Strong Supervision From Weak Annotation: Interactive Training of Deformable Part Models

Steve Branson[1], Pietro Perona[2], Serge Belongie[1]

[1] University of California, San Diego
La Jolla, CA, 92093, USA
{sbranson,sjb}@cs.ucsd.edu

[2] California Institute of Technology
Pasadena, CA, 91125, USA
perona@caltech.edu

Abstract

We propose a framework 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 framework is scalable to large datasets and complex image models and is shown to have excellent theoretical and practical properties in terms of train time, optimality guarantees, and bounds on the amount of annotation effort per image. We apply this framework to part-based detection, and introduce a novel algorithm for interactive labeling of deformable part models. The labeling tool updates and displays in real-time the maximum likelihood location of all parts as the user clicks and drags the location of one or more parts. 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.

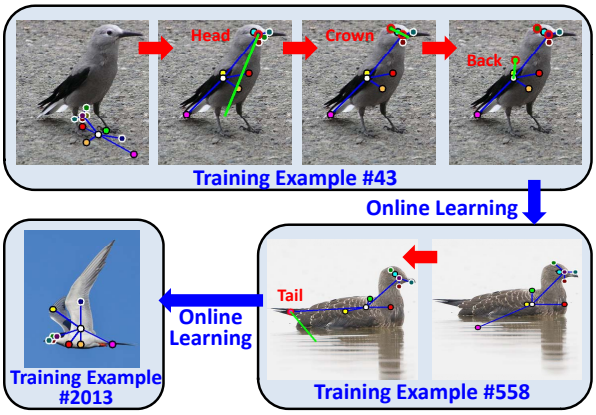


Figure 1. **Interactive Labeling and Online Learning of Part Models:** A part model is trained in online fashion, where annotation becomes increasingly automated as more images are labeled. The diagram shows how the interactive labeling interface changes on a particular test image as the size of the training set increases, with green lines representing parts that were dragged by the user.

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 [10, 3, 7] 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 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 [6], latent parts [10], latent structural SVMs [30], and expectation maximization on constellation models [25]. 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 anno-

tation 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 [3] 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 [29] of Zhu et al. echoes this sentiment.

In this paper, we ask the question *is it possible to maintain the benefits of strongly supervised methods–computational tractability, performance guarantees, and scalability to models of greater complexity–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. 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.

Contributions: There are two main contributions of this paper: (1) We propose a framework for scalable annotation of structured models that interleaves online learning and interactive labeling, and show that it has excellent practical and theoretical properties in terms of optimality guarantees, bounds on the amount of annotation effort per training image, and computational scalability. (2) We introduce an algorithm and UI for realtime interactive labeling of deformable part models. While this UI is particularly suited to be used in conjunction with an online structured learning algorithm, it is also useful as a standalone application. For example, it could be used for semi-automated annotation of biomedical images or as an object-specific smart plugin for photo-editing software. In these applications, achieving a high-degree of accuracy is more important than having a fully automated system, and a motivated human-in-the-loop can help correct imperfect computer vision systems.

Interactive Labeling of Deformable Part Models: Our interactive labeling interface applies to a variant of the popular deformable part model of Felzenszwalb et al. [10]. Our model employs semantically meaningful, strongly supervised parts and uses mixture models to handle multiple poses or aspects of an object, as in [28, 15, 25]. We introduce efficient algorithms which update and display in realtime the maximum likelihood location of all parts as the user drags one or more parts with the mouse. We encourage the reader to watch videos of the annotation system in the supplementary material before reading this paper.

Online Structured Learning: We employ online algorithms which optimize a structured SVM objective function [20], a convex optimization problem that has been applied to a wide variety of different problems in computer vision [2, 28, 15, 5, 19]. While the $\mathrm{SVM}^{\mathrm{struct}}$ solver is most commonly used, online optimization algorithms have been observed to be faster in practice [5, 28]. This result is supported theoretically by results relating to online learning algorithms for strongly convex loss functions [12, 11, 18, 16].

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 [17, 27, 13], semi-automated video annotation [31, 23, 1], and active classification for class-attribute models [4, 14]. 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 annotation 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. [21, 22] 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 complexityand 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.

Organization: The paper is organized as follows: In Section 2, we describe our basic model and algorithms for part-based recognition and show how user input can be incorporated in a realtime interface. In Section 3, we show how our interactive interface can be integrated with a scalable online learning algorithm for training deformable part models and discuss computational properties and bounds on annotation effort. In Section 4, we perform experiments to analyze how the level of automation evolves as more training examples are added.

2. Interactive Part Localization

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.

2.1. Model and Notation

Given an image x, let $\Theta = \theta_1 ... \theta_P$ 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 2(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:

$$s(\Theta; x) = \sum_{p \in V} \psi_p(\theta_p; x) + \sum_{(p,q) \in E} \lambda_{pq}(\theta_p, \theta_q) \qquad (1)$$

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 [28].

2.2. Incorporating User Input

Let $U_t = \{\tilde{\theta}_{j(1)} ... \tilde{\theta}_{j(t)}\}$ 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 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 $u_p(\theta_p; \tilde{\theta}_p)$:

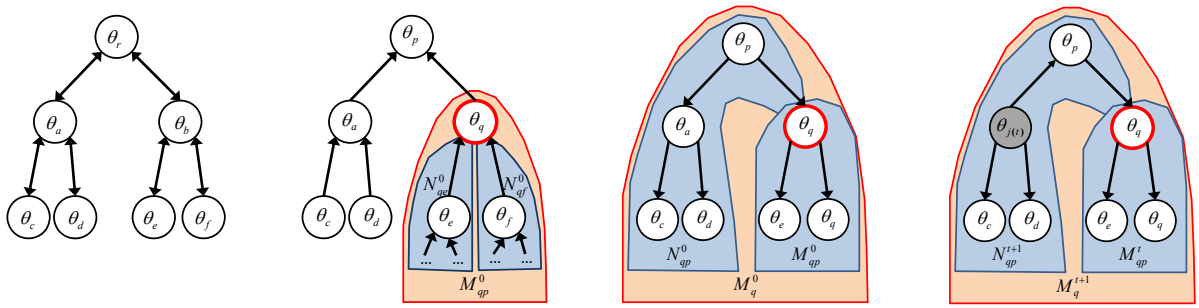
$$s^t(\Theta; x, U_t) = s(\Theta; x) + \sum_{\tilde{\theta}_p \in U_t} u_p(\theta_p; \tilde{\theta}_p) \qquad (2)$$

$$u_p(\theta_p; \tilde{\theta}_p) = \begin{cases} -\infty & \text{if } \theta_p \neq \tilde{\theta}_p \\ 0 & \text{otherwise} \end{cases} \qquad (3)$$

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).

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.



(a) Hierarchical Part Model (b) Bottom-Up Preprocessing (c) Top-Down Preprocessing (d) Propagate User Response

Figure 2. **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 6-7, c) In a second top-down algorithm, information is passed from the root back down to the children using Eq 8-10, 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 12-15.

Algorithm 1 INTERACTIVEPARTLABELER

Input: An image x and model weights $\mathbf{w}$
Output: Verified labels Θ_t^*

Compute part detection responses Ψ_p
Precompute solution for any possible user response:
  Bottom-up traversal, evaluating Eq 6-7
  Top-down traversal, evaluating Eq 8-10
while User unsatisfied with $\Theta_t^* = \max_{\theta_r} M_r^t[\theta_r]$ **do**
  As user drags $j(t)$, interactively show $M_{j(t)}^t[\tilde{\theta}_{j(t)}]$
  On mouse release, finalize solution $\tilde{\theta}_{j(t)}$:
    Update unary score $M_{j(t)}^{t+1}$ (Eq 11)
    Breadth first traversal from $j(t)$, evaluate Eq 12-15
  $t \leftarrow t+1$
end while

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 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 1.

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:

$$M_p^t[\tilde{\theta}_p] = \max_{\Theta} s^t(\Theta; x, U_t) \quad \text{s.t. } \theta_p = \tilde{\theta}_p \tag{4}$$

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 though 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 and q:

$$N_{pq}^t[\tilde{\theta}_p] = \max_{\theta_q} \left(M_{qp}^t[\theta_q] + \lambda_{pq}(\theta_p, \theta_q) \right) \tag{5}$$

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 $\max_{\theta_q}$; 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 [9], 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:

$$M_{qp}^0 = \Psi_q + \sum_{r \in \text{child}(q)} N_{qr}^0 \tag{6}$$

$$N_{pq}^0 = M_{qp}^0 \otimes \Lambda_{pq} \tag{7}$$

where Eq 6 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:

$$M^0_{pq} = M^0_p - N^0_{pq} \quad (8)$$
$$N^0_{qp} = M^0_{pq} \otimes \Lambda_{qp} \quad (9)$$
$$M^0_q = M^0_{qp} + N^0_{qp} \quad (10)$$

This top-down pass computes M^0_{pq}, N^0_{qp}, and M^0_q for all parent-child pairs p, q and relies on M^0_{qp} and N^0_{pq} being precomputed (these were computed during the initial dynamic progamming 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 $M^t_{j(t)}[\tilde{\theta}_{j(t)}]$. 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, which is used to update the ML solution for that part:

$$M^{t+1}_{j(t)}[\theta_{j(t)}] = M^t_{j(t)} + u_{j(t)}(\theta_{j(t)}; \tilde{\theta}_{j(t)}) \quad (11)$$

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 2(d):

$$M^{t+1}_{qp} = M^t_{qp}, \quad N^{t+1}_{pq} = N^t_{pq} \quad (12)$$
$$M^{t+1}_{pq} = M^{t+1}_p - N^{t+1}_{pq} \quad (13)$$
$$N^{t+1}_{qp} = M^{t+1}_{pq} \otimes \Lambda_{qp} \quad (14)$$
$$M^{t+1}_q = M^{t+1}_{qp} + N^{t+1}_{qp} \quad (15)$$

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.

3. 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 [20]), which searches for the optimal vector of weights $\mathbf{w}^*$ that minimizes the error function $F_n(\mathbf{w})$:

$$F_n(\mathbf{w}) = \frac{\lambda}{2}\|\mathbf{w}\|^2 + \frac{1}{n}\sum_{i=1}^{n} \ell_i(\mathbf{w}) \quad (16)$$
$$\ell_i(\mathbf{w}) = \max_y \left(\mathbf{w} \cdot \Phi(x_i, y) - \mathbf{w} \cdot \Phi(x_i, y_i) + \Delta(y_i, y)\right) \quad (17)$$

where $\{(x_1, y_1)...(x_n, y_n)\}$ 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 predicting part locations y when the true locations are y_i.

Algorithm 2 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:
$y_i \leftarrow$ INTERACTIVEPARTLABELER($\mathbf{w}_s$, x_i)
4: Update weights using Eq 20 or 22
5: $s \leftarrow s + 1$
6: Optional: w/ spare CPU cycles, repeat lines 4-5
7: **end for**

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:

$$\bar{y}_i = \max_y \left(\mathbf{w} \cdot \Phi(x_i, y) + \Delta(y_i, y)\right) \quad (18)$$
$$\nabla \ell_i = \Phi(x_i, \bar{y}_i) - \Phi(x_i, y_i) \quad (19)$$

Learning with strongly convex loss functions (this includes arbitrary convex loss function with L_2 regularization added such as Eq 16) has recently been extensively studied in online learning literature [12, 11, 18, 16]. [11, 12] show that for learning problems with λ-strongly convex loss functions (*e.g.*, the form of Eq 16), an online stochastic gradient descent (SGD) which streams in an example (x_s, y_s) and takes an update step

$$\mathbf{w}_s = \mathbf{w}_{s-1} - \frac{1}{\lambda s}\left(\mathbf{w}_{s-1} + \nabla \ell_s\right) \quad (20)$$

achieves at most logarithmic regret

$$\sum_{s=1}^{S} f_s(\mathbf{w}_s) - \min_{\mathbf{w}} \sum_{s=1}^{S} f_s(\mathbf{w}) \leq \frac{R^2\left(\log S + 1\right)}{2\lambda} \quad (21)$$

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 16 by at least as much as SGD [12]. A variant of SGD is used by Pegasos [18], a popular online learning algorithm for linear SVMs. A slightly better update

$$\mathbf{w}_s = \frac{s-1}{s}\mathbf{w}_{s-1} - \min\left(\frac{1}{\lambda s}, \frac{\ell_s(\mathbf{w}_{s-1})}{\|\nabla \ell_i\|^2}\right)\nabla \ell_i \quad (22)$$

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 [8], a fast optimizer for linear SVMs.

By choosing S such that regret bounds in Eq 21 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 19) 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 $\mathbf{w}_s$ to accelerate the labeling process (see Algorithm 2). 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 $\Delta(y_i, y)$ [20], 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 21) 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. 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 [24], 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 [24] is an extended version of the

Figure 3. **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)

Caltech-UCSD Birds 200 dataset [26] 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 processing 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.

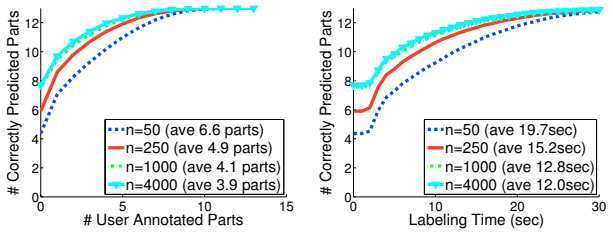
We processed training images in random order, ignoring bird species labels. The standard deviation of user click responses was computed separately for each part, using 5 different MTurk responses per image. Qualitatively, the simulated interface was fairly true to life (see Figure 3).

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. Each curve in Figure 4a 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 4b 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 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 online 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.



(a) Part Prediction By # Corrections (b) Part Prediction By Time

Figure 4. **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.

5. 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.

6. Acknowledgments

The authors thank Boris Babenko, Kristin Branson, and Peter Welinder for helpful discussions and feedback. Funding for this work was provided by NSF Grant AGS-0941760, ONR MURI Grant N00014-08-1-0638, ONR MURI Grant N00014-06-1-0734, and ONR MURI Grant 1015 G NA127.

References

[1] Y. Abramson and Y. Freund. Semi-automatic visual learning (seville). CVPR Tutorial, 2005.

[2] M. Blaschko and C. Lampert. Learning to localize objects with structured output regression. In *ECCV*, 2008.

[3] L. Bourdev and J. Malik. Poselets: Body part detectors trained using 3d human pose annotations. In *ICCV*, 2010.

[4] 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.

[5] C. Desai, D. Ramanan, and C. Fowlkes. Discriminative models for multi-class object layout. In *ICCV*, 2009.

[6] T. Dietterich, R. Lathrop, and T. Lozano-Pérez. Solving the multiple instance problem. *A.I.*, 1997.

[7] P. Dollár, B. Babenko, S. Belongie, P. Perona, and Z. Tu. Multiple component learning. In *ECCV*, 2008.

[8] R. Fan, K. Chang, C. Hsieh, X. Wang, and C. Lin. LIBLINEAR: library for large linear classification. *JMLR*, 2008.

[9] P. Felzenszwalb and D. Huttenlocher. Efficient matching of pictorial structures. In *CVPR*, 2002.

[10] P. Felzenszwalb, D. McAllester, and D. Ramanan. A discriminatively trained, multiscale, deformable part model. In *CVPR*, 2008.

[11] E. Hazan, A. Agarwal, and S. Kale. Logarithmic regret algorithms for online convex optimization. *M.L.*, 2007.

[12] S. Kakade and S. Shalev-Shwartz. Mind the duality gap: Logarithmic regret algorithms for online optimization. In *NIPS*, 2008.

[13] A. Levin, D. Lischinski, and Y. Weiss. A closed-form solution to natural image matting. *PAMI*, 2007.

[14] T. Mensink, J. Verbeek, and G. Csurka. Learning structured prediction models for interactive labeling. In *CVPR*, 2011.

[15] P. Ott and M. Everingham. Shared parts for deformable part-based models. In *CVPR*, 2011.

[16] N. Ratliff. (Online) Subgradient Methods for Structured Prediction. In *AIStats*, 2007.

[17] C. Rother, V. Kolmogorov, and A. Blake. Grabcut: Interactive foreground extraction using graph cuts. In *TOG*, 2004.

[18] S. Shalev-Shwartz, Y. Singer, and N. Srebro. Pegasos: Primal estimated sub-gradient solver for SVM. In *ICML*, 2007.

[19] Q. Shi, L. Wang, L. Cheng, and A. Smola. Discriminative human action segmentation and recognition using semi-markov model. In *CVPR*, 2008.

[20] I. Tsochantaridis, T. Joachims, T. Hofmann, and Y. Altun. Large margin methods for structured and interdependent output variables. *JMLR*, 2006.

[21] S. Vijayanarasimhan and K. Grauman. What's It Going to Cost You?: Predicting Effort vs. Informativeness for Multi-Label Image Annotations. In *CVPR*, 2009.

[22] S. Vijayanarasimhan and K. Grauman. Large-scale live active learning. In *CVPR*, 2011.

[23] C. Vondrick, D. Ramanan, and D. Patterson. Efficiently Scaling Up Video Annotation. In *ECCV*, 2010.

[24] C. Wah, S. Branson, P. Welinder, P. Perona, and S. Belongie. Caltech-UCSD Birds-200-2011. In *FGVC*, 2011.

[25] M. Weber, M. Welling, and P. Perona. Unsupervised learning of models for recognition. In *ECCV*, 2000.

[26] P. Welinder, S. Branson, T. Mita, C. Wah, F. Schroff, S. Belongie, and P. Perona. Caltech-UCSD Birds 200. Technical report, Caltech and UCSD, 2010.

[27] W. Wu and J. Yang. SmartLabel: an object labeling tool. In *ACM Multimedia*, 2006.

[28] Y. Yang and D. Ramanan. Articulated Pose Estimation using Flexible Mixtures of Parts. In *CVPR*, 2011.

[29] B. Yao, X. Yang, and S. Zhu. Introduction to a large-scale general purpose ground truth database. In *EMMCVPR*, 2007.

[30] C. Yu and T. Joachims. Learning structural SVMs with latent variables. In *ICML*, 2009.

[31] J. Yuen, B. Russell, C. Liu, and A. Torralba. Labelme video. In *ICCV*, 2010.