Timely Object Recognition

Anonymous ECCV submission

Paper ID 1617

Abstract. In a large visual multi-class detection system, the problem of timeliness of results is crucial. We are motivated by situations where running all detectors would take an unacceptably long time, and that the best answer must be given by some deadline. Our system for multi-class detection aims to give the best possible results at any single point after a start time; it is terminated at a deadline time. Toward this goal, we formulate a dynamic policy, guided by inference of the contents of the image, to decide which detector to deploy next. We evaluate parametrizations of the policy with respect to performance in the novel AP vs. Time evaluation on the PASCAL VOC dataset.

1 Introduction

In recent years, the computer vision field has converged on a general method for object detection, and a standard evaluation of results. Most current state-of-the-art object detection systems consist of three tasks: proposing regions of the image, evaluating a given region for presence of an object of a given category, and post-processing the results. In the evaluation ground truth, each object is most commonly assumed to belong to one of a fixed set of classes; its location is approximated by placing a bounding box around pixels belonging to it. Large datasets of such human annotations are used for evaluation of detection algorithms [1,2].

With few exceptions, current detection systems are not inherently multi-class. Instead, separate detectors are trained per class, and they are deployed and evaluated independently. The evaluation of multi-class performance, if at all given, consists of averaging the per-class metrics. Some notable papers do make multi-class detection a priority, and accordingly evaluate in a multi-class setting, where false positives can be generated both by incorrect localization and incorrect labeling. State-of-the-art detection systems also do not generally make efficiency a goal.

It is of course important to refrain from locking object recognition research into a single detector architecture with a focus only on increasing efficiency. That said, there are applications for which performance truly is time-sensitive. In robotics, a small finite amount of processing power per unit time is all that is available for robust object detection if the robot is to usefully interact with humans. In large-scale detection system deployments, such as for image search, results need to be obtained quickly per image as the number of images to process is large and growing. When processing large photo collection on end-user machines for immediate consumer navigation, the same is true.

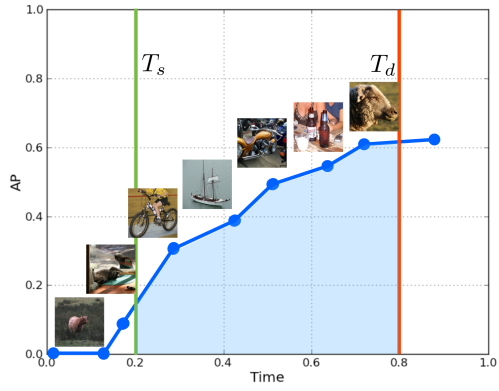


Fig. 1: We aim for *Anytime* performance within the bounds of the curve. That is, the policy should give the best possible answer at any time from start time T_s to deadline T_d.

In all these cases, an acceptable answer at a reasonable time may be more valuable than the best answer given too late. Furthermore, the value of the answer depends largely on the target application.

A hypothetical recognition system for a vision-based advertising deployment presents a case study. The system will have different accuracies for objects of different classes; detections will have different values based on confidence and class; and the queue of unprocessed images will vary in size. The most rational detection strategy in such an environment should depend on all of these variables.

We argue that the key to tackling such problems of dynamic recognition resource allocation is to start asking a new question: *What is the best performance we can get on a budget?* To answer it, we consider the evaluation metric of performance vs. time, presented in Figure 1 and discussed further in the text.

Our goal is a dynamic policy for selecting classifiers or detectors to achieve the highest recognition performance under this evaluation.

2 Recognition Problems

We deal with a dataset of images $\mathcal{D}$, where each image $\mathcal{I}$ contains at least one, and often multiple, objects. Each object is labeled with exactly one category label $k \in \{1, \ldots, K\}$.

The multi-class, multi-label **classification** problem asks whether $\mathcal{I}$ contains at least one object of class k. The answer for a single label is given with a real-valued confidence by a function $\mathit{classify}(\mathcal{I}, k)$. We write the ground truth for an image as $\mathbf{C} = \{C_1, \ldots, C_K\}$, where $C_k \in \mathbb{B} = \{0, 1\}$ is set to 1 if an object of class k is present.

The answer is evaluated by plotting precision vs. recall across dataset $\mathcal{D}$ (by progressively lowering the confidence threshold for a positive label) and

integrating to yield the Average Precision (AP) metric, which has become the standard evaluation for recognition performance on challenging datasets [1].

The **detection** problem is to output a list of bounding boxes (sub-images defined by four coordinates), each with a real-valued confidence that it encloses a single instance of an object of class k, for each k. The answer for a single class is given by an algorithm $detect(\mathcal{I}, k)$, which outputs a list of sub-image bounding boxes B and their associated confidences.

A common measure of a correct detection is the PASCAL overlap: two bounding boxes are considered to match if they have the same label and the ratio of their intersection to their union is at least $\frac{1}{2}$. Again, Average Precision is the single-number metric for the performance of a detector.

As our task is fundamentally in *multi-class* object detection, we rely on a slightly different evaluation than is commonly used (although it has precedent in [3]). Instead of pooling detections across images in the dataset, and considering classes individually, we pool detections across classes, but consider images individually, reporting results averaged across the dataset.

To highlight the hierarchical structure of these problems, we note that (1) the confidences for each sub-image $b \in B$ may be given by $classify(b, k)$; (2) the correct answer to the detection problem also answers the classification problem.

Our goal is a general recognition policy that outputs both classification and detection results; we evaluate on both tasks.

3 Multi-class Recognition Policy

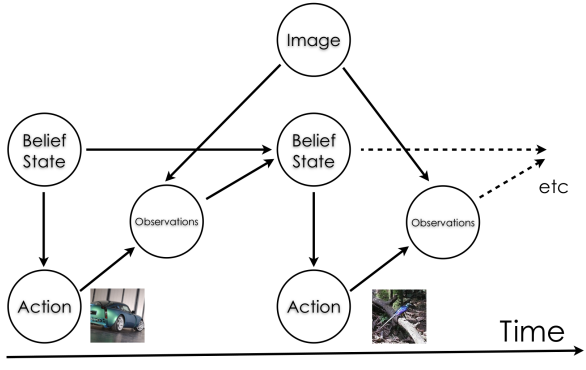


Fig. 2: Summary of our approach to the problem. Our system has two major parts: (1) selecting an action by predicting its value; (2) updating the belief state with observations resulting from the action.

As presented in Figure 2, our goal is a multi-class recognition policy π that takes an image $\mathcal{I}$ and then outputs $\{classify(1), \ldots, classify(K)\}$ and a list of multi-class detection results $detect(I)$.

The policy repeatedly selects an action a_i from a set of actions $\mathcal{A}$, executes it, potentially receives an observation o_i, and selects the next action. The set of actions can include classifiers, detectors, or hybrid actions (detector followed by classification of its output).

A dynamic, or "closed-loop," policy bases action selection on observations received from previous actions, exploiting the signal in inter-object and scene context for a maximally efficient path through the actions. This is our goal, and what sets our formulation apart from multi-class systems that evaluate in a fixed order, such as simple cascades [4] or decision trees.

Let $\mathcal{A}$ consist of K detectors L_i^{det}. We use one-vs-all deformable part-model classifiers on a HOG featurization of the image [5], with associated linear classification of the detections.

Time and Evaluation Each action L has an expected cost $c(\cdot)$ of execution. Depending on the setting, the cost can be defined in terms of algorithmic runtime analysis, an idealized property such as number of *flops*, or simply the empirical runtime on specific hardware. We take the empirical approach: every executed action advances t, the *time into episode*, by its empirical runtime.

As shown in Figure 1, the system is given two times: the setup time T_s and deadline T_d. From the setup time to the deadline, we want to obtain the best possible answer if stopped at any given time. This corresponds to the general notion of *Anytime* algorithms, and is motivated by desired flexibility in the system.

A single-number metric that corresponds to this objective is simply the ratio of the area captured under the curve to the total area between the start and deadline bounds. We evaluate policies by this more robust metric and not simply by the final performance at deadline time for the same reason that Average Precision is used instead of a fixed Precision vs. Recall point.

Sequential Execution An action a_i that consists of running a classifier L_i returns a real-valued observation $o_i \sim P(O_i)$. The state records the fact that a has been taken by adding it to the initially empty set $\mathcal{O}$. We refer to the current set of observations as $\mathbf{o} = \{o_i | L_i \in \mathcal{O}\}$.

We define the belief state b of the decision process by the the distribution over class presence variables $P(\mathbf{C}) = P(C_1, \ldots, C_K)$, where we write $P(C_k)$ to mean $P(C_k = 1)$. Additionally, b records the time into episode t, and the set of executed actions $\mathcal{O}$ with corresponding observations $\mathbf{o}$.

A recognition *episode* takes an image $\mathcal{I}$ and proceeds from the initial belief state b^0 and action a^0 to the next pair b^1, and so on until t exceeds T_d. At that point, the policy is terminated and a new episode begins on a new image.

The policy's performance at time t is determined by the detection and classification observations that have been observed at the last belief state b^j before

that point. For classification of unobserved classes, we treat the corresponding values of $P(\mathbf{C})$ as the set of confidence scores $\{classify(k)\}$. Detection results of unobserved classes are an empty set.

Our notation is summarized in Table 1.

Table 1: Summary of the notation.

$\mathcal{I}$	image
C_k	presence of class $k \in \{1, \ldots, K\}$
t	time into episode
T_s, T_d	start and deadline times
b^j	belief state at step j
π	policy function, $b \mapsto a \in \mathcal{A}$
$\mathcal{A}$	set of actions a
o_i	a real-valued observation upon executing $a_i \in \mathcal{A}$
$\mathcal{O}$	set of executed actions
$\mathbf{o}$	set of observations $\{o_i \vert a_i \in \mathcal{O}\}$
$c(a_i)$	cost of executing a_i, in units of t

4 Selecting actions

As our goal is to pick actions dynamically, we want to formulate a function $V(b, a)$ that assigns a value to a potential action, given the current state of the decision process. We can then define the policy as simply the untaken action with the maximum value:

$$\pi(b) = \underset{a_i \in \mathcal{A} \setminus \mathcal{O}}{\operatorname{argmax}} V(b, a_i) \tag{1}$$

For this policy to be *closed-loop*, the observations o_i generated by taking an action a_i need to update b and thus influence the selection of the next action. Figure 2 visualizes such an execution process.

One simple heuristic value function that we can try is simply picking the action corresponding to the class presence variable C_k with the highest probability. Of course, we'd like to learn the value function from the data, but the above heuristic will be used as a baseline in the evaluation to follow.

Before we discuss how to set $V(b, a_i)$ such that our policy obtains best performance under our evaluation, we present our model for updating the belief state with observations.

4.1 The model of the belief state

The quantities that may be useful to us for selecting which actions to deploy are the probabilities and entropies of the class presence variables C_k. These allow

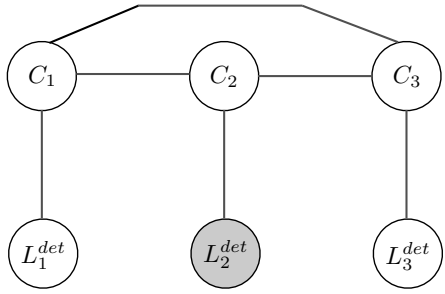


Fig. 3: The MRF inference model used in our system. We show a point in the middle of the decision process for 3 classes; one action has already been taken.

us to look for the most probable classes given the observations. When the policy starts, the model should present the prior distributions $P(C_k)$; as observations are accrued, the model should present the updated conditionals $P(C_k|\mathbf{o})$.

We employ a fully-connected Markov Random Field (MRF), as shown in Figure 3. The L_i variables are discretized from real-valued responses of a classifier on the detections output by the deformable part-model detector we employ (see Section 3).

The classifier is a linear kernel SVM on the top two max detection scores in the list of detections. The responses are discretized per variable based on the distribution of the scores on the training dataset.

The MRF is implemented with an open-source graphical model package [6]. The parameters of the model are trained on fully-observed data. Exact inference is generally intractable in this model. Instead, we use Loopy Belief Propagation, which does not provide general convergence guarantees but has been shown to work well empirically on similar tasks [3].

4.2 Learning the Value Function

Remember that our heuristic value function consisted of picking the action corresponding to the class presence variable with the highest probability $P(C_k)$. We can formulate such a policy as a scalar product:

$$\pi(b) = \underset{a_i \in \mathcal{A} \setminus \mathcal{O}}{\operatorname{argmax}} \, \theta^\top \phi(b, a_i) \tag{2}$$

where $\phi(b, a_i)$ is a feature vector representation of the belief state. This approach is known as function approximation in reinforcement learning [7].

Let us take take the feature representation for an action a_i to be $[P(C_k), 1]$, where C_k corresponds to the action. This corresponds to our heuristic value function, with a bias variable. The feature representation $\phi(b, a_i)$ then is a vector of size $F|\mathcal{A}|$, with $F = 2$ for this feature, where all values are 0 except those corresponding to a_i.

This representation allows us to use a single vector of weights θ as a compact representation of our policy.

At each time step, our system is evaluated by properties of the state b (the list of detections and the classification outputs). The final evaluation metric is a function of the history of execution $h^0 = b^0, b^1, \ldots, b^J$, with J being the last step of the process with $t \leq T_d$.

Ideally, the value function for a point in the decision process b^j should give the expected value of the final evaluation metric, over all possible histories starting at point j:

$$V(b^j, a_i) = \mathbb{E}_{h^j \sim P(h^j|b,a_i)}[R(h^j)] \tag{3}$$

The reward function $R(h^j)$ assigns a real-valued score to a history. We have considerable flexibility in defining R; it does not necessarily have to be directly tied to the final evaluation.

One feature we want the reward function to have is additivity. Let's say that given deadline T_d and some image, the policy had time to take J actions. The total reward of a policy π starting at state b^j is then defined as the sum of rewards

$$R(h^j) = \sum_{j'=j}^{J} R(b'_j, a^{j'}) \tag{4}$$

We formulate two reward definitions: one strives to match the final AP evaluation of the detections, and one is motivated by lowering uncertainty of $P(\mathbf{C})$.

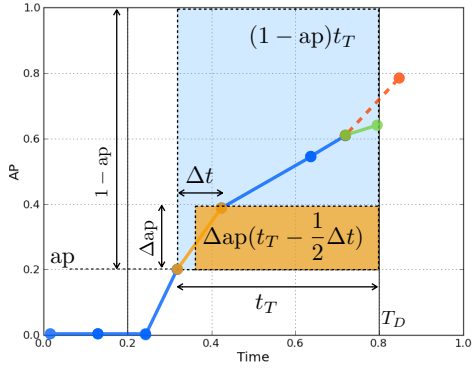


Fig. 4: A per-action greedy value function that corresponds to the maximization of our objective function is the area of the horizontal slice under the curve due to the action. The figure shows this analysis for the action highlighted in orange.

Reward: area under the AP vs. Time curve The final evaluation of a policy consists of the area under the performance vs. time curve (normalized

by the total area). Accordingly, we formulate per-action rewards such that their addition results in this quantity.

Specifically, as shown in Figure 4, we define the reward of an action as

$$R(b^j, a_i) = \frac{\Delta \text{ap}_i (t_T^j - \frac{1}{2}\Delta t_i)}{(1 - \text{ap}^j) t_T^j} \tag{5}$$

where t_T^j and ap^j are the time left until deadline and the AP at state b^j, and Δt_i and Δap_i are the time taken and AP change produced by the action a_i.

The reward is 1 if the policy takes an action that obtains the maximum possible area under the curve at that point, and 0 if no area under the curve is captured.

Reward: decrease in mean entropy We also consider a similar additive reward function based on the mean entropy of the variables $P(C_k)$: $\frac{1}{K}\sum_{k=1}^{K} H(C_k)$, where $H(C_k) = -\sum_{c_k \in 0,1} P(c_k) \log P(c_k)$. We follow the same setup as above, but strive to maximize the area *above* the curve of mean entropy vs. time.

The goal of a policy that maximizes these rewards is to take actions that reduce uncertainty of the most uncertain variables most quickly.

4.3 Learning the weights

The feature representation of the belief state for a given action a_i that corresponds to class k is defined to be

$$\phi_k(b) = [P(C_k),\ H(C_k),\ 1] \tag{6}$$

Our procedure for learning the weights is standard generalized policy iteration [7]. We first initialize the weights θ to the heuristic value function of simply picking the maximum $P(C_k)$: $\theta_i = [1, 0, 0]$.

With this policy, we run N recognition episodes. From the state-action samples gathered in running the episodes, we formulate a matrix Φ from the featurizations $\phi(b^j, a_i)$, and a vector y consisting of the individual rewards $R(b^j, a_i)$.

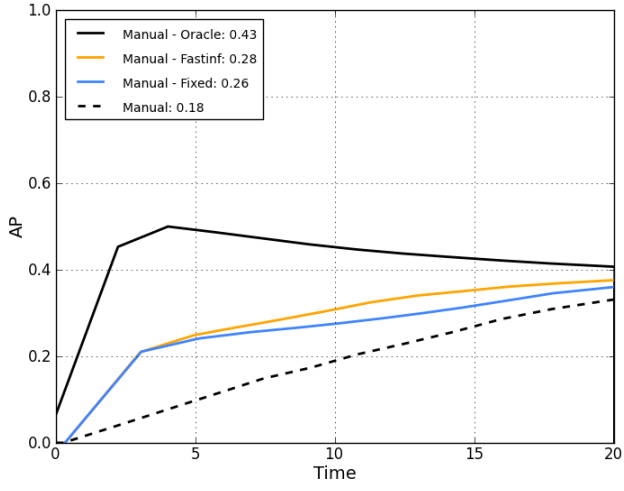
The rewards are computed as the sum of discounted rewards to the end of the episode:

$$R(b^j, a_i) = \sum_{i=0}^{J-j} \lambda^i R(b_{j+i}, a^{j+i}) \tag{7}$$

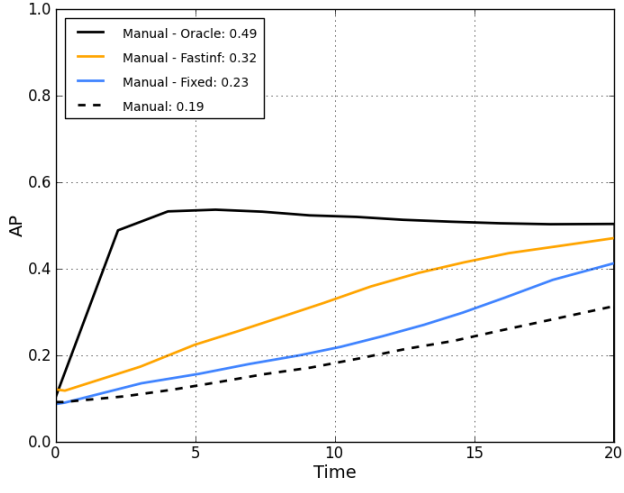
Note that with $\lambda = 0$, the reward is determined entirely by the actual action taken; with $\lambda = 1$, the reward is the sum of all rewards until the end of the episode.

We then solve the system $\Phi\theta = y$ for θ with Lasso regression. With the updated weights θ, we run N more episodes and repeat the procedure until convergence.

The parameters of this training procedure are the α weight on the regularization term in the regression and the λ discount weight. We cross-validate for both values.



(a) Detection



(b) Classification

Fig. 5: Heuristic value function.

5 Evaluation

To summarize, we evaluate our system in the multi-class, multi-label detection and classification regime. We evaluate on the PASCAL VOC 2007 dataset: train-

ing on the training and validation data and testing on the test set. We run our policies on all images in the test set, cutting off execution at T_d.

To evaluate performance at a given time, we gather all detections and the classification answers found to this point in all the recognition episodes. As described before, we evaluate detection performance by averaging per-image performance in the multi-class regime. We evaluate classification performance by pooling the ground truth across the dataset, in standard procedure.

We establish the baseline performance of our system by selecting actions randomly at each step. The time setting for our evaluation is $T_s = 0$, $T_d = 20$ (in seconds). As shown in Figure 5, the random policy results in a roughly linear gain of performance vs. time.

To establish an upper bound on performance, we also plot the Oracle curve, obtained by re-ordering the actions with the hindsight of the results they produced. Note that the curves turn downward in this setting: this is due to the different performance levels of the detectors: as the weaker detectors are inevitably run, they introduce more false positives and false negatives than true positives.

Figure 5 also shows the performance of our policy with the heuristic value function and two inference models: the MRF model and a simple fixed-order model that follows the priors. We can see that due to the dataset bias, the fixed-order policy performs well at first (the person class is disproportionately likely to be in the image), but is overtaken by the proper inference model as execution goes on.

Figure 6 shows the performance of the polices with learned weights, comparing our two reward functions. We found the entropy-based reward to work better than the AP-based reward.

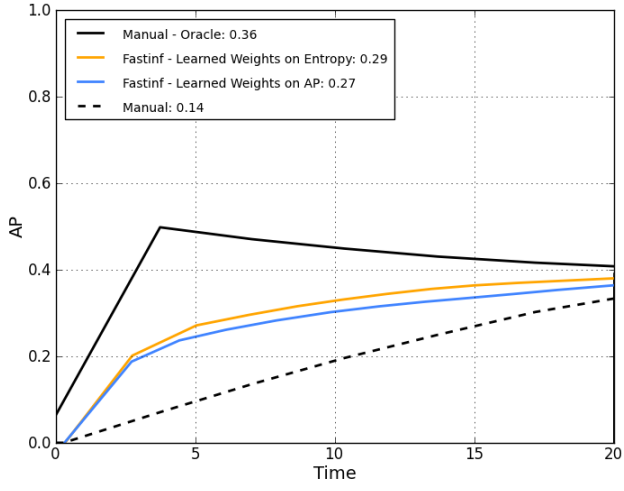
A notable difference between the MRF inference model and the fixed-order model was in the weights learned: while the fixed-order model learned weights onto the bias feature only, the MRF inference model learned weights onto a mix of the entropy and probability features.

6 Related Work

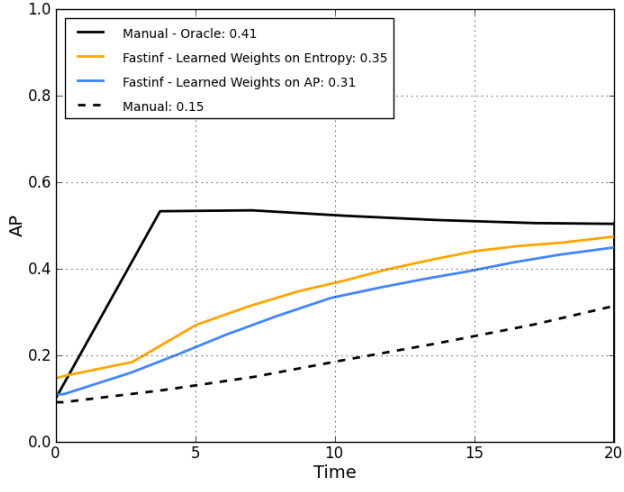
The literature on object detection is vast. Here we briefly summarize detection work that sufficiently contextualizes our contribution.

An early success in efficient object detection used simple Haar features to build up a *cascade* of classifiers, which then considered image regions in a sliding window regime [4]. Although the simple features and classifiers of this method have been surpassed by more complex detectors, the idea of cascading classifier evaluations is still often used.

The best recent performance has come from detectors that use gradient-based features to represent objects as either a collection of local patches or as object-sized windows [8,9]. Usually, SVM classifiers are used to learn separating hyperplanes in such feature spaces between objects of a given class and all other possible contents of an image window.



(a) Detection



(b) Classification

Fig. 6: Entropy reward function.

Window proposal is often done exhaustively over the image space, in a "sliding window" fashion. Using "jump windows" (window hypotheses voted on by local features) as region proposals is another common idea [10,11,12]. For local feature-based approaches, a bounded search over the space of all possible

windows works reasonably well [13]—however, the method has not been able to obtain state-of-the-art performance.

Although none of the best-performing systems treat window proposal and evaluation as a closed-loop system, with feedback from evaluation to proposal, some work has been done in the area, mostly inspired by ideas from biological vision and attention research [14,15,16].

Anytime performance in vision systems is a surprisingly little-explored idea. A pioneering recent paper picks features with maximum value of information in a Hough-voting framework, and explicitly evaluates performance vs. time [17].

Inherently multi-class detection has its own line of work, focusing largely on making detection time sublinear in the number of classes through sharing features [18,19,20]. A recent post-processing extension to detection systems uses structured prediction to incorporate multi-class context as a principled replacement for the common post-processing step of non-maximum suppression [3].

Context of course has a long history in vision. One source of context is the scene or non-detector cues; for the PASCAL VOC, these are quantitatively considered in [21]. Another source is inter-object context, used for detection in a random field setting in [22]. A critical summary of the main approaches to using context for object detection is given in [23].

References

1. Everingham, M., Van Gool, L., Williams, C.K.I., Winn, J., Zisserman, A.: The PASCAL VOC Challenge 2010 Results. http://www.pascal-network.org/challenges/VOC/voc2010/workshop/index.html (2010) 1, 3
2. Deng, J., Dong, W., Socher, R., Li, L.J., Li, K., Fei-Fei, L.: ImageNet: A Large-Scale Hierarchical Image Database. In: CVPR09. (2009) 1
3. Desai, C., Ramanan, D., Fowlkes, C.: Discriminative models for multi-class object layout. In: 2009 IEEE 12th International Conference on Computer Vision, Ieee (2009) 229–236 3, 6, 12
4. Viola, P., Jones, M.: Rapid object detection using a boosted cascade of simple features. In: CVPR. (2001) 4, 10
5. Felzenszwalb, P.F., Girshick, R.B., McAllester, D., Ramanan, D.: Object detection with discriminatively trained part-based models. PAMI **32** (2010) 1627–45 4
6. Jaimovich, A., Mcgraw, I.: FastInf : An Efficient Approximate Inference Library. Journal of Machine Learning Research **11** (2010) 1733–1736 6
7. Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction. MIT Press (1998) 6, 8
8. Dalal, N., Triggs, B.: Histograms of Oriented Gradients for Human Detection. In: CVPR, Ieee (2005) 886–893 10
9. Lowe, D.G.: Distinctive Image Features from Scale-Invariant Keypoints. International Journal of Computer Vision **60** (2004) 91–110 10
10. Chum, O., Zisserman, A.: An Exemplar Model for Learning Object Classes. In: CVPR, Ieee (2007) 1–8 11
11. Vedaldi, A., Gulshan, V., Varma, M., Zisserman, A.: Multiple kernels for object detection. ICCV (2009) 606–613 11
12. Vijayanarasimhan, S., Grauman, K.: Large-Scale Live Active Learning: Training Object Detectors with Crawled Data and Crowds. In: CVPR. (2011) 11

13. Lampert, C., Blaschko, M.: A Multiple Kernel Learning Approach to Joint Multi-class Object Detection. Lecture Notes in Computer Science: Pattern Recognition **5096** (2008) 12
14. Butko, N., Movellan, J.: Optimal scanning for faster object detection. CVPR (2009) 2751–2758 12
15. Vogel, J., de Freitas, N.: Target-directed attention: Sequential decision-making for gaze planning. ICRA (2008) 2372–2379 12
16. Paletta, L., Fritz, G., Seifert, C.: Q-learning of sequential attention for visual object recognition from informative local descriptors. In: ICML, New York, New York, USA, ACM Press (2005) 649–656 12
17. Vijayanarasimhan, S., Kapoor, A.: Visual Recognition and Detection Under Bounded Computational Resources. In: CVPR. (2010) 1006–1013 12
18. Torralba, A., Murphy, K.P., Freeman, W.T.: Sharing visual features for multiclass and multiview object detection. IEEE transactions on pattern analysis and machine intelligence **29** (2007) 854–69 12
19. Fan, X.: Efficient Multiclass Object Detection by a Hierarchy of Classifiers. In: 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05), Ieee (2005) 716–723 12
20. Razavi, N., Gall, J., Gool, L.V.: Scalable Multi-class Object Detection. In: CVPR. (2011) 12
21. Divvala, S., Hoiem, D., Hays, J., Efros, A., Hebert, M.: An empirical study of context in object detection. In: CVPR, Ieee (2009) 1271–1278 12
22. Torralba, A., Murphy, K.P., Freeman, W.T.: Contextual Models for Object Detection Using Boosted Random Fields. MIT Computer Science and Artificial Intelligence Laboratory Technical Report (2004) 12
23. Galleguillos, C., Belongie, S.: Context based object categorization: A critical survey. Computer Vision and Image Understanding **114** (2010) 712–722 12