{c_j}(b_j, e_j)$ is the cost of spending $e_j - b_j$ frames in action c_j , which is 0 if the duration is within that action’s standard range (observed in training) and grows exponentially with its distance from the range. The learning algorithm will jointly learn all of the weights vectors, the transition cost matrix, and the duration cost vector.

Given weights \mathbf{w} and the costs λ and γ (see Eq. 7.77), the optimal solution to the score function $\max_Y f(X, Y)$ can be found using dynamic programming in time $O(T^2(C^2 + D))$, where T is the number of frames in X , C is the number of behavior classes and D is the dimensionality of the bout-level feature space $\psi_{seg}(X, Y)$. The algorithm is shown in Algorithm 9.

Algorithm 9 BEHAVIORSEGMENTATIONINFERENCE(X, \mathbf{w})

- 1: Initialize: $S[0][1..C] \leftarrow 0, M[0][1..C] \leftarrow \emptyset$
 - 2: **for** $e = 1$ to T **do**
 - 3: **for** $c \in C$ **do**
 - 4: Compute the optimal score $S[e][c]$ and its corresponding solution $M[e][c]$ for beginning a bout of class c at time e :

$$S[e][c] \leftarrow \max_{0 \leq b < e, c' \in C} \left(S[b][c'] + \langle \mathbf{w}_{c'}, \vec{\psi}_{bout}(x, b, e, c') \rangle - \lambda_{c'}(c) - \gamma_{c'}(b, e) \right)$$
 - 5: **end for**
 - 6: **end for**
 - 7: Backtrack to extract optimal solution:

$$(b_m, c_m) \leftarrow \max_c M[T][c]$$
for $i = m, m - 1, \dots, 2$, $(b_{i-1}, c_{i-1}) \leftarrow M[b_i][c_i]$ **end for**
 - 8: **return** $Y = \{(b_1, e_1, c_1), (b_2, e_2, c_2), \dots, (b_m, e_m, c_m)\}$
-

At each time frame e , the dynamic program searches for the bout ending at frame e , which has the highest score when combined with the score of the sequence ending in that bout's start frame. It searches through all possible frames $b < e$ as possible start frames, and all possible behaviors c' .

An $O(T \log T)$ approximate inference algorithm is also possible. Here, we consider searching over just a subset of all possible bout durations in Line 4. This includes all durations less than K time steps in length, and thereafter bout durations that are geometrically increasing in size.

7.6.3 Loss Functions For Behavior Recognition

The loss function measures how well a predicted segmentation Y approximates a ground truth segmentation Y_i . It should be constructed such that when the loss function is small, then the results from the inference are satisfactory. We use

$$\Delta(Y, Y_i) = \sum_{(b,e,c) \in Y_i} \frac{\ell_{fn}^c}{e-b} \left(\bigcap_{Y, \hat{c} \neq c} (b, e) \right) + \sum_{(\hat{b}, \hat{e}, \hat{c}) \in Y} \frac{\ell_{fp}^c}{\hat{e} - \hat{b}} \left(\bigcap_{Y_i, c \neq \hat{c}} (\hat{b}, \hat{e}) \right) \quad (7.78)$$

where $\bigcap_{Y, \hat{c} \neq c} (b, e)$ is the number of frames in Y intersecting with (b, e) with different action class $\hat{c} \neq c$, ℓ_{fn}^c is the cost for missing a bout of class c , and behavior $\ell_{fp}^{\hat{c}}$ is the cost

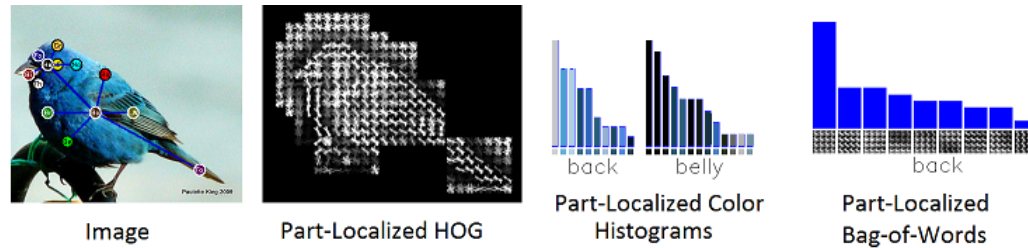


Figure 7.8: Feature Visualization: Three types of features are extracted from the bird image on the left, localized around the depicted part locations. From left to right: 1) 7×7 HOG templates, 2) color histograms, 3) bag-of-words encoded SIFT features.

for incorrectly detecting a bout of class \hat{c} . This loss function softly penalizes predictions where the start or end of the bout is slightly incorrect. On the other hand, since the loss is normalized by the bout duration, it effectively counts the number of incorrectly predicted bouts. Unlike a per-frame loss, behaviors such as walking and standing still (which may last for minutes at a time) are not deemed to be more important than behaviors like lunging and wing extensions (which may last for a fraction of a second). Furthermore, noisy segmentations that break a bout into many smaller bouts are heavily penalized by the false positive cost. As a result, $\Delta(Y, Y_i)$ heavily favors smooth segmentations that can explain the observed data using a small number of bouts.

The function can be generalized to take into account an arbitrary class confusion cost, such that mistaking action c_1 for action c_2 can be penalized more than mistaking action c_3 for action c_2 , if c_3 and c_2 are more similar than c_1 and c_2 .

7.7 Features and Implementation Details

In Sections 7.1-7.5, referenced image features $\phi(X)$ and part-localized features $\psi(\theta; X)$, which were left undefined. In this section, we define details for efficiently computing localized features $\psi(\theta; X)$ for some fixed window size $W \times H$, with respect to some location $\theta = \{x, y, s_p, r\}$. I have included the following features in my software package (they are all efficiently computable in sliding window fashion and applicable to classification, detection, part-based, segmentation, and tracking problems):

- **HOG/SIFT:** Histograms of Oriented Gradients (HOG) [15] capture gradients at

different orientations in different spatial locations. SIFT corresponds to a specific set of parameter choices for HOG. Templates of HOG features are the most popular descriptor for object detection [29, 15].

- **LBP:** Local Binary Patterns (LBP) [51] is a binary vector that is obtained by comparing the relative intensity of pairs of pixels. It is the most popular descriptor for face recognition [2] and is also popular for object detection [82].
- **RGB/Lab Color:** These are simply raw pixel values in some color space

Each of these features can be encoded using templates (Section 7.7.1), bag-of-words (Section 7.7.2), or fisher vectors (Section 7.7.3).

7.7.1 Template Features

Template features are raw values of one of the above feature descriptors on a spatial grid extracted from some window defined by θ . For example, for a template of HOG features in a $W \times H$ window, $\psi(\theta; X)$ is a $W \times H \times C$ descriptor, where C is the number of orientation bins in the HOG descriptor. An RGB template is a $W \times H \times 3$ descriptor. In both cases, the sliding window detection score $\langle \mathbf{w}, \psi(\theta; X) \rangle$ for all possible window locations θ is efficiently computable in time that is $O(n \log n)$ in the number of possible values of θ , but independent of the size of each template window. This can be done by stacking the weight vector \mathbf{w} into a $W \times H$ template and applying a convolution operator (this must be done separately for each image channel).

7.7.2 Bag-of-Words Features

Bag-of-words (applied to computer vision) is a descriptor that is popular for datasets where training images are not spatially aligned. Using bag-of-words on SIFT descriptors is popular for multiclass classification, whereas bag-of-words on RGB/Lab color patches is one way of implementing color histograms. Such features encode statistics on the relative frequency of different colors or textures, but discard information related to the global shape of an object. If bag-of-words is computed within a particular window region $\psi_p(\theta_p; X)$ (e.g., in the window region around a predicted location of a

part p) instead of the entire image, it can be thought of as an encoding of the color/texture of a particular part.

- **Training:** A codebook is trained by extracting image patches and computing one of the above features descriptors (*e.g.*, HOG, SIFT, LBP, RGB, Lab) for each patch. K-means or hierarchical k-means is used to cluster the image patches into k visual codewords.
- **Testing:**
 - **Feature encoding:** To compute a k -dimensional bag-of-words feature vector $\psi(\theta; X)$, image patches are extracted on a regular grid within the window defined by θ , and each patch is assigned to the visual codeword that is the nearest neighbor in Euclidean space. The i -th entry of the induced feature vector $\psi(\theta; X)$ is a count of the number of words assigned to the i -th codeword. The induced feature vector is typically L2-normalized.
 - **Sliding window computation:** Suppose we want to compute the sliding window detection score $\langle \mathbf{w}, \psi(\theta; X) \rangle$ for all possible window locations θ , where \mathbf{w} is a k -vector of weights on codewords. This is efficiently computable in time linear in the number of possible values of θ , but independent of the size of each window (the number of image patches in each window that are used to compute the bag-of-words descriptor) using the integral image trick [76]. An integral image of the detection score is efficiently computable using a single pass over the image. Suppose patch j is assigned to the i -th codeword. The value of the integral image at location t is equal to the value at location $t - 1$ plus the weight for the i -th codeword w_i . A similar method can be used to compute an integral image of the squared Euclidean magnitude for L2 normalization.

7.7.3 Fisher Vector Features

Fisher encoded features [53] can be thought of as a variant of bag-of-words that induces a higher dimensional feature space. They tend to work well in practice. When

used with simple linear classifiers, they typically outperform the best kernelized methods for many popular image classification datasets (*e.g.*, Caltech-256 and ImageNet).

- **Training:** Similar to bag-of-words, a codebook is trained by extracting image patches and computing one of the above features descriptors (*e.g.*, HOG, SIFT, LBP, RGB, Lab) for each patch. A GMM with a diagonal covariance matrix is trained (in place of k-means), and is used to learn k mixture components. Let p_i, μ_i, Σ_i be the parameters of the i -th mixture component, where p_i is a prior probability, μ_i is a d -dimensional mean vector, and Σ_i is a $d \times d$ diagonal covariance matrix.

- **Testing:**

- **Feature encoding:** The Fisher encoded feature vector $\psi(\theta; X)$ will be a $2kd$ -dimensional vector, where d is the dimension of the original feature descriptor (*e.g.*, SIFT). As with bag-of-words, image patches are extracted on a $Q = W \times H$ grid within the window defined by θ ; however, this time each patch $\mathbf{x}_t = x_{t1}, x_{t2}, \dots, x_{td}$ is soft-assigned to each visual codeword, where $\gamma_i(\mathbf{x}_t)$ is the posterior probability that patch \mathbf{x}_t is assigned to the i -th mixture component, according to the GMM distribution. The induced feature vector concatenates the following feature vectors for each value of i :

$$\mathcal{G}_i^\mu = \frac{1}{\sqrt{p_i}} \sum_{t=1}^Q \gamma_i(\mathbf{x}_t) \bar{\mathbf{x}}_{ti} \quad (7.79)$$

$$\mathcal{G}_i^\sigma = \frac{1}{\sqrt{2p_i}} \sum_{t=1}^Q \gamma_i(\mathbf{x}_t) (\bar{\mathbf{x}}_{ti}^2 - 1) \quad (7.80)$$

where $\bar{\mathbf{x}}_{ti} = \Sigma_i^{-.5} (x_t - \mu_i)$ is a d -dimensional vector. The induced vector is normalized as described in [53].

- **Sliding window computation:** Sliding window computation takes time independent of the size of the window in similar fashion as for bag-of-words (using integral images). This time, the value of the integral image at location t is equal to the value at location $t - 1$ plus $\langle \mathbf{w}, [\mathcal{G}_{t,1} \dots \mathcal{G}_{t,k}] \rangle$, where
$$\mathcal{G}_{t,i} = \left[\frac{1}{\sqrt{p_i}} \gamma_i(\mathbf{x}_t) \bar{\mathbf{x}}_{ti}, \frac{1}{\sqrt{2p_i}} \gamma_i(\mathbf{x}_t) (\bar{\mathbf{x}}_{ti}^2 - 1) \right].$$

Acknowledgements

Section 7.6 is based on work with Eyrun Eyjolfsson and Kristin Branson, for which the dissertation author contributed to developing the algorithms and writing. The rest of this chapter is was written by the author of this dissertation.

Chapter 8

Synthetic Distributions For Which Popular Machine Learning Algorithms Perform Poorly

In this section, we consider synthetically generated images that can be defined by a small number of intrinsic parameters (*e.g.*, the number of classes, the number of attributes that classes differ by, and the number of degrees of freedom of geometric deformation). We analyze the scalability of various machine learning techniques that are commonly used in computer vision as a function of the number of intrinsic parameters. Our goal is to come up with synthetic distributions that 1) are simple enough such that we can analytically compute upper bounds on the expected performance of different algorithms, and 2) are parameterized by a set of intrinsic parameters that are intuitively understandable in terms of natural images, such that it is difficult to argue that natural images don't have similar properties.

We argue that many machine learning algorithms will have major scalability problems if objects can deform geometrically (*e.g.*, by moving in 3D or through articulation) and appearance information in multiple portions of the object is necessary for discrimination. In the presence of deformable objects, we demonstrate that methods based on linear classifiers will get poor performance even with an infinite amount of training data, and methods based on nearest neighbor will require an amount of training examples that is exponential in the number of degrees of freedom of articulation of the

object. Additionally, we show that performance of methods based on bag-of-words or spatial histogramming—the most popular type of feature that is designed to cope with geometric deformation of objects—will experience a saturation in performance as the number of classes increases. By contrast, on the synthetic distribution described in this section, a method that explicitly models and annotates object deformation (Section 7.3.2) produces a perfect classifier with just one training example per class. We attempt to relate these results to performance trends in computer vision research.

8.1 Synthetic Distribution Definition

In our simplified object model, we assume that a set of C classes $c_1, c_2 \dots c_C$ can be represented using a shared vocabulary of k low-level visual attributes and P parts. Each class c can be represented by a binary class-attribute vector \mathbf{a}^c of length M , where $M = Pk$. For example, the k low-level visual attributes may correspond to things such as colors, materials, or low-level shapes in some local region, whereas each of the M class-level attributes correspond to the association of a low-level attribute with a specific part (*e.g.*, blue wing, striped belly).

The parameters C , k , and P are intrinsic parameters of our synthetic distribution. The hamming distance $h(\mathbf{a}^{c_1}, \mathbf{a}^{c_2})$ between two class attribute vectors is a measure of the difference between classes c_1 and c_2 (*e.g.*, interclass variation). The decomposition of class-attributes into low-level visual attributes and parts is introduced in order to have a parameter controlling the relative importance of spatial or shape information (the ratio of P to k), which tends to break bag-of-words methods. Additionally, the parameter P controls the number of degrees of freedom of geometric deformation of an object. This is a form of intraclass variance that is known to be problematic to computer vision algorithms.

For the next two sections, we assume images are randomly generated using the following procedure:

1. Let \mathcal{A} be the space of all 2^M possible class-attribute vectors. For each class $c = 1 \dots C$, select a class attribute vector \mathbf{a}^c uniformly at random without replacement.
2. For each class c , synthesize n training examples $X_{c1}, X_{c2}, \dots, X_{cn}$ as visualized in

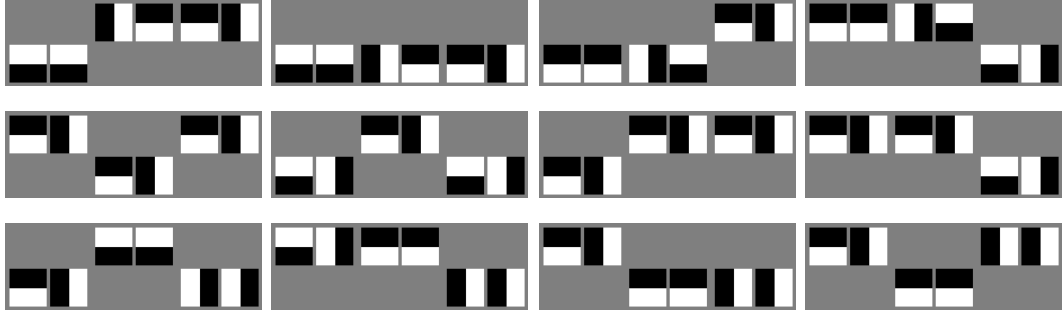


Figure 8.1: Synthetic Distribution For Which Linear Classifiers and Nearest Neighbor Do Poorly: Images are generated using the procedure described in Section 8.1, with $C = 3$ classes, $n = 4$ examples per class, $P = 3$ parts, and $k = 2$ attributes per part. Each row shows 4 images from each of the 3 classes, and classes have class-attribute vectors $\mathbf{a}^{c1} = 001001$, $\mathbf{a}^{c2} = 010101$, $\mathbf{a}^{c3} = 010011$

Figure 8.1. For each such example X :

- (a) For each part p , independently choose a part location $\theta_p = \{i_p, f_p\}$, where i_p is a random integer between 0 and $l = h/w$, and b_p is a Bernoulli random variable with probability .5.
- (b) Represent each low level visual attribute by a square of size $w \times w$ pixels that is half white (pixels are 1) and half black (pixels are -1). If an attribute $a_{pi} = 1$ the square will be split vertically in half, whereas if $a_{pi} = -1$ the square will be split horizontally in half.
- (c) Render a gray background (pixels are 0) of size $Mw \times h$ pixels, where $h = lw$.
- (d) Draw the square for the j_{th} attribute of the p_{th} part at xy -location $(w(pk + j), wi_p)$. If $f_p = \text{true}$, then also rotate the squares by 180° .

This definition is a little complicated to explain, and in general any type of object where different portions of the object deform separately will have similar problems. However, the defined synthetic distribution has a few properties that make it easier to analyze. Note that irrespective of the class or attribute, the expected value of image intensities at all pixel locations is 0 due to equal probability of rotating parts. In Sections 8.2 and 8.3, we show that when using image pixels as features, linear classifiers and nearest neighbor do poorly on this distribution. While we show this only for pixel

features, similar results will hold for other types of features that preserve spatial information like HOG, spatial pyramids, filter responses (not necessarily on this particular distribution, but for some related distribution). In Section 8.4, we show that features based on histogramming or bag-of-words will also have poor scaling properties. By contrast, on this particular distribution, a part-based structured model (*e.g.*, as described in Section 7.2.3) is capable of getting perfect classification performance with $n = 1$ training example per class.

8.2 No Linear Classifier Is Better Than Random Chance

The rendering of image pixels for an example (X, c) is a deterministic function $I(X, c, \Theta)$ of the attribute parameters \mathbf{a}^c and the part location parameters $\Theta = i_1 \dots i_P, f_1 \dots f_p$, where $I(X, c, \Theta)$ is a vector of $Mw \times h$ pixels. Let us define the quantity $\text{flip}(\Theta) = i_1 \dots i_P, \bar{f}_1 \dots \bar{f}_p$, which is obtained by negating all parameters $f_1 \dots f_p$, such that each rendered square is rotated by 180° . Consequently, it follows that

$$I(X, c, \Theta) = -I(X, c, \text{flip}(\Theta)) \quad (8.1)$$

We show that there does not exist a linear classifier operating on pixels as features that does better than random chance. Let us consider two different classes c_1 and c_2 . Let \mathbf{w}_1, t_1 each be a vector of weights and a threshold, such that c_1 is predicted over c_2 . If $\langle \mathbf{w}_1, I(X, c, \Theta) \rangle > t_1$. Let Θ be a set of part locations that renders an image $I(X_1, c_1, \Theta)$ for which our classifier predicts the correct class c_1 . This implies that

$$\langle \mathbf{w}_1, I(X_1, c_1, \Theta) \rangle > t \quad (8.2)$$

By Eq 8.1, it must be the case that

$$\langle \mathbf{w}_1, -I(X_1, c_1, \text{flip}(\Theta)) \rangle > t \quad (8.3)$$

$$\langle \mathbf{w}_1, I(X_1, c_1, \text{flip}(\Theta)) \rangle < -t \quad (8.4)$$

Consequently, the image $I(X_1, c_1, \text{flip}(\Theta))$ must be incorrectly classified. Since Θ and $\text{flip}(\Theta)$ have equal probability mass, this implies that the expected classification accuracy of a 1-vs-1 classifier cannot be greater than .5.

8.3 Nearest Neighbor Requires Exponential Data

Assume that we synthesize images using the process described in the previous section. Intuitively, a nearest neighbor classifier will likely fail unless there exists a training example with all parts in the same configuration as a test example, and thus will require an exponentially large amount of training data in the number of parts.

For images X_1 and X_2 that are randomly generated from classes c_1 and c_2 respectively, let $d_p(X_1, X_2)$ denote the Euclidean distance in the portion of the images where the p th part could be located (the $kw \times h$ rectangle with upper left corner at $x = wpk$, $y = 0$). Let $h(\mathbf{a}_p^{c_1}, \mathbf{a}_p^{c_2})$ be the hamming distance between class-attributes that pertain to the p th part. Then

$$d(X_1, X_2) = \sum_p d_p^P(X_1, X_2) \quad (8.5)$$

$$d_p(X_1, X_2) = \begin{cases} kw^2 & \text{if } i_1 \neq i_2 \text{ (w/ probability } 1 - \frac{1}{l}) \\ h(\mathbf{a}_p^{c_1}, \mathbf{a}_p^{c_2})w^2 & \text{if } \theta_1 = \theta_2 \text{ (w/ probability } \frac{1}{2l}) \\ (2k - h(\mathbf{a}_p^{c_1}, \mathbf{a}_p^{c_2}))w^2 & \text{if } i_1 = i_2 \text{ and } f_1 \neq f_2 \text{ (w/ probability } \frac{1}{2l}) \end{cases} \quad (8.6)$$

Let us assume that for class c there exists another class c' with no more than k different attributes $h(\mathbf{a}_p^c, \mathbf{a}_p^{c'}) \leq k$. Let X be a random test example of class c and part locations Θ , and suppose a training set of $n = (2l)^P$ examples per class is generated with random part locations, yielding a train set that includes $(X_{c1}, c, \Theta_{c1}) \dots (X_{cn}, c, \Theta_{cn})$ and $(X_{c'1}, c', \Theta_{c'1}) \dots (X_{c'n}, c', \Theta_{c'n})$. Then the expected loss of a nearest neighbor classifier $\mathbb{E}\Delta(X)$ is at least one half the probability that there exists a training example of class c' that is at least as close as the nearest example of class c :

$$\mathbb{E}_{\Theta, \Theta_{c1} \dots \Theta_{cn}, \Theta_{c'1} \dots \Theta_{c'n}} \Delta(X) \geq \frac{1}{2} p(\min_i d(X, X_{ci}) \geq \min_j d(X, X_{c'j})) \quad (8.7)$$

$$\geq \frac{1}{2} p(\exists_i, \Theta_{ci} \neq \Theta) p(\exists_j, \Theta_{c'j} = \Theta) \quad (8.8)$$

$$\geq \frac{1}{2} \left(1 - \frac{1}{(2l)^P}\right)^n \left(1 - \left(1 - \frac{1}{(2l)^P}\right)^n\right) \quad (8.9)$$

$$\geq .1 \quad (8.10)$$

where the last line holds because $.316 \leq (1 - \frac{1}{n})^n \leq \frac{1}{e}$ for $n \geq 4$. We thus expect nearest neighbor algorithms to require an exponentially large amount of training data in

the number of degrees of freedom of object deformation. In this example, it also scales badly with the parameter l —this can roughly be interpreted as the ratio between the number of pixels a part can spatially deform to the resolution of appearance information that is needed to detect the part.

8.4 Bag-of-Words Scales Badly With Number of Classes

Bag-of-words based features are the most popular and successful type of feature on object recognition datasets such as Caltech-256 [31], the VOC Classification Challenge [23], and ImageNet [18] (datasets that typically have only one object per image and don't provide annotation of object location). Empirically, people have observed that these methods tend to work well when the number of possible classes is small (*e.g.*, less than 20), but experience a steady drop off in performance as the number of classes increases (see Fig 8.2). Furthermore, there is a general sentiment in the field that research in such methods has become increasingly saturated, with very little quantitative improvement in performance in the last 5 years. We explore the possibility that this performance drop off and saturation is a consequence of an over-reliance on bag-of-words or histogram features.

In computer vision, bag-of-words methods discard information about the relative ordering or geometric arrangement of different visual words. This can potentially still work if there is a sufficiently large number of differences between classes, such that classes are distinguishable even with the loss of spatial information. However, we expect performance to decline as 1) the number of classes grows such that the number of attributes in which classes differ by decreases, 2) the relative importance of spatial information becomes more important (in our model, this is related to the ratio of P to k).

In this section, our analysis applies to any bag-of-words method, irrespective of the method of synthesizing images from class attribute vectors. We compute an upper bound on how well bag-of-words based systems can perform, even if we assume that researchers could solve the following problems: 1) the selection of features and classifier is perfect, 2) infinite training data, 3) perfect codebook learning, such that the learned

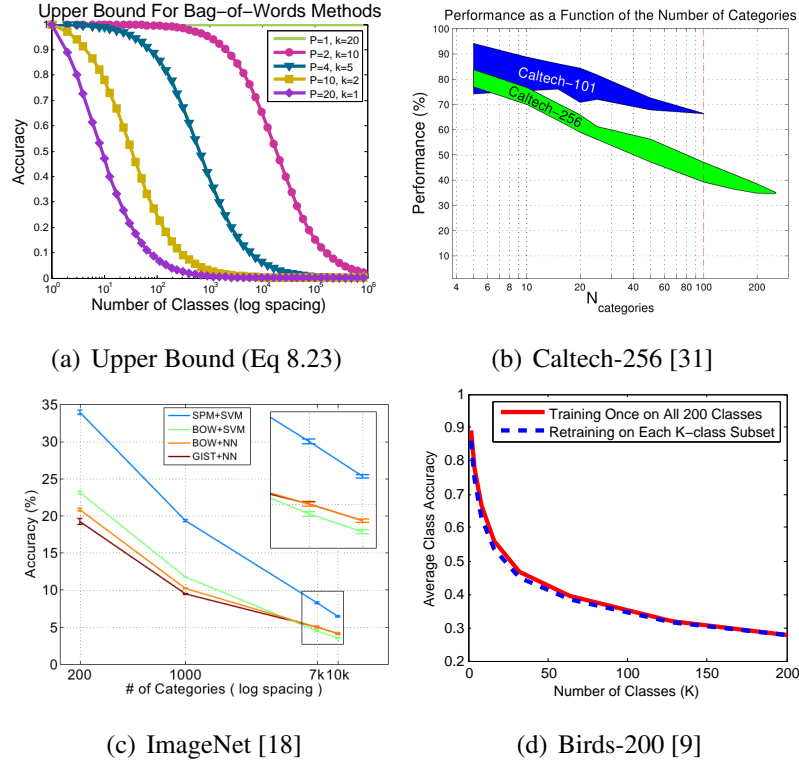


Figure 8.2: Object recognition methods scale poorly with the number of classes. Is this partially a consequence of an over-reliance on bag-of-words? a) An upper bound on the classification accuracy of bag-of-words methods for our synthetic distribution, with different parameter settings for the number of parts P and number of attributes $M = Pk$ (see Eq 8.23), predicts poor performance of methods based on bag-of-words as the number of classes increases. b-d) Plots taken from [31, 18, 9] show that multiclass classification accuracy for state-of-the-art methods scales badly with the number of classes on Caltech-256, ImageNet, and Birds-200.

visual words correspond to the underlying k visual attributes of the generating distribution, and 4) perfect region of interest selection (segmentation), such that codewords are sampled exactly from the object and not the background. All error in this upper bound can be attributed to the loss of spatial information from the bag-of-words assumption. In this case, for a data example x of class c , we assume that the idealized detected bag-of-words feature space can be expressed as a vector of length k , such that the j_{th} feature $\phi_j(x)$ is the total number of occurrences of the j_{th} attribute:

$$\phi_j(x) = \sum_{p=1}^p 1[a_{pj}^c = 1] \quad (8.11)$$

where $a_{pj}^c = \pm 1$ indicates whether or not part p in class c has the j_{th} attribute. As described in the previous section, we assume class attribute vectors are selected uniformly at random and each class occurs with equal prior probability. Let c_1 and c_2 be two randomly selected classes. The probability that both classes will map to the same bag-of-words feature vector is

$$p(\phi(c_1) = \phi(c_2)) = \sum_{\phi(c_1)} p(\phi(c_1))p(\phi(c_1) = \phi(c_2)) \quad (8.12)$$

$$p(\phi(c_1) = \phi(c_2)) = \sum_{\phi(c_1)} p(\phi(c_1))^2 \quad (8.13)$$

where the second line occurs because classes are selected independently at random with replacement. Since attributes are selected independently

$$p(\phi(c_1) = \phi(c_2)) = \sum_{\phi(c_1)} \left(\prod_{j=1}^k p(\phi_j(c_1)) \right)^2 \quad (8.14)$$

$$= \prod_{j=1}^k \sum_{\phi_j(c_1)=0}^P p(\phi_j(c_1))^2 \quad (8.15)$$

$$= \left(\sum_{\phi_j(c_1)=0}^P p(\phi_j(c_1))^2 \right)^k \quad (8.16)$$

Since $\phi_j(c_1)$ is a count of the number of parts with the j_{th} attribute, $\phi_j(c_1) \sim \text{BINOMIAL}(P, .5)$ and

$$p(\phi(c_1) = \phi(c_2)) = \left(\sum_{k=0}^P \left(\frac{\binom{P}{k}}{2^P} \right)^2 \right)^k \quad (8.17)$$

The sum of squares of binomial coefficients $\sum_{k=0}^P \binom{P}{k}^2$ is known to equal the central binomial coefficient¹

$$\sum_{k=0}^P \binom{P}{k}^2 = \binom{2P}{P} = \frac{2^P (2P-1)!!}{P!} = \frac{2^{2P} \Gamma(P + \frac{1}{2})}{\sqrt{\pi} \Gamma(P+1)} \quad (8.18)$$

where Γ is the gamma function. It follows that

$$p(\phi(c_1) = \phi(c_2)) = \left(\frac{\Gamma(P + \frac{1}{2})}{\sqrt{\pi} \Gamma(P+1)} \right)^k \quad (8.19)$$

¹<http://mathworld.wolfram.com/BinomialSums.html>

By contrast, the probability that two classes have the same class-attribute vectors (since they are drawn without replacement) is

$$p(a^{c_1} = a^{c_2}) = \frac{1}{2^{Pk}} \quad (8.20)$$

and the probability that two random classes with different class-attribute vectors collide is

$$p(\phi(c_1) = \phi(c_2) | a^{c_1} \neq a^{c_2}) = \left(\frac{\Gamma(P + \frac{1}{2})}{\sqrt{\pi}\Gamma(P + 1)} \right)^k - \frac{1}{2^{Pk}} \quad (8.21)$$

Thus given C randomly selected classes, the expected number of classes that have the same bag-of-words feature representation as a particular class c_1 is:

$$\mathbb{E}_{c_1 \dots c_C} \sum_{j=2}^C p(\phi(c_1) = \phi(c_j)) = (C - 1) \left(\left(\frac{\Gamma(P + \frac{1}{2})}{\sqrt{\pi}\Gamma(P + 1)} \right)^k - \frac{1}{2^{Pk}} \right) \quad (8.22)$$

In the event of multiple examples that map to the same feature vector, we assume ties are broken by random guessing. Thus we can establish an upper bound on the expected classification accuracy of our bag-of-words based method as:

$$A_{bow}(C, P, k) = \frac{1}{1 + (C - 1) \left(\left(\frac{\Gamma(P + \frac{1}{2})}{\sqrt{\pi}\Gamma(P + 1)} \right)^k - \frac{1}{2^{Pk}} \right)} \quad (8.23)$$

Thus this model predicts that the maximum classification accuracy is inversely proportional to the total number of classes (although the maximum number of classes is 2^{Pk}). A more intuitively understandable version of Eq 8.23 is shown below:

$$A_{bow}(C, 1, k) = 1 \quad (8.24)$$

$$A_{bow}(C, 2, k) = \frac{1}{1 + (C - 1) \left(\left(\frac{3}{8} \right)^k - \left(\frac{1}{4} \right)^k \right)} \quad (8.25)$$

$$\forall_{P \geq 3}, A_{bow}(C, P, k) \leq \frac{1}{1 + (C - 1) \left(\frac{1}{P} \right)^k} \quad (8.26)$$

Thus it is easy to see that classification accuracy will become close to zero as C increases toward 2^{Pk} as long as $P \neq 1$. For a fixed value of C , increasing both P and k improve the accuracy of bag-of-words (because they both increase the number of attributes that classes differ by); however increasing k has a much more significant effect than increasing P . We plot different values.

Acknowledgements

This chapter is based on work in progress by the author of this dissertation.

Chapter 9

Conclusion

9.1 Summary of Contributions

In this dissertation, we explored three types of interactive methods in computer vision: interactive computer vision applications, interactive learning methods, and interactive feedback mechanisms to researchers. The specific contributions of this dissertation include:

- We introduced two different human-in-the-loop methods that enhance state-of-the-art learned computer vision models into applications that are more usable in the real world. The first builds off existing methods for multiclass recognition or attribute-based recognition. It can be used to create practical systems for classifying objects that are recognizable by people with appropriate expertise (*e.g.*, animal species or airplane model), but not (in general) by people without such expertise. The second builds off existing methods for part-based localization and pose registration, and has been used to create a tool for semantic annotation of images.
- We proposed a simple form of active learning that consists of alternating between online learning and semi-automated interactive annotation. This method makes it realistic to scale computer vision problems to more complex image models, while keeping annotation time and computational costs of learning tractable.

- We introduced faster customized learning algorithms for object detection and deformable part model training.
- We introduced feedback mechanisms to help diagnose different sources of test error due to insufficient training data, a bad model or feature space, annotation error, or insufficient computation time.
- We introduced a structured learning and annotation software package that integrates these 3 components (interactive labeling, online learning, and feedback mechanisms)
- We introduced a method for behavior detection and segmentation that uses a more appropriate time-localized model of behavior

9.2 Future Work

- *Deployment of largescale real world recognition systems:* We are currently collaborating with the Cornell Lab of Ornithology to create a web-based system for bird species recognition. We are also planning on expanding methods to other types of domains, such as insect species classification or car model recognition.
- *Collecting large datasets with interactive learning:* We are also collaborating to collect a large dataset of richly annotated images of North American birds with part and attribute annotations. In doing so, we are developing a generic set of annotation tools that integrate with our interactive learning methods.
- *Mechanisms supporting evolving definitions of classes, parts, and attributes:* In building Visipedia, we are constructing authoring tools for different types of semantic representations of images. This is important not only as a mechanism to allow users to define what problems we want computer vision to solve, but also as a means for researchers to explore what representations are appropriate to achieve better performance. In doing so, we would like to create an augmented set of diagnosis techniques that integrate with part and attribute vocabulary authoring tools and online learning techniques.

Bibliography

- [1] Y. Abramson and Y. Freund. Semi-automatic visual learning (seville): a tutorial on active learning for visual object recognition. In *Proc. CVPR*, 2005.
- [2] T. Ahonen, A. Hadid, and M. Pietikäinen. Face recognition with local binary patterns. *ECCV*, 2004.
- [3] P. Belhumeur, D. Chen, S. Feiner, D. Jacobs, W. Kress, H. Ling, I. Lopez, R. Ramamoorthi, S. Sheorey, S. White, and L. Zhang. Searching the world’s herbaria: A system for visual identification of plant species. In *ECCV*, pages 116–129, 2008.
- [4] M. Beynon, D. Cosker, and D. Marshall. An expert system for multi-criteria decision making using Dempster Shafer theory. *Expert Systems with Applications*, 20(4), 2001.
- [5] M. Blaschko and C. Lampert. Learning to localize objects with structured output regression. *ECCV*, 2008.
- [6] L. Bourdev and J. Malik. Poselets: Body part detectors trained using 3d human pose annotations. In *Computer Vision, 2009 IEEE 12th International Conference on*, pages 1365–1372. IEEE, 2010.
- [7] Y. Boykov and G. Funka-Lea. Graph cuts and efficient nd image segmentation. *International Journal of Computer Vision*, 70(2):109–131, 2006.
- [8] S. Branson, P. Perona, and S. Belongie. Strong supervision from weak annotation: Interactive training of deformable part models. In *ICCV*, 2011.
- [9] S. Branson, C. Wah, F. Schroff, B. Babenko, P. Welinder, P. Perona, and S. Belongie. Visual recognition with humans in the loop. In *ECCV*, 2010.
- [10] S. Branson, C. Wah, F. Schroff, B. Babenko, P. Welinder, P. Perona, and S. Belongie. Visual Recognition with Humans in the Loop. *ECCV*, 2010.
- [11] L. Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.
- [12] L. Cai and T. Hofmann. Hierarchical document categorization with support vector machines. In *CIKM*, 2004.

- [13] C. Cortes and V. Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.
- [14] K. Crammer and Y. Singer. On the learnability and design of output codes for multiclass problems. *M.L.*, 2002.
- [15] N. Dalal and B. Triggs. Hog for human detection. In *CVPR*, 2005.
- [16] A. Dembo, T. Cover, and J. Thomas. Information theoretic inequalities. *IEEE Transactions on Information Theory*, 37(6):1501–1518, 1991.
- [17] J. Deng, A. C. Berg, K. Li, and L. Fei-Fei. What does classifying more than 10,000 image categories tell us? In *ECCV*, 2010.
- [18] J. Deng, W. Dong, R. Socher, L. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. *CVPR*, 2009.
- [19] C. Desai, D. Ramanan, and C. Fowlkes. Discriminative models for multi-class object layout. In *CVPR*, 2009.
- [20] T. Dietterich and G. Bakiri. Solving multiclass learning problems via error-correcting output codes. *Journal of Artificial Intelligence Research*, 2:263–286, 1995.
- [21] T. Dietterich, R. Lathrop, and T. Lozano-Pérez. Solving the multiple instance problem with axis-parallel rectangles. *Artificial Intelligence*, 89(1-2):31–71, 1997.
- [22] P. Dollár, B. Babenko, S. Belongie, P. Perona, and Z. Tu. Multiple component learning for object detection. *Computer Vision–ECCV 2008*, pages 211–224, 2008.
- [23] M. Everingham, L. Van Gool, C. Williams, J. Winn, and A. Zisserman. The PASCAL visual object classes (VOC) challenge. *IJCV*, 2010.
- [24] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The PASCAL VOC Challenge 2009 Results, 2009.
- [25] R. Fan, K. Chang, C. Hsieh, X. Wang, and C. Lin. LIBLINEAR: A library for large linear classification. *The Journal of Machine Learning Research*, 9:1871–1874, 2008.
- [26] A. Farhadi, I. Endres, D. Hoiem, and D. Forsyth. Describing objects by their attributes. In *CVPR*, 2009.
- [27] R. Farrell, O. Oza, N. Zhang, V. Morariu, T. Darrell, and L. Davis. Birdlets: Subordinate categorization using volumetric primitives and pose-normalized appearance. In *Computer Vision (ICCV), 2011 IEEE International Conference on*, pages 161–168. IEEE, 2011.

- [28] P. Felzenszwalb and D. Huttenlocher. Efficient matching of pictorial structures. In *Computer Vision and Pattern Recognition, 2000. Proceedings. IEEE Conference on*, volume 2, pages 66–73. IEEE, 2002.
- [29] P. Felzenszwalb, D. McAllester, and D. Ramanan. A discriminatively trained, multiscale, deformable part model. In *CVPR*, 2008.
- [30] Y. Freund and R. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. In *Computational learning theory*, pages 23–37. Springer, 1995.
- [31] G. Griffin, A. Holub, and P. Perona. Caltech-256 object category dataset. Technical Report 7694, California Institute of Technology, 2007.
- [32] G. Griffin and P. Perona. Learning and using taxonomies for fast visual categorization. In *CVPR*, pages 1–8, 2008.
- [33] E. Hazan, A. Agarwal, and S. Kale. Logarithmic regret algorithms for online convex optimization. *ML*, 2007.
- [34] A. Holub, P. Perona, and M. Burl. Entropy-based active learning for object recognition. In *Workshop on Online Learning for Classification (OLC)*, pages 1–8, 2008.
- [35] T. Joachims. Training linear SVMs in linear time. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 217–226. ACM, 2006.
- [36] S. Kakade and S. Shalev-Shwartz. Mind the duality gap: Logarithmic regret algorithms for online optimization. *NIPS*, 2008.
- [37] A. Kapoor, K. Grauman, R. Urtasun, and T. Darrell. Active learning with gaussian processes for object categorization. In *ICCV*, pages 1–8, 2007.
- [38] S. Keerthi, S. Sundararajan, K. Chang, C. Hsieh, and C. Lin. A sequential dual method for large scale multi-class linear svms. In *SIGKDD*, 2008.
- [39] N. Kumar, A. C. Berg, P. N. Belhumeur, and S. K. Nayar. Attribute and Simile Classifiers for Face Verification. In *ICCV*, 2009.
- [40] C. Lampert, H. Nickisch, and S. Harmeling. Learning to detect unseen object classes by between-class attribute transfer. In *CVPR*, 2009.
- [41] S. Lazebnik, C. Schmid, and J. Ponce. A maximum entropy framework for part-based texture and object recognition. In *ICCV*, volume 1, pages 832–838, 2005.
- [42] S. Lazebnik, C. Schmid, and J. Ponce. Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories. In *CVPR*, 2006.

- [43] A. Levin, D. Lischinski, and Y. Weiss. A closed-form solution to natural image matting. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 228–242, 2007.
- [44] S. Maji and A. Berg. Max-margin additive classifiers for detection. In *Computer Vision, 2009 IEEE 12th International Conference on*, pages 40–47. IEEE, 2010.
- [45] S. Maji, A. Berg, and J. Malik. Classification using intersection kernel support vector machines is efficient. In *CVPR*, 2008.
- [46] Martinez-Munoz et al. Dictionary-free categorization of very similar objects via stacked evidence trees. In *CVPR*, 2009.
- [47] T. Mensink, J. Verbeek, and G. Csurka. Learning structured prediction models for interactive labeling. In *CVPR*, 2011.
- [48] R. E. Neapolitan. *Probabilistic reasoning in expert systems: theory and algorithms*. John Wiley & Sons, Inc., New York, NY, USA, 1990.
- [49] M. Nilsback and A. Zisserman. Automated flower classification over a large number of classes. In *Indian Conf. on Comp. Vision, Graphics & Image Proc.*, pages 722–729, 2008.
- [50] D. Nister and H. Stewenius. Recognition with a vocabulary tree. In *CVPR*, 2006.
- [51] T. Ojala, M. Pietikainen, and T. Maenpaa. Multiresolution gray-scale and rotation invariant texture classification with local binary patterns. *PAMI*, 2002.
- [52] P. Ott and M. Everingham. Shared parts for deformable part-based models. In *CVPR*, 2011.
- [53] F. Perronnin, J. Sánchez, and T. Mensink. Improving the fisher kernel for large-scale image classification. *ECCV*, 2010.
- [54] J. Platt. Probabilities for SV machines. In *NIPS*, pages 61–74, 1999.
- [55] J. C. Platt. Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods. In *Advances in Large Margin Classifiers*, 1999.
- [56] J. R. Quinlan. *C4.5: programs for machine learning*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993.
- [57] N. Ratliff. (Online) Subgradient Methods for Structured Prediction. In *AISTats*, 2007.
- [58] C. Rother, V. Kolmogorov, and A. Blake. Grabcut: Interactive foreground extraction using iterated graph cuts. *ACM Transactions on Graphics (TOG)*, 23(3):309–314, 2004.

- [59] B. Sapp, C. Jordan, and B. Taskar. Adaptive pose priors for pictorial structures. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 422–429. IEEE, 2010.
- [60] B. Settles. *Curious machines: active learning with structured instances*. 2008.
- [61] B. Settles. Closing the loop: Fast, interactive semi-supervised annotation with queries on features and instances. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 2011.
- [62] S. Shalev-Shwartz, O. Shamir, N. Srebro, and K. Sridharan. Stochastic convex optimization. In *COLT*, 2009.
- [63] S. Shalev-Shwartz, Y. Singer, and N. Srebro. Pegasos: Primal estimated sub-gradient solver for SVM. In *ICML*, 2007.
- [64] Q. Shi, L. Wang, L. Cheng, and A. Smola. Discriminative human action segmentation and recognition using semi-markov model. In *CVPR*, 2008.
- [65] J. Sivic, B. Russell, A. Zisserman, W. Freeman, and A. Efros. Unsupervised discovery of visual object class hierarchies. In *CVPR*, pages 1–8, 2008.
- [66] B. Taskar, C. Guestrin, and D. Koller. Max-margin markov networks. In *NIPS*, volume 16, page 25. MIT Press, 2004.
- [67] B. Taskar, S. Lacoste-Julien, and M. Jordan. Structured prediction via the extra-gradient method. 2005.
- [68] S. Tong and D. Koller. Support vector machine active learning with applications to text classification. *JMLR*, 2:45–66, 2002.
- [69] A. Torralba, K. Murphy, and W. Freeman. Sharing features: efficient boosting procedures for multiclass object detection. In *CVPR*, volume 2, 2004.
- [70] S. Tsang, B. Kao, K. Yip, W. Ho, and S. Lee. Decision trees for uncertain data. In *International Conference on Data Engineering (ICDE)*, 2009.
- [71] I. Tsochantaridis, T. Joachims, T. Hofmann, and Y. Altun. Large margin methods for structured and interdependent output variables. *JMLR*, 2006.
- [72] A. Vedaldi, V. Gulshan, M. Varma, and A. Zisserman. Multiple kernels for object detection. In *ICCV*, 2009.
- [73] A. Vedaldi, V. Gulshan, M. Varma, and A. Zisserman. Multiple kernels for object detection. In *Computer Vision, 2009 IEEE 12th International Conference on*, pages 606–613. IEEE, 2009.

- [74] S. Vijayanarasimhan and K. Grauman. What's It Going to Cost You?: Predicting Effort vs. Informativeness for Multi-Label Image Annotations. In *CVPR*, 2009.
- [75] S. Vijayanarasimhan and K. Grauman. Large-scale live active learning. In *CVPR*, 2011.
- [76] P. Viola and M. Jones. Robust real-time object detection. *IJCV*, 2001.
- [77] C. Vondrick, D. Ramanan, and D. Patterson. Efficiently Scaling Up Video Annotation with Crowdsourced Marketplaces. In *Proceedings of European Conference on Computer Vision*, 2010.
- [78] C. Wah, S. Branson, P. Perona, and S. Belongie. Multiclass recognition and part localization with humans in the loop. In *ICCV*, 2011.
- [79] C. Wah, S. Branson, P. Welinder, P. Perona, and S. Belongie. The Caltech-UCSD Birds-200-2011 Dataset. FGVC, 2011.
- [80] C. Wah, S. Branson, P. Welinder, P. Perona, and S. Belongie. The Caltech-UCSD Birds-200-2011 Dataset. Technical Report CNS-TR-2011-001, Caltech, 2011.
- [81] M. Waite. whatbird.com. <http://www.whatbird.com/>.
- [82] X. Wang, T. Han, and S. Yan. An hog-lbp human detector with partial occlusion handling. In *ICCV*, 2009.
- [83] Y. Wang and G. Mori. A discriminative latent model of object classes and attributes. *ECCV*, 2010.
- [84] M. Weber, M. Welling, and P. Perona. Unsupervised learning of models for recognition. *Computer Vision-ECCV 2000*, pages 18–32, 2000.
- [85] P. Welinder, S. Branson, S. Belongie, and P. Perona. The multidimensional wisdom of crowds. In *NIPS*, 2010.
- [86] P. Welinder, S. Branson, T. Mita, C. Wah, F. Schroff, S. Belongie, and P. Perona. Caltech-UCSD Birds 200. Technical report, California Institute of Technology and UC San Diego, 2010.
- [87] W. Wu and J. Yang. SmartLabel: an object labeling tool using iterated harmonic energy minimization. In *Proceedings of the 14th annual ACM international conference on Multimedia*, pages 891–900. ACM, 2006.
- [88] Y. Yang and D. Ramanan. Articulated Pose Estimation using Flexible Mixtures of Parts. In *CVPR*, 2011.

- [89] B. Yao, X. Yang, and S. Zhu. Introduction to a large-scale general purpose ground truth database: methodology, annotation tool and benchmarks. In *Energy minimization methods in computer vision and pattern recognition: 6th international conference, EMMCVPR 2007, Ezhou, China, August 27-29, 2007; proceedings*, page 169. Springer-Verlag New York Inc, 2007.
- [90] C. Yu and T. Joachims. Learning structural SVMs with latent variables. In *Proceedings of the 26th Annual International Conference on Machine Learning*. ACM, 2009.
- [91] J. Yuen, B. Russell, C. Liu, and A. Torralba. Labelme video: building a video database with human annotations. In *Computer Vision, 2009 IEEE 12th International Conference on*, pages 1451–1458. IEEE, 2010.
- [92] X. Zhou and T. Huang. Relevance feedback in image retrieval: A comprehensive review. *Multimedia Systems*, 8(6):536–544, 2003.
- [93] X. Zhu and D. Ramanan. Face detection, pose estimation, and landmark localization in the wild. In *CVPR*, 2012.