Large-Scale Live Active Learning:
Training Object Detectors with Crawled Data and Crowds

Sudheendra Vijayanarasimhan and Kristen Grauman
University of Texas at Austin
{svnaras,grauman}@cs.utexas.edu

Abstract

Active learning and crowdsourcing are promising ways to efficiently build up training sets for object recognition, but thus far techniques are tested in artificially controlled settings. Typically the vision researcher has already determined the dataset's scope, the labels "actively" obtained are in fact already known, and/or the crowd-sourced collection process is iteratively fine-tuned. We present an approach for live learning *of object detectors, in which the system autonomously refines its models by actively requesting crowd-sourced annotations on images crawled from the Web. To address the technical issues such a large-scale system entails, we introduce a novel part-based detector amenable to linear classifiers, and show how to identify its most uncertain instances in sub-linear time with a hashing-based solution. We demonstrate the approach with experiments of unprecedented scale and autonomy, and show it successfully improves the state-of-the-art for the most challenging objects in the PASCAL benchmark. In addition, we show our detector competes well with popular nonlinear classifiers that are much more expensive to train.*

1. Introduction

Object detection is a fundamental vision problem: given an image, which object categories are present, and where? Ongoing research is devoted to developing novel representations and classification algorithms in support of this task, and challenge datasets encourage further progress [1, 2, 3, 4, 5]. Today's best-performing detection methods employ discriminative learning together with window-based search, and assume that a large number of cleanly labeled training examples are available. For example, thousands of bounding box annotations per category is standard.

Given the substantial human effort required to gather good training sets—as well as the expectation that *more* data is almost always advantageous—researchers have begun to explore novel ways to collect labeled data. Both

active learning and *crowd-sourced labeling* are promising ways to efficiently build up training sets for object recognition. Active learning work shows how to minimize human effort by focusing label requests on those that appear most informative to the classifier [6, 7, 8, 9, 10], whereas crowd-sourcing work explores how to package annotation tasks such that they can be dispersed effectively online [11, 12, 13, 14, 15]. The interesting questions raised in these areas—such as dealing with noisy labels, measuring reliability, mixing strong and weak annotations—make it clear that data collection is no longer a mundane necessity, but a thriving research area in itself.

However, while ostensibly intended to distance algorithm developers from the data collection process, in practice existing techniques are tested in artificially controlled settings. Specifically, we see four limiting factors. First, previous work uses "sandbox" datasets, where the vision researcher has already determined the dataset's source and scope, meaning there is a fixed (and possibly biased) set of images that will even be considered for labeling. In fact, active learning methods have only been tested on sandbox data where the true labels are really known, and merely temporarily withheld from the selection algorithm. These common simulations likely inflate the performance of both active and passive learners, since anything chosen for labeling is relevant. Second, nearly all work targets the active *image classification* problem—not detection—and so images in the unlabeled pool are artificially assumed to contain only one prominent object. Third, most crowd-sourced collection processes require iterative fine-tuning by the algorithm designer (e.g., revising task requirements, pruning responses, barring unreliable Mechanical Turkers) before the data is in usable form. Fourth, the computational complexity of the active selection process is generally ignored, yet it is critical when running a live system to avoid keeping the human annotators idle. Thus, it is unknown to what extent current approaches could translate to real settings.

Our goal is to take crowd-sourced active annotation out of the "sandbox". We present an approach for *live learning* of object detectors, in which the system directly inter-

acts with human annotators online and iteratively poses annotation requests to refine its models. Rather than fill the data pool with some canned dataset, the system itself gathers possibly relevant images via keyword search (we use Flickr). It repeatedly surveys the data to identify unlabeled sub-windows that are most uncertain according to the current model, and generates tasks on MTurk to get the corresponding bounding box annotations. After an annotation budget is spent, we evaluate the resulting detectors both on benchmark data, as well as a novel test set from Flickr. Notably, throughout the procedure *we do not intervene with what goes into the system's data pool, nor the annotation quality from the hundreds of online annotators.*

To make the above a reality requires handling some important technical issues. Active selection for window-based detection is particularly challenging since the object extents (bounding boxes) are unknown in the unlabeled examples; naively one would need to evaluate all possible windows within the image in order to choose the most uncertain. This very quickly leads to a prohibitively large unlabeled pool to evaluate exhaustively. Thus, we introduce a novel part-based detector amenable to linear classifiers, and show how to identify its most uncertain instances in sub-linear time with a hashing-based solution we recently developed.

We show that our detector strikes a good balance between speed and accuracy, with results competitive with and even exceeding the state-of-the-art on the PASCAL VOC. Most importantly, we show successful live learning in an uncontrolled setting. The system learns accurate detectors with much less human effort than strong baselines that rely on human-verified keyword search results.

2. Related Work

Object detection has received various treatments in the literature; see [5] and references therein for an overview. Currently window-based approaches based on gradient features and subwindow parts provide state-of-the-art results using discriminative classifiers. A known limitation, however, is their fairly significant computational expense, due both to the need to search exhaustively through all windows in the image, as well as the classifiers' complexity (e.g., SVMs with nonlinear kernels or latent variables [3, 2]).

Various ways to reduce detection time have been explored, including cascades [3], branch-and-bound search [4], or jumping windows [16]. To reduce classifier training and testing costs, simpler linear models are appealing. While linear models tend to underperform with common representations (e.g., see tests in [3, 17]), recent work in image classification shows very good results when also incorporating sparse coding and feature pooling [18, 19, 17]. We propose a part-based object model that exploits a related representation, and show it to be competitive with state-of-the-art detection results. To our

knowledge, no previous work considers sparse coding and linear models for object detection.

Active learning has been shown to better focus annotation effort for image recognition tasks [6, 7, 9] and region labeling [8, 10]. However, no previous work uses active learning to train a window-based detector. To do so introduces major scalability issues, which we address with a new linear detector combined with a hashing algorithm [20] for sub-linear time search of the unlabeled pool. Further, all previous work tests active selection only in a sandbox.

Researchers have investigated issues in annotation tools and large-scale database collection for recognition. Keyword-based search is often used for dataset creation, and several recent efforts integrate crowd-sourced labeling [11, 12, 15] or online and incremental learning [21]. Even with a human in the loop, annotation precision varies when using Web interfaces and crowds, and so some research explores ways to automatically provide quality assurance [13, 14]. Other work attempts to directly learn object models from noisy keyword search (e.g., [22, 21]); however, such methods assume a single prominent object of interest per image, whereas for detection we will have cluttered candidate images that require a bounding box to identify the object of interest.

Overall, previous active learning methods focus on image classification, and/or demonstrate results under controlled settings on prepared datasets of modest scale. Ours is the first complete end-to-end approach for scalable, automatic online learning of object detectors.

3. Approach

Our goal is to enable online active crowd-sourced object detector training. Given the name of a class of interest, our system produces a detector to localize novel instances using automatically obtained images and annotations. To make this feasible, we first propose a part-based linear SVM detector, and then show how to identify its uncertain examples efficiently using a hashing scheme.

3.1. Object Representation and Linear Classifier

We first introduce our part-based object model. Our goal is to design the representation such that a simple linear classifier will be adequate for robust detection. A linear model has many complexity advantages important to our setting: i) SVM training requires time linear in the number of training examples, rather than cubic [23], ii) classification of novel instances requires constant time rather than growing linearly with the number of training examples, iii) exact incremental classifier updates are possible, which makes an iterative active learning loop practical, and iv) hash functions enable sub-linear time search to map a *query hyperplane* to its nearest points according to a linear kernel [20].
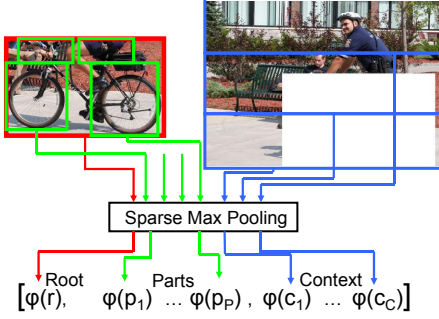
Figure 1. Our part-based object representation.

Inspired by recent findings in sparse coding for image classification [18, 19, 17], we explore a detection model based on sparse coding of local features combined with a max pooling scheme. Previous representations pool coded SIFT features in a single global window or in a fixed class-independent hierarchy of grid cells (i.e., a spatial pyramid structure). While sufficient for whole-image classification, we instead aim to represent an *object* separately from its context, and to exploit its part-based structure with *class-dependent* subwindow pooling.

To this end, we propose an object model consisting of a root window r, multiple part windows $\{p_1, \ldots, p_P\}$ that overlap the root, and context windows $\{c_1, \ldots, c_C\}$ surrounding it. See Figure 1. Let $O = [r, p_1, \ldots, p_P, c_1, \ldots, c_C]$ denote a candidate object configuration within an image, and let $\phi(W)$ denote the sparse feature encoding for local image descriptors extracted from a given subwindow W (to be defined below). The detector scores a candidate configuration as a simple linear sum:

$$f(O) = \boldsymbol{w}^T \phi(O) \tag{1}$$
$$= \boldsymbol{w}_r \phi(r) + \sum_{i=1}^{P} \boldsymbol{w}_{p_i} \phi(p_i) + \sum_{i=1}^{C} \boldsymbol{w}_{c_i} \phi(c_i),$$

where $\boldsymbol{w}$ denotes the learned classifier weights, which we obtain with SVM training. We next flesh out the window descriptions; Sec. 3.2 explains how we obtain candidate root placements.

Window descriptions Given a novel test image, we first extract local image descriptors; we use a dense multi-scale sampling of SIFT in our implementation. Each window type (r, p_i, or c_j) uses these features to create its encoding $\phi(\cdot)$. The *root window* provides a global summary of the object appearance, and is invariant to spatial translations of features within it.

Similarly, each *part window* summarizes the local features within it, discarding their positions; however, the location of each part is defined relative to the current root, and depends on the object class under consideration (i.e., bicycles and cars each have a different configuration of the

p_i windows). Thus, they capture the locations and spatial configurations of the most important parts of the object. We train with the part locations and bounds learned by the detector in [2] on an initial labeled set; alternatively, they could be requested once directly from annotators.

The *context windows* incorporate contextual cues surrounding the object, such as the presence of "sky", "ground", "road", etc., and also help discard poorer candidate windows that cover only parts of objects (in which case object features spill into the context window). We create the context windows using a 3×1 partition of r's complement, as shown in the top right of Figure 1.

Feature encoding Each window is represented using a nonlinear feature encoding based on sparse coding and max-pooling, which we refer to as Sparse Max Pooling (SMP). The SMP representation is related to the well-known bag-of-features (BoF); however, unlike BoF, each component local descriptor is first encoded as a combination of *multiple* visual words, and the weights are then pooled within a region of interest using the max function.

Offline, we cluster a large corpus of randomly selected features to obtain a dictionary of $|V|$ visual words: $\boldsymbol{V} = [\boldsymbol{v}_1, \ldots, \boldsymbol{v}_{|V|}]$, where each column $\boldsymbol{v}_i \in \Re^{128}$ is a cluster center in SIFT space. For any window W (whether root/part/context), let $F = \{\boldsymbol{f}_i\}_{i=1}^{|F|}$ be the set of local features falling within it, where each $\boldsymbol{f}_i \in \Re^{128}$ is a SIFT descriptor. We represent this window with a sparse $|V|$-dimensional vector, as follows.

First, each feature $\boldsymbol{f}_i$ is quantized into a $|V|$-dimensional sparse vector $\boldsymbol{s}_i$ that approximates $\boldsymbol{f}_i$ using some existing sparse coding algorithm and the dictionary $\boldsymbol{V}$, that is, $\boldsymbol{f}_i \approx \boldsymbol{s}_i \boldsymbol{V}$. Taking this encoding for every $\boldsymbol{f}_i$ as input, the SMP representation of W is given by:

$$\phi(W) = [\phi^1, \ldots, \phi^{|V|}], \text{where} \tag{2}$$
$$\phi^j = \max(\boldsymbol{s}_i(j)), i = 1, \ldots, |F|,$$

for $j = 1, \ldots, |V|$, and $\boldsymbol{s}_i(j)$ is the j-th dimension of the sparse vector encoding the i-th original feature, $\boldsymbol{f}_i$. Finally, we normalize $\phi(W)$ by its L_2 norm.[1]

The rationale behind the SMP window encoding is twofold: the sparse coding gives a fuller representation of the original features by reflecting nearness to multiple dictionary elements (as opposed to BoF's usual hard vector quantization), while the max pooling gives better discriminability amidst high-variance clutter [17]. See [17, 18] for useful comparisons between various sparse coding approaches, which shows their clear advantage when combined with a linear kernel as compared to the popular BoF.

Relationship to existing detection models Our model intentionally strikes a balance between two recent state-of-

---

[1] We use Locality-constrained Linear Coding (LLC) [19] to obtain the sparse coding, though other algorithms could also be used for this step.

(a) Spatial pyramid model (SP) (b) Latent deformable part model (LSVM) (c) Proposed model
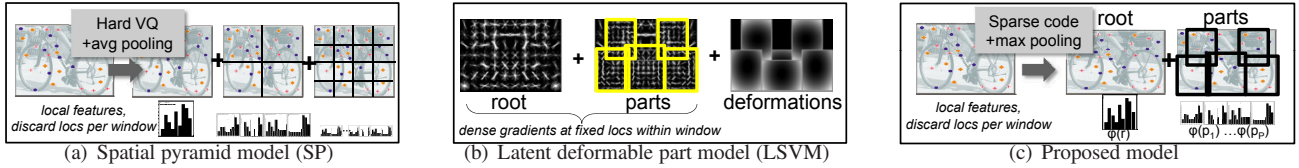
Figure 2. Sketch to illustrate contrasts with related existing models. See text for details.

the-art detection models: i) a nonlinear SVM with a spatial pyramid (SP) in which each grid cell is a histogram of unordered visual words [3], and ii) a latent SVM (LSVM) with a root+deformable part model in which each part is a rigid histogram of ordered oriented gradients [2]. See Figure 2.

On the one hand, the SP model is robust to spatial translations of local features within each grid cell. On the other hand, its nonlinear kernels (required for good performance [3]) makes the classifier quite expensive to train and test, and rigid class-independent bins may fail to capture the structure of the best parts on an object (see Fig. 2(a)). In contrast, the LSVM model can robustly capture the parts, since it learns multiple part filters that deform relative to the root. However, its dynamic programming step to compute parts' alignment makes it expensive to train. Furthermore, its use of the spatially dense gradient histograms for both the root and parts make them less tolerant to *internal* shifts and rotations (see Fig. 2(b)).

Our model attempts to incorporate positive aspects of the above two models, while maintaining a much lower computational cost. In particular, we have class-specific part configurations, like [2], but they are fixed relative to the root, like [3]. Our SMP-based encoding is robust to shifts within the part and object windows, thereby tolerating some deformation to the exact part placement without needing the additional DP alignment step during detection. In short, by utilizing a part-based representation and a linear classifier, our approach provides a very good trade-off in terms of model complexity and accuracy.

3.2. Generating Candidate Root Windows

So far we have defined a representation and scoring function for any candidate window. Now we discuss how to generate the candidates, whether in novel test images or unlabeled images the system is considering for annotation. A thorough but prohibitively expensive method would be the standard *sliding window* approach; instead, we use a grid-based variant of the *jumping window* method of [16, 24].

The jumping window approach generates candidate windows with a Hough-like projection using visual word matches, and prioritizes these candidates according to a measure of how discriminative a given word and coarse location is for the object class (see Figure 3). First, each root window in the training images is divided into an $N \times M$ grid. Let $W_{loc}(r)$ denote a root window's position and
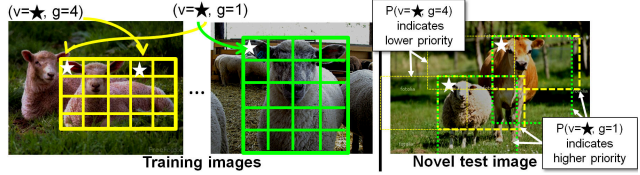

Figure 3. Illustration of jumping window root candidates. Grid cells serve to refine the priority given to each box (but do not affect its placement). Here, location $g = 1$ has higher priority than $g = 4$ for visual word $v = \star$ since it appears more consistently in the training images (left two images).

scale. Given a training window r and a visual word v occurring at grid position $g \in \{1, \ldots, NM\}$, we record the triplet $(v, g, W_{loc}(r))$. We build a lookup table indexed by the v entries for all training examples. Then, given a test image, for each occurrence of a visual word v, we use the lookup table to retrieve all possible $W_{loc}(r)$'s, and project a bounding box in the test image relative to that v's position. Note, candidates can vary in aspect ratio and scale.

The grid cell g in each triple is used to assign a priority score to each candidate, since we may not want to examine all possible candidates mapped from the lookup table. Specifically, we score a given pair (v, g) based on how predictive it is for the true object bounding box across the training set: $P(v, g)$ is the fraction of the occurrences of word v that appear at grid location g. This function gives a higher score to bounding boxes where the visual word occurs consistently across positive training examples at a particular position (see Figure 3).

Given a test image, we take the top K candidate jumping windows based on their priority scores. The detector is run only on these boxes. In experiments, we obtain 95% recall on most categories when taking just $K = 3,000$ candidates per test image. The same level of recall would require up to 10^5 bounding boxes if using sliding windows (see [3]).

3.3. Active Selection of Object Windows

We initialize our online active learning system with a linear SVM trained with a small number of labeled examples for the object. Then, it crawls for a pool of potentially relevant unlabeled data using keyword search with the object name (i.e., it downloads a set of images tagged 'dog' when learning to detect dogs). We want to efficiently determine which images among those retrieved should be labeled next by the human annotators. As an active learning criterion, we use the "simple margin" selection method for

SVMs [25], a widely used criterion that seeks points that most reduce the version space. Given an SVM with hyperplane normal w and an unlabeled pool of data $\mathcal{U}_O = \{\phi(O_1), \dots, \phi(O_n)\}$, the point that minimizes the distance to the current decision boundary is selected for labeling: $O^* = \arg\min_{O_i \in \mathcal{U}_O} |w^T \phi(O_i)|$.

A naive application of this criterion entails computing the classifier response on all unlabeled data, ranking them by $|w^T \phi(O_i)|$. However, even with a linear classifier, exhaustively evaluating all unlabeled examples at each iteration is prohibitively expensive. Whereas previous active learning work is generally unconcerned about the amount of time it actually takes to compute the next labeling request, it becomes a real issue when working out of the sandbox, since we have live annotators awaiting the next labeling jobs and massive unlabeled data pools. In particular, since we need to apply the active selection function at the level of the *object*, not the entire *image*, we have an enormous number of instances—all bounding boxes within the unlabeled image data. Even using jumping windows, thousands of images yield millions of candidates. Thus, a simple linear scan of all unlabeled data is infeasible.

Therefore, we adopt our *hyperplane-hashing* algorithm [20] to identify the most promising candidate windows in sub-linear time. The algorithm maps inputs to binary keys using a randomized hash function that is locality-sensitive for the angle between the hyperplane normal and a database point. Given a "query hyperplane", one can hash directly to those points that are nearest to the hyperplane, with high probability.

Formally, let $\mathcal{U}_I$ denote the set of unlabeled images, and $\mathcal{U}_O$ denote the pool of candidate object windows obtained using the jumping window predictor on $\mathcal{U}_I$. Note that $|\mathcal{U}_O| = K \times |\mathcal{U}_I|$. The locality-sensitive hash family $\mathcal{H}$ generates randomized functions with two-bit outputs:

$$h_{\mathcal{H}}(z) = \begin{cases} h_{u,v}(\phi(O_i), \phi(O_i)), & \text{if } z \text{ is a database vector,} \\ h_{u,v}(w, -w), & \text{if } z \text{ is a query hyperplane,} \end{cases}$$

where the component function is defined as

$$h_{u,v}(a, b) = [\text{sign}(u^T a), \text{sign}(v^T b)], \quad (3)$$

$\text{sign}(u^T a)$ returns 1 if $u^T a \geq 0$, and 0 otherwise, and u and v are sampled from a standard multivariate Gaussian, $u, v \sim \mathcal{N}(0, I)$. These functions guarantee high probability of collision for a query hyperplane and the points nearest to its boundary. The two-bit hash limits the retrieved points' deviation from the perpendicular by constraining the angle with respect to both w and $-w$. See [20] for details.

We use these functions to hash the crawled data into the table.[2] Then, at each iteration of the active learning loop, we

---

[2]Hyperplane hashes can be used with existing approximate nearest-neighbor search algorithms; we use Charikar's formulation, which guarantees the probability with which the nearest neighbor will be returned.

Figure 4. MTurk interface to obtain bboxes on actively selected examples.

hash the current classifier as a query, and directly retrieve examples closest to its decision boundary. We search only those examples, i.e., we compute $|w^T \phi(O_i)| = |f(O_i)|$ for each one, and rank them in order of increasing value. Finally, the system issues a label request for the top T images under this ranking. Since we only need to evaluate the classifier for examples that fall into a particular hash bucket—typically less than 0.1% of the total number of unlabeled examples—this strategy combined with our new detector makes online selection from large datasets feasible.

3.4. Online Annotation Requests

To automatically obtain annotations on the actively selected examples, our system posts jobs on Mechanical Turk, where it can pay workers to provide labels. The system gathers the images containing the most uncertain bounding boxes, and the annotators are instructed to use a rectangle-drawing tool to outline the object of interest with a bounding box (or else to report that none is present). We ask annotators to further subdivide instances into "normal", "truncated", or "unusual", consistent with PASCAL annotations, and to flag images containing more than 3 instances. Figure 4 shows the annotation interface.

While MTurk provides easy access to a large number of annotators, the quality of their labels varies. Thus, we design a simple but effective approach to account for the variability. We issue each request to 10 unique annotators, and then cluster their bounding boxes using mean shift to obtain a consensus. We keep only those clusters with boxes from more than half of the annotators. Finally, we obtain a single representative box from each cluster by selecting the one with the largest mean overlap with the rest.

Note how each image consists of thousands of unlabeled window instances, each of which serves as a candidate query; once a single image annotation is obtained, however, it tells us the labels for all windows within it.

3.5. Training the Detector

Training our detector entails learning the linear SVM weights in Eqn. 2 to distinguish windows that contain the object of interest from all others. To limit the number of negative windows used to train, we mine for "hard" negatives: at each iteration, we apply the updated classifier to the newly labeled images, and add the 10 top-scoring windows as negatives if they overlap the target class by $< 20\%$.

	classif	parts	feats	cands	aero.	bicyc.	bird	boat	bottl	bus	car	cat	chair	cow	dinin.	dog	horse	motor.	person	potte.	sheep	sofa	train	tvmon.	Mean
Ours	linear	yes	single	jump	**48.4**	48.3	14.1	13.6	15.3	43.9	49.0	**30.7**	11.6	30.3	13.3	**21.8**	43.6	45.0	18.2	11.1	**28.8**	**33.0**	47.7	43.0	30.5
BoF SP	linear	no	single	jump	30.4	43.1	6.9	3.5	10.8	35.8	45.0	17.7	11.5	24.6	3.5	18.0	43.5	44.0	15.3	1.5	19.1	14.7	35.7	34.9	23.0
LLC SP	linear	no	single	jump	35.9	46.7	6.4	6.3	16.5	45.6	49.8	26.7	12.5	27.3	6.8	18.2	44.9	45.0	18.2	4.6	23.2	22.6	41.3	42.0	27.0
LSVM+HOG [2]	nonlinear	yes	single	slide	32.8	**56.8**	2.5	**16.8**	**28.5**	39.7	**51.6**	21.3	**17.9**	18.5	**25.9**	8.8	49.2	41.2	**36.8**	**14.6**	16.2	24.4	39.2	39.1	29.1
SP+MKL [3]	nonlinear	no	multiple	jump	37.6	47.8	**15.3**	15.3	21.9	**50.7**	50.6	30.0	17.3	**33.0**	22.5	21.5	**51.2**	**45.5**	23.3	12.4	23.9	28.5	45.3	**48.5**	**32.1**

Table 1. Average precision compared to a spatial pyramid BoF baseline (BoF SP), a sparse coding max pooling spatial pyramid baseline modeled after [19] (LLC SP), and two state-of-the-art approaches [2, 3] on the PASCAL VOC, where all methods are trained and tested on the standard benchmark splits.
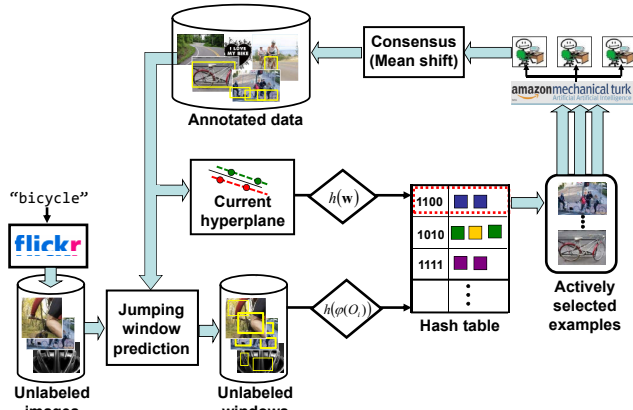

Figure 5. Summary of our system for live learning of object detectors.

We can now actively train an object detector automatically using minimal crowd-sourced human effort. To recap, the main loop consists of using the current classifier to generate candidate jumping windows, storing all candidates in a hash table, querying the hash table using the hyperplane classifier, giving the actively selected examples to online annotators, taking their responses as new ground truth labeled data, and updating the classifier. See Figure 5.

4. Results

The goal of our experiments is three-fold. First, we compare the proposed detector to the most closely related state-of-the-art techniques. Second, we validate our large-scale active selection approach with benchmark data. Third, we deploy our complete live learning system with crawled images, and compare to strong baselines that request labels for the keyword search images in a random sequence. We use two datasets: the PASCAL VOC 2007, and a new Flickr dataset (details below).

Implementation details We use dense SIFT at three scales (16, 24, 32 pixels) with grid spacing of 4 pixels, for 30K features per image. We obtain $|V| = 56,894$ visual words with two levels of hierarchical k-means on a sample of training images. We use the fast linear SVM code `svm_perf` [23], $C = 100$. We use the LLC code [19], and set k, the number of non-zero values in the sparse vector s_i to 5, following [19]. We use $P = 6$ parts per object from each of a 2-mixture detector from [2] trained on PASCAL data, take $T = 100$ instances per active cycle,

set $K = 3000$, and $N, M = 4$. We fix $N^\rho = 500$ and $\epsilon' = 0.01$ for the hash table [20]. During detection we run non-max suppression on top ranked boxes and select 10 per image. We score all results with standard PASCAL metrics and train/test splits.

4.1. Comparison to State-of-the-Art Detectors

First we compare our detector to the algorithms with the current best performance on VOC 2007 benchmark of 20 objects, as well as our own implementation of two other relevant baselines.

Table 1 shows the results. The first three rows all use the same original SIFT features, a linear SVM classifier, and the same jumping windows in the test images. They differ, however, in the feature coding and pooling. The **BoF SP** baseline maps the local features to a standard 3-level spatial pyramid bag-of-words descriptor with L_2-normalization. The **LLC SP** baseline applies sparse coding and max pooling within the spatial pyramid cells. LLC SP is the method of [19]; note, however, we are applying it for detection, whereas the authors propose their approach for image classification.

The linear classifier with standard BoF coding is the weakest. The LLC SP baseline performs quite well in comparison, but its restriction to a global spatial pyramid structure appears to hinder accuracy. In contrast, our detector improves over LLC SP noticeably for most objects (compare rows 1 and 3), likely due to its part windows.

Our detector is competitive with both of the state-of-the-art approaches discussed in Section 3.1: **SP+MKL** [3], which uses a cascade of classifiers that culminates with a learned combination of nonlinear SVM kernels over multiple feature types, and **LSVM+HOG** [2], which uses the latent SVM and deformation models for parts. In fact, our detector outperforms **all existing results** for 6 of the 20 objects, improving the state-of-the-art. At the same time, it is significantly faster to train (about 50 to 600 times faster; see Table 4).

The classes where we see most improvements seem to make sense, too: our approach outperforms the rigid spatial pyramid representation used in [3] for cases with more class-specific part structure (aeroplane, bicycle, train), while it outperforms the dense gradient parts used in [2] for the more deformable objects (dog, cat, cow).
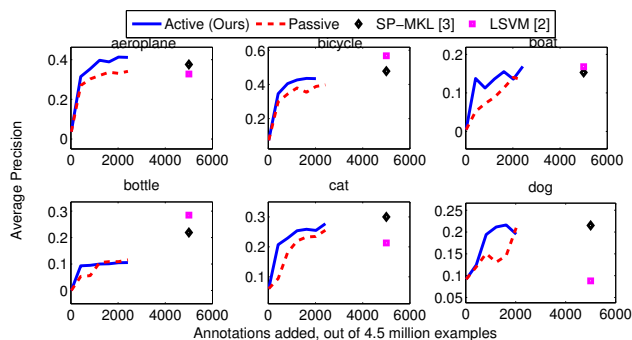
Figure 6. Active detector training on PASCAL. Our large-scale active selection yields steeper learning curves than passive selection, and reaches peak state-of-the-art performance using only ∼30% of the data.

	aeroplane	bird	boat	cat	dog	sheep	sofa	train
Ours	**48.4**	**15.8**[*]	**18.9**[*]	**30.7**	**25.3**[*]	**28.8**	**33.0**	**47.7**
Previous best	37.6	15.3	16.8	30.0	21.5	23.9	28.5	45.3

Table 2. Categories for which our method yields the best AP on PASCAL VOC 2007, compared to any result we found in the literature. ([*] means extra Flickr data automatically obtained by our system was used to train.)

4.2. Active Detector Training on PASCAL

We next compare our active selection scheme to a passive learning baseline that randomly selects images for bounding box annotation. We select six representative categories from PASCAL: we take two each from those that are "easier" (>40 AP), "medium" (25-40 AP) and "hard" (0-25 AP) according to the state-of-the-art result (max of rows 4 and 5 in Table 1). We initialize each object's classifier with 20 examples, and then let the remainder of the training data serve as the unlabeled pool, a total of 4.5 million examples. At each iteration, both methods select 100 examples, add their true bounding boxes (if any) to the labeled data, and retrain. This qualifies as learning in the "sandbox", but is useful to test our jumping window and hashing-based approach. Furthermore, the natural cluttered images are significantly more challenging than data considered by prior active object learning approaches, and our unlabeled pool is orders of magnitude larger.

Figure 6 shows the results. We see our method's clear advantage; the steeper learning curves indicate it improves accuracy on the test set using fewer labels. In fact, in most cases our approach reaches state-of-the-art performance (see markers above 5000 labels) using only one-third of the available training data.

4.3. Online Live Learning on Flickr

Finally, we deploy our complete live learning system, where new training data is crawled on Flickr. We consider all object classes for which state-of-the-art AP is less than 25.0 (boat, dog, bird, pottedplant, sheep, chair) in order to provide the most challenging case study, and to seek improvement through live learning where other methods have struggled most. To form the Flickr test set, we download
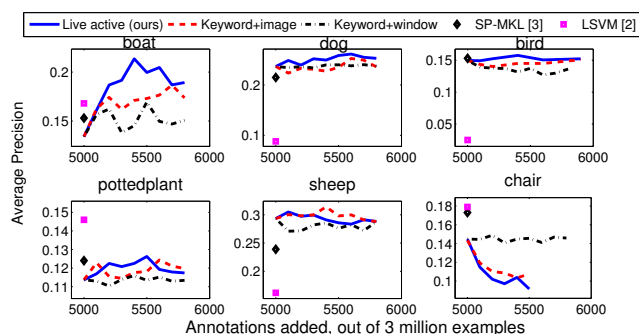

Figure 7. Live learning results on PASCAL test set.

	bird	boat	chair	dog	pottedplant	sheep
Flickr-crawled	2936	3138	2764	1831	1566	1570
Flickr-test	655	628	419	780	364	820

Table 3. Number of images in the crawled data and the new Flickr test set.

images tagged with the class names dated in 2010; when running live training, our system is restricted to images dated in 2009. See Table 3 for the data stats.

We compare to (1) a **Keyword+image baseline** that uses the same crawled image pool, but randomly selects images to get annotated on MTurk, and (2) a **Keyword+window baseline** that randomly picks jumping windows to get labeled. These are strong baselines since most of the images will contain the relevant object. In fact, *they exactly represent the status quo approach*, where one creates a dataset by manually pruning keyword search results. We initialize all methods with the PASCAL-trained models (5000 training images), and run for 10 iterations.

Live learning applied to PASCAL test set Figure 7 shows the results. For four of the six categories, our system improves test accuracy, and outperforms the keyword approaches. The final AP also exceeds the current state-of-the-art for three categories (see Table 2, comparing to best of [2, 3]). This is an exciting result, given the size of the unlabeled pools (∼3 million examples), and the fact that the system refined its models completely automatically.

However, for two classes (chair, sheep), live learning decreases accuracy. Of course, more data cannot guarantee improved performance on a fixed test set. We suspect the decline is due to stark differences in the distribution of PASCAL and Flickr images, since the PASCAL dataset creators do some manual preparation and pruning of all PASCAL data. Our next result seems to confirm this.

Live learning applied to Flickr test set Figure 8 shows the results on the new Flickr test set, where we apply the same live-learned models from above. While this test set appears more challenging than PASCAL, the improvements made by our approach are dramatic—both in terms of its absolute climb, as well as its margin over the baselines. In all, the results indicate that our large-scale live learning approach can autonomously build models appropriate
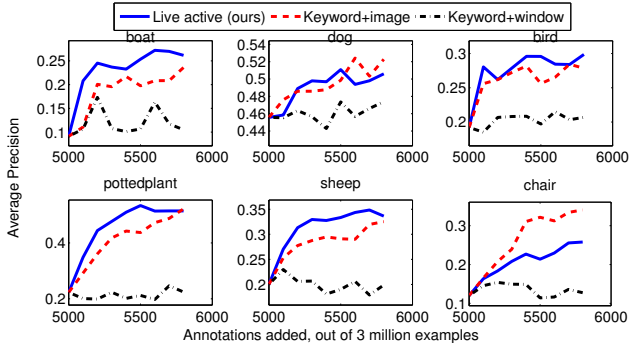
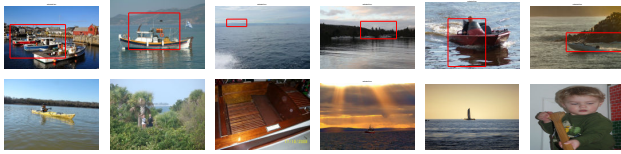Figure 8. Live learning results on Flickr test set.

Figure 9. Selections by our live approach (top), Keyword+image (bottom).

	Active selection	Training	Detection per image
Ours + active	10 mins	5 mins	150 secs
Ours + passive	0 mins	5 mins	150 secs
LSVM [2]	3 hours	4 hours	2 secs
SP+MKL[3]	93 hours	> 2 days	67 secs

Table 4. Run-time comparisons. Our detection time is mostly spent pooling the sparse codes. Active times are estimated for [2, 3] models based on linear scan. Our approach's efficiency makes live learning practical.

for detection tasks with realistic and unbiased data. Figure 9 shows selections made by either method when learning "boat", illustrating how ours focuses human attention among the crawled tagged images.

4.4. Computation Time

Table 4 shows the time complexity of each stage, and illustrates our major advantages for selection and retraining compared to existing strong detectors. Our times are based on a dual-core 2.8 GHz CPU, comparable to [2, 3]. Our jumping window+hashing scheme requires on average 2-3 seconds to retrieve 2,000 examples nearest the current hyperplane, and an additional 250 seconds to rank and select 100 images to query. In contrast, a linear scan over the entire unlabeled pool would require about 60 hours.

The entire online learning process requires 45-75 minutes per iteration: 5-10 min. to retrain, 5 min. for selection, and ~1 hour to wait for the MTurk annotations to come back (typically 50 unique workers gave labels per task). Thus, waiting on MTurk responses takes the majority of the time, and could likely be reduced with better payment. In comparison, the same selection with [2, 3] would require about 8 hours to 1 week, respectively.

5. Conclusions

Our contributions are i) a novel efficient part-based linear detector that provides excellent performance, ii) a jumping window and hashing scheme suitable for the proposed detector that retrieves relevant instances among millions of candidates, and iii) the first active learning results for which both data and annotations are automatically obtained, with minimal involvement from vision experts. Tying it all together, we demonstrated an effective end-to-end system on two challenging datasets.

References

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

[2] P. Felzenszwalb, R. Girshick, D. McAllester, and D. Ramanan. Object Detection with Discriminatively Trained Part Based Models. *TPAMI*, 99(1), 2009.

[3] A. Vedaldi, V. Gulshan, M. Varma, and A. Zisserman. Multiple Kernels for Object Detection. In *ICCV*, 2009.

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

[5] M. Everingham, L. Van Gool, C. Williams, J. Winn, and A. Zisserman. The Pascal Visual Object Classes Challenge. *IJCV*, 88(2):303–338, June 2010.

[6] A. Kapoor, K. Grauman, R. Urtasun, and T. Darrell. Active Learning with Gaussian Processes for Object Categorization. In *ICCV*, 2007.

[7] G. Qi, X. Hua, Y. Rui, J. Tang, and H. Zhang. Two-Dimensional Active Learning for Image Classification. In *CVPR*, 2008.

[8] S. Vijayanarasimhan and K. Grauman. Multi-Level Active Prediction of Useful Image Annotations for Recognition. In *NIPS*, 2008.

[9] A. Joshi, F. Porikli, and N. Papanikolopoulos. Multi-Class Active Learning for Image Classification. In *CVPR*, 2009.

[10] B. Siddiquie and A. Gupta. Beyond Active Noun Tagging: Modeling Context for Multi-Class Active Learning. In *CVPR*, 2010.

[11] L. von Ahn and L. Dabbish. Labeling Images with a Computer Game. In *CHI*, 2004.

[12] B. Russell, A. Torralba, K. Murphy, and W. Freeman. Labelme: a Database and Web-Based Tool for Image Annotation. *IJCV*, 2007.

[13] A. Sorokin and D. Forsyth. Utility Data Annotation with Amazon Mechanical Turk. In *Wkshp on Internet Vision*, 2008.

[14] P. Welinder and P. Perona. Online Crowdsourcing: Rating Annotators and Obtaining Cost-Effective Labels. In *ACVHL*, 2010.

[15] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A Large-Scale Hierarchical Image Database. In *CVPR*, 2009.

[16] O. Chum and A. Zisserman. An Exemplar Model for Learning Object Classes. In *CVPR*, 2007.

[17] Y.-L. Boureau, F. Bach, Y. LeCun, and J. Ponce. Learning Mid-level Features for Recognition. In *CVPR*, 2010.

[18] J. Yang, K. Yu, Y. Gong, and T. Huang. Linear Spatial Pyramid Matching Sparse Coding for Image Classification. In *CVPR*, 2009.

[19] J. Wang, J. Yang, K. Yu, F. Lv, T. Huang, and Y. Gong. Locality-Constrained Linear Coding for Image Classification. In *CVPR*, 2010.

[20] P. Jain, S. Vijayanarasimhan, and K. Grauman. Hashing Hyperplane Queries to Near Points with Applications to Large-Scale Active Learning. In *NIPS*, 2010.

[21] L. Li, G. Wang, and L. Fei-Fei. Optimol: Automatic Online Picture Collection via Incremental Model Learning. In *CVPR*, 2007.

[22] R. Fergus, L. Fei-Fei, P. Perona, and A. Zisserman. Learning Object Categories from Google's Image Search. In *ICCV*, 2005.

[23] T. Joachims. Training Linear SVMs in Linear Time. In *KDD*, 2006.

[24] S. Vijayanarasimhan and A. Kapoor. Visual Recognition and Detection Under Bounded Computational Resources. In *CVPR*, 2010.

[25] S. Tong and D. Koller. Support Vector Machine Active Learning with Applications to Text Classification. In *ICML*, 2000.