Context as Supervisory Signal: Discovering Objects with Predictable Context

Carl Doersch, Abhinav Gupta Alexei A. Efros
Carnegie Mellon University UC Berkeley

Abstract. This paper addresses the well-established problem of unsupervised object discovery with a novel method inspired by weakly-supervised approaches. In particular, the ability of an object patch to predict the rest of the object (its context) is used as supervisory signal to help discover visually consistent object clusters. The main contributions of this work are: 1) framing unsupervised clustering as a leave-one-out context prediction task; 2) evaluating the quality of context prediction by statistical hypothesis testing between *thing* and *stuff* appearance models; and 3) an iterative region prediction and context alignment approach that gradually discovers a visual object cluster together with a segmentation mask and fine-grained correspondences. The proposed method outperforms previous unsupervised as well as weakly-supervised object discovery approaches, and is shown to provide correspondences detailed enough to transfer keypoint annotations.

Keywords: context, prediction, unsupervised object discovery, mining

1 Introduction

Proponents of unsupervised representation learning [1,2,3,4] and unsupervised object discovery [5,6,7,8,9,10,11,12] have long argued that these approaches have the potential to solve two fundamental problems with supervised methods. The first is obvious: training labels are expensive to collect. More subtly, human annotations can introduce unwanted biases into representations [13]. Unsupervised object discovery has, however, proven extremely difficult; one state-of-the-art result [3] uses a million CPU-hours, yet reports only three discovered objects (cats, faces, and bodies), and the "neurons" sensitive to these objects could only be identified through the use of labeled data.

At its core, object discovery is a clustering problem; the goal is to group together image regions (patches or segments) that depict the same object. Standard clustering algorithms like K-means rely on a good distance metric, but unfortunately, distances in different regions of the feature space often aren't comparable [14]. This means that the "tightness" of each cluster will be a poor measure of whether it actually depicts an object. A number of recent works have argued that weak supervision can be an effective way to get more visually meaningful clusters [14,15,16,17,18,19,20,21,22,23]. The supervision (e.g., scene labels, GPS coordinates, etc.) gives information about which image regions should be

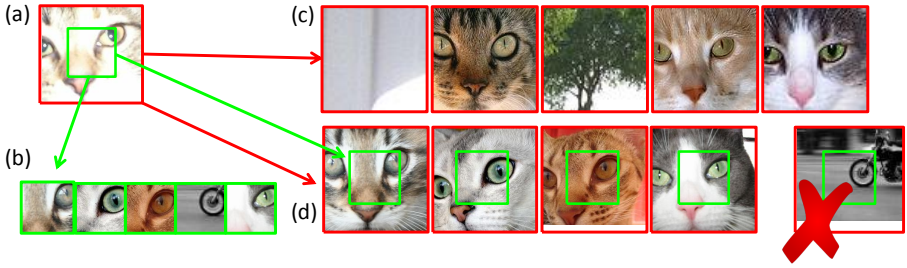

Fig. 1. Suppose we want to create a visually meaningful cluster containing the cat eye patch in (a). (b) shows a cluster produced by simple nearest neighbors, but there is too little information in just a cat eye, so it is confused with a motorbike. Nearest neighbors on a larger patch (c) introduces new errors because the patch now captures too much variation. Our proposed method (d) starts with the cluster in (b) and uses the context as a "supervisory signal" to discard the incorrect match.

close together (e.g., belong to the same cluster) and which should be far apart. But can a similar effect be achieved without any supervision?

The main contribution of this paper is the use of *context* [24] as a supervisory signal. At a high level, context provides similar information as a weak label: e.g., given a set of matched cat eye patches on Figure 1b, we expect the context surrounding those patches to depict cat faces. Errors in the matching (e.g. the motorcycle wheel) can then be detected and discarded because the context will not match. (One might object that we could simply include context as part of the feature used for matching, but Figure 1c shows that this performs poorly, as it is unable to handle the large variations between the cat faces).

Using context as a supervisory signal means we need a way to determine whether two contexts are sufficiently similar. However, standard distance metrics will be just as unreliable at measuring the visual similarity of the context as the visual similarity of the patches themselves. An 'easy' context (e.g., a uniform region) will have too many matches, whereas a 'difficult' context (e.g., a complex shape) will potentially match nothing. Our key insight is to normalize for this, by modeling the 'difficulty' of the context. Mathematically, our formulation is reminiscent of statistical hypothesis testing for object recognition [25]. For a given image, we have two competing hypotheses: 1) that the context in that image is best described as a 'thing,' i.e. an object with a well-defined shape, versus 2) that the image is best described as 'stuff' [26], i.e. that it is best modeled using low-level image statistics. Both models "predict" what the context will contain, i.e. they produce a probability distribution in image feature space, such that we can compute a single probability value for the image context. If the *thing* model predicts better, then the cluster is likely a good one, and the patch is likely a member of it. We perform a simple likelihood ratio test to determine if this is the case.

At what granularity should our models be allowed to predict? If we force the *thing* model to predict a cat face all at once, even a correct prediction might align poorly with the ground truth. Evaluating whether such a prediction is correct then becomes difficult. Making small predictions near the initial patch will be

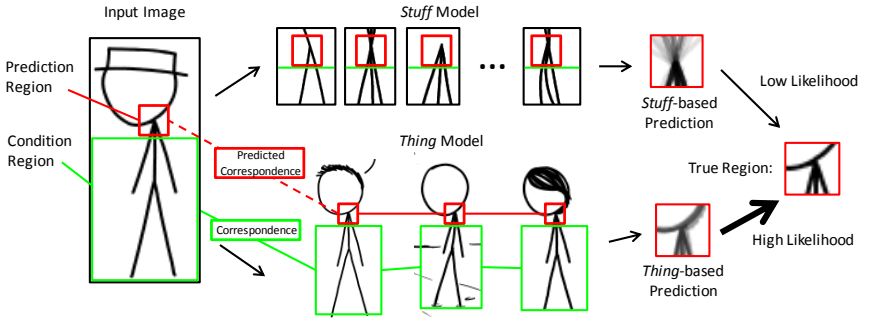

Fig. 2. Algorithm overview. Given a "condition" region (in green), our algorithm predicts the "prediction" region (in red) twice: once using a model that assumes correspondence with some complex shape (the *thing* model, bottom), and once assuming that the region is best modeled as texture (the *stuff* model, top). Both models' predictions are compared to the true region. If *thing* outperforms *stuff*, the prediction region is considered to be part of the discovered object. This process then repeats with the a new prediction region (anatomically accurate stick-figures from xkcd [28])

easier because errors due to misalignment will be small, but they will contain little information. Our approach finds middle ground by iteratively predicting small regions over a larger area. Between each prediction, we align the model to the true data. That is, we "grow" the predicted region one small step at a time, reminiscent of texture synthesis [27]. The model's alignment errors are thus corrected before they drift too far.

2 Overview

At a high level, our pipeline is similar to algorithms for mid-level patch discovery [16,15], especially in the early stages. Like [16], we first sample a large number of random patches (10,000 for our PASCAL VOC experiments), and then find the top few nearest neighbors in HOG feature space for each of them, across the entire dataset. ([16] uses normalized correlation as a distance metric, but we found Exemplar LDA [29], with a Gaussian learned from the entire unlabeled dataset, to give slightly better matches). These cluster proposals form the input to our object discovery algorithm. At the high level, the algorithm: 1) discards patches within each cluster whose context is inconsistent with the other patches, 2) ranks the clusters, and 3) discards clusters that do not contain visually consistent objects. The ranking (i.e. 'score' of a cluster) is simply the sum of the scores of patches that weren't discarded. Thus, the meat of our algorithm boils down to the process of scoring the context around a single patch. A given patch is scored using all the other patches in the cluster using *leave-one-out prediction*. That is, given n patches in a cluster, we use the context associated with patch 1 through patch $n-1$ to predict the context around the n'th patch.

But as was discussed earlier, a major difficulty is that some contexts are easier to predict than others. For instance, given a patch of blank wall, it's easy to predict that the context will be similarly blank. If we don't account for

this, then any cluster that's full of blank patches (or any other simple texture) might be declared an object. Note, however, that this prediction doesn't really require the algorithm to understand that the patch is a wall; a highly accurate prediction could be made just based on low-level statistics. Hence, we don't measure how well the context of a patch can be predicted, but instead, how much the clustering *helps* us predict the context. Specifically, our algorithm uses two models that produce two predictions. The first—the *stuff* model—produces predictions based solely on knowledge of low-level image/texture statistics, which it extrapolates from the single patch whose context it is predicting. This model could easily predict that a blank wall will continue indefinitely. The other—the *thing* model—uses the specific correspondence defined by the cluster to make its predictions. Figure 2 illustrates why this is effective. The initial patch cluster (which generates the correspondence outlined in green) contains the bodies of the stick figures. The *thing* model can align these bodies and predict the presence of the neck. The *stuff* model, however, uses only low-level image statistics and predicts (incorrectly) that the contours will most likely continue straight. We then compare the likelihoods; the patch is considered a member of the cluster if the *thing* likelihood is significantly higher than the *stuff* likelihood.

To make this algorithm work as stated, however, we must compute the likelihood $P(c|p)$ of the context c given the patch p, under two separate models, and do so with reasonable accuracy. The problem of generative image modeling has a long history in computer vision [1,4,25,30,31,32], but historically these algorithms have performed poorly for object recognition problems, especially compared to the discriminative methods that have, of late, largely displaced them in the field. A core difficulty shared by generative methods is that they assume the image features are independent, conditioned on some set of latent variables. Obtaining a likelihood $P(c|p)$ requires integrating out those latent variables, which is generally intractable. Approximations (e.g. MCMC or variational methods) either do not scale well, or produce probability estimates that cannot be compared between different models. To get around this problem, our algorithm partitions the context c into small regions c_k (for example, if c is the HOG representation of the context, each c_k may be a single cell.) Next, we factorize the conditional likelihood as follows:

$$P(c|p) = \prod_{k=1}^{m} P(c_k|c_1, ..., c_{k-1}, p) \tag{1}$$

Here, the ordering of the c_k's can be whatever makes the computation easiest (in the case of HOG, c_k may be adjacent to the region covered by $\{c_1, ..., c_{k-1}, p\}$.) This deceptively simple algebraic manipulation—really just an application of the probability chain rule—is remarkably powerful. First, note that it is not an approximation, even though it makes no independence assumptions. It remains tractable because the c_k's are actually observed values; unlike in latent-variable models, the c_k's do not need to be integrated out in order to compute a valid likelihood. Furthermore, each c_k may be chosen so that its conditional distribution is well approximated with a simple parametric distribution (we find that a single HOG cell is well approximated by a Gaussian), even though we do not

assume anywhere that the joint distribution has a parametric representation. Despite these good properties, we have found no previous work which attempts to compute image likelihoods using such a factorization (note that [31] comes close; however, their model still contains higher-order potentials, meaning that the terms in the product cannot be computed independently, and that inference still requires MCMC). We show that these incremental predictions can be made efficiently and with surprising accuracy, enough that the resulting likelihoods can be compared between our *thing* and *stuff* models.

3 Algorithm

We first formalize our notation. Assume we have a cluster proposal containing n patches. We select one 'held out' patch, and number it 0 (the others are numbered 1 through $n-1$). Let H^0 denote the feature representation for the image containing patch number 0, which we will call the *query* image. Let H^0_k be the k'th feature in H^0, in our case, a single HOG cell. Let $\mathcal{P}$ index the subset of features in H^0 that were inside patch 0 (in Figure 2, $\mathcal{P}$ would be a strict subset of the region outlined in green for all but the first term in the product in Equation 1). Finally, let $\mathcal{C}$ be an *ordered* set of indices for the features in the context, i.e. the complement of $\mathcal{P}$ (in Figure 2, $\mathcal{C}$ indexes the remainder of the green, the red, and also the rest of the image). This means we predict $\mathcal{C}[1]$ using $\mathcal{P}$ alone, $\mathcal{C}[2]$ using $\mathcal{P} \cup \mathcal{C}[1]$, and so on. $\mathcal{C}[1:t]$ indexes the first t HOG cells in the context that get predicted. Our original factorization (Eq. 1) for the *thing* model can now be written more formally as:

$$P_T(H^0_{\mathcal{C}}|H^0_{\mathcal{P}}) = \prod_{t=1}^{|\mathcal{C}|} P_T(H^0_{\mathcal{C}[t]}|H^0_{\mathcal{C}[1:t-1]}, H^0_{\mathcal{P}}) \tag{2}$$

We will have a similar factorization for P_S of the *stuff* model. In Figure 2, the region outlined in red corresponds to $H^0_{\mathcal{C}[t]}$, and those outlined in green correspond to $\{H^0_{\mathcal{C}[1:t-1]}, H^0_{\mathcal{P}}\}$. We repeat this computation of $P_.(H^0_{\mathcal{C}[t]}|H^0_{\mathcal{C}[1:t-1]}, H^0_{\mathcal{P}})$ for all t; i.e. at the next iteration, the red region will get added to the green region and we'll choose a new red region.

For simplicity, we assume that the conditional distributions are Gaussian for both *thing* and *stuff* models; we find empirically that forcing both *thing* and *stuff* predictions into the same, simple family makes the likelihoods more comparable. To ease exposition, we'll call the HOG cells $\{H^0_{\mathcal{C}[1:t-1]}, H^0_{\mathcal{P}}\}$ the "condition" region, and $H^0_{\mathcal{C}[t]}$ the "prediction" region. We choose the order of $\mathcal{C}$ by increasing distance from the center of the patch; this means that, for each HOG cell we predict, at least one of its neighbors will be in the condition region.

3.1 *Stuff* Model

To construct the *stuff* prediction $P_S(H^0_{\mathcal{C}[t]}|H^0_{\mathcal{C}[1:t-1]}, H^0_{\mathcal{P}})$ (which we will abbreviate as $p^S_{\mathcal{C}[t]}$), the simplest approach is to 1) extract a subset of the condition region that is spatially close to $\mathcal{C}[t]$, 2) find nearest neighbors for that subset from a large database of images, and 3) use the context around those retrieved

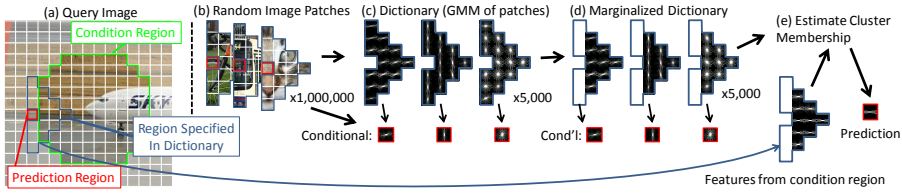

Fig. 3. Summary of our *stuff* model. (a) Given a condition region (green), we extract cells (blue) that are near to the prediction region (red). We assume we have a dictionary reminiscent of visual words (in our case, a GMM) learned from many sampled patches (b-c). For each dictionary element, we estimate the conditional distribution over the prediction region (red). We remove cells that aren't in the condition region (d) before assigning the extracted cells to the dictionary. Finally, we use the associated conditional distribution as our prediction.

neighbors to form our prediction. Of course, this would be extremely computationally expensive, so we instead summarize our dataset using clustering, in a manner reminiscent of visual words.

Our more efficient approach is shown in Figure 3. We begin with a query image (Figure 3a), with a condition region (in green) and a prediction region (in red). We assume that we have available a 'dictionary' (Figure 3c) constructed from a large sample of image patches (Figure 3b), each of which was in a shape that's similar (but not necessarily identical) to the shape of the selected subset of the condition region (which is outlined in blue in Figure 3a). We learn 12 separate dictionaries to ensure that we always have a reasonably good match to a given local condition region. To construct these dictionaries, we first sample about a million such patches (Figure 3b), and learn a Gaussian Mixture Model (GMM) from the HOG features of these image patches. We temporarily ignore the region of these patches that corresponds to the prediction region (outlined in red in Figure 3b) and learn the GMM only on the rest. We restrict each GMM component to have a diagonal covariance matrix for computational efficiency. We use 5000 GMM components, and show some centroids in Figure 3c. We also estimate, for each component of the GMM, the prediction that will be made by this component for the red region. For this, we first soft-assign each of our sampled patches to the components of the GMM, and compute the empirical mean and covariance of the associated red cells for each component. This mean and covariance are interpreted as the parameters a Gaussian conditional distribution; we show the means of these conditional distributions outlined in red in Figure 3c.

To actually make a prediction, we first determine which components of the GMM should be responsible for the prediction. We soft-assign the condition region of our query image (specifically, the subset outlined in blue) to the components of our GMM. Unfortunately, there may be dimensions of our GMM components that correspond to HOG cells outside the condition region; for instance, the leftmost cells highlighted in blue in Figure 3a). To deal with this, we marginalize out any such cells from the GMM as shown in Figure 3d (Hence why

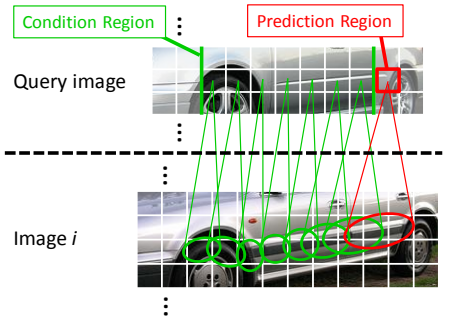

Fig. 4. Our *thing* model predicts the prediction region (red rectangle) given the condition region (green border) in the query image. We estimate correspondence for the prediction region (red ellipse)—i.e. regions in other images likely to have similar contents—as the basis for this prediction. The red correspondence must be obtained without observing the prediction region, so we first estimate correspondence for the condition region (green ellipses) and extrapolate to the prediction region.

we use GMM's instead of K-means, as the marginalization of a GMM is well-defined). We next soft-assign the image data to the components of the GMM, which gives us a weighted set of conditional distributions over the prediction region. We average these predictions into a single Gaussian (specifically, we treat the set of predictions as a GMM over a single HOG cell, and compute a single Gaussian that matches the mean and variance of this GMM).

3.2 *Thing* Model

The *thing* model attempts to capture the details of a complex shape using the set of images that were retrieved when we built our initial patch cluster. Making a prediction for a particular prediction region $P_T(H^0_{\mathcal{C}[t]}|H^0_{\mathcal{C}[1:t-1]}, H^0_{\mathcal{P}})$ (which we will abbreviate as $p^T_{\mathcal{C}[t]}$) boils down to the problem of correspondence: if we can estimate which regions in the other images are likely to correspond to our current prediction region, then we can predict that the features will be the same. To avoid biasing the likelihood value, we must not to access the features $H^0_{\mathcal{C}[t]}$ while making the prediction, but there are cells in the condition region near the prediction region that we could use. Hence, we find the correspondence for each cell in the condition region (a standard image warping problem). Once we have this correspondence, we extrapolate it to the prediction region. Note, though, that we cannot assume this correspondence will exist for every image. Besides the standard problem of occlusion, we also have to deal with the many errors in the Exemplar-LDA matching. We have found that the top 20 matches are usually reasonably pure, but for some interesting objects the lower-ranked matches may be wrong. Hence, we only use the top 20 images per prediction, meaning we must use the data extremely efficiently.

Formally, recall that $\mathcal{C}$ and $\mathcal{P}$ index the HOG cells in our query image, and so each index in these sets can be written as 2-dimensional points (x, y) on the grid of HOG cells. We represent the correspondence as a mapping $f^i(x, y)$ from the cell (x, y) in the query image to cells in the HOG grid for image i, where i ranges from 1 to $n-1$ (we'll call these the 'predictor' images). We optimize our mapping such that $H^i_{f^i(x,y)}$ is as similar as possible to $H^0_{(x,y)}$ for all (x, y) *in the condition region* (we defer the computation of f to Appendix A). Note that we are interested in correspondence for $\mathcal{C}[t] = (x_t, y_t)$, the *prediction* region, but we

aren't allowed to access the HOG feature at $H^0_{\mathcal{C}[t]}$; therefore (x_t, y_t) isn't, strictly speaking, in the domain of f^i. To find correspondence for the prediction region, we find the nearest point (x^*_t, y^*_t) in the condition region and compute a simple linear extrapolation:

$$f^i(x_t, y_t) = f^i(x^*, y^*) + (x_t, y_t) - (x^*_t, y^*_t) \tag{3}$$

Thus far, we've treated $f^i(x_t, y_t)$ as if it indexed a single HOG cell, but is that enough? Recall that we have about 20 images; actually less than the dimensionality of HOG! Worse, correspondence is often ambiguous. Consider the example in Figure 4. The condition region contains the front wheel of a car and some of the car's side panel, and we are interested in a prediction region further to the right. Ideally, the algorithm should give some probability mass to the event that the panel will continue, and some mass to the event that it will end and a wheel will start. However, if $f^i(x_t, y_t)$ returns a single point as the correspondence for the prediction region, then the algorithm will be arbitrarily confident that *either* the prediction region should contain a continuation of the panel *or* that it will end. To address this, we alter our definition of f such that its range is the space of tuples of mean vectors and covariance matrices parameterizing 2-d Gaussian distributions.

$$f^i(x, y) := \left[\mu^i_{x,y}, \Sigma^i_{x,y}\right] \tag{4}$$

Thus, $f^i(x, y)$ defines Gaussian distribution over the HOG grid of image i. (In Equation 3, the addition is only performed on μ: i.e. $f^i(x, y) + (a, b) = [\mu^i_{x,y} + (a, b), \Sigma^i_{x,y}]$). Figure 4 visualizes these Gaussians as ellipses. In this illustration, note that the covariance of the Gaussians are small near the wheel (where there is less ambiguity), but they grow as the matching becomes more ambiguous. While this makes the optimization of f somewhat more complicated, ultimately it means the algorithm uses more data to make each prediction, and in the case of Figure 4 guesses that the prediction region could correspond to panel or to wheel.

This correspondence allows us to extract many HOG cells from each image that may correspond to the prediction region; to actually form a prediction, we aggregate these samples across all predictor images, with each sample weighted by the likelihood that it actually corresponds to the prediction region. Mathematically, we form our prediction by fitting a Gaussian in HOG feature space to the *weighted* set of HOG cells in $H^1 ... H^{n-1}$ that the prediction region potentially corresponds to:

$$p^T_{\mathcal{C}[t]} = \mathcal{N}(H^0_{\mathcal{C}[t]}; \mu^f_{\mathcal{C}[t]}, \Sigma^f_{\mathcal{C}[t]}) \tag{5}$$

where

$$\mu^f_{\mathcal{C}[t]} = \sum_{i,u,v} \eta^i_{\mathcal{C}[t],u,v} H^i_{u,v}\ ; \quad \Sigma^f_{\mathcal{C}[t]} = \sum_{i,u,v} \eta^i_{\mathcal{C}[t],u,v} (H^i_{u,v} - \mu^f_t)(H^i_{u,v} - \mu^f_t)^\top \tag{6}$$

Here, $\sum_{i,u,v} \eta^i_{\mathcal{C}[t],u,v} = 1$. There are two components of this weighting: $\eta^i_{\mathcal{C}[t],u,v} = w^i_{\mathcal{C}[t],u,v} \omega^i_{\mathcal{C}[t]}$. The first is based on the spatial correspondence f, and is defined

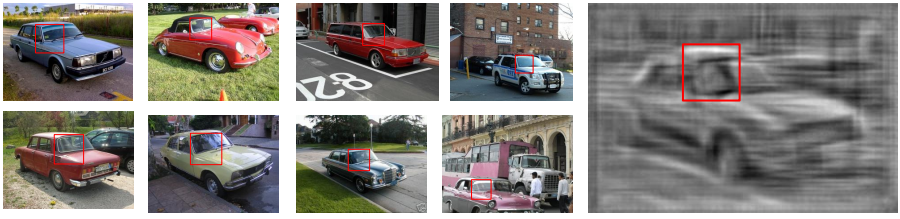

Fig. 5. An example of our *thing* model running in "generative mode." As we run the alignment procedure, we generate each HOG cell in the prediction image as the average of the cells it corresponds to in the predictor images (we allow the algorithm to make a fresh prediction for each cell at each iteration, to allow better alignment). Starting only from the images on the left and the element-level correspondence shown by the red bounding boxes, we can synthesize a new, plausible car in HOG space, which we then render using inverse HOG [33].

as $w^i_{\mathcal{C}[t],u,v} = \mathcal{N}([u,v]; \mu^i_{x_t,y_t}, \Sigma^i_{x_t,y_t})$ for prediction region (x_t, y_t) (normalized to sum to 1 across u and v). This weight, however, is not sufficient by itself, because the correspondence from the prediction region to image i might be completely wrong (e.g. if there is nothing in image i that corresponds to the prediction region). Hence, we use $\omega^i_{\mathcal{C}[t]}$ to downweight the images where we expect the correspondence to be incorrect. Intuitively, we simply observe how useful image i was for the earlier predictions of other regions near to $\mathcal{C}[t]$. The mathematical details are somewhat involved and not required for understanding the rest of the algorithm, so we defer them to Appendix B.

3.3 Determining what to predict

A remaining problem is that our *thing* model generally won't do a good job predicting every cell in the query image, since the object of interest may not fill the image, or it may be occluded. One possible resolution is to throw away any region where the *thing* model predicted poorly, but we find that this biases the entire algorithm toward overestimating the probability that the image is a *thing* (much like a gambler who judges his luck based only on the days when he won). A better solution is to have the *thing* model gracefully degrade to 'mimic' the *stuff* model when it believes it can't predict well. For simplicity, our algorithm makes a binary decision. Either it uses the correspondence-based algorithm (Equation 5) exclusively, or it 'mimics' the *stuff* model exactly (i.e. it sets its conditional distribution $P_T(h|H^0_{\mathcal{C}[1:t-1]}, H^0_{\mathcal{P}})$ equal to $P_S(h|H^0_{\mathcal{C}[1:t-1]}, H^0_{\mathcal{P}})$ for all h). Note that when the *thing* model mimics the *stuff* model, $p^T_{\mathcal{C}[t]}$ will be equal to $p^S_{\mathcal{C}[t]}$, and hence the value of $H^0_{\mathcal{C}}$ will have no effect on the likelihood ratio score. To determine when the *thing* model should mimic the *stuff* model, we use two heuristics. First, we measure how well the *thing* model has predicted the query image in regions near the current prediction region. Second, to do a better job estimating the bounds of the object, we measure whether the *thing* model believes $\mathcal{C}[t]$ corresponds to regions in other images that were predicted poorly. For implementation details, see Appendix C.

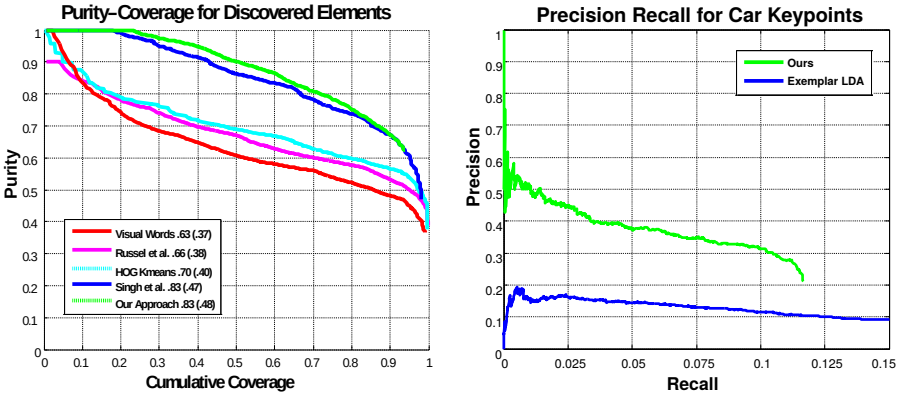

Fig. 6. Left: purity vs coverage for objects discovered on a subset of Pascal VOC 2007. The numbers in the legend indicate area under the curve (AUC). In parentheses is the AUC up to a coverage of .5 (lower ranked clusters generally aren't objects). Right: precision-recall for car keypoints transferred via unsupervised correspondence.

3.4 Details of our discovery pipeline

The above sections outline a verification procedure that tells us whether the *thing* model predicted better than the *stuff* model for each individual element detection. However, one final difficulty is that a single cluster initialized by exemplar-LDA may actually contain two separate objects, and the verification procedure will happily verify both of them. To prevent this, we start by verifying a single patch, and attempt to grow the cluster in a way that selects for the object depicted in that first patch. After the first prediction, we can compute a "usage score" for every predictor image (see Appendix B), which captures how much that image helped predict the query image. We take the top image according to this score and compute its likelihood ratio, which produces more usage scores for the predictor images. We average the resulting usage scores for each image. We use 20 predictor images for each verification. To choose them, we first select at most 10 of the verified images with highest usage score, and for the rest we select the unverified images with the top exemplar-LDA score.

4 Results

Our goal is to demonstrate unsupervised object discovery on realistic databases with as little human intervention as possible. Our experiments focus on the PASCAL VOC, a challenging database for modern, supervised object detection algorithms. We evaluate discovery on PASCAL object categories, and also show results for keypoint transfer on the "car" category.

4.1 Quantitative Results: Purity-Coverage on PASCAL VOC 2007

Following the experimental setup of [15], we perform unsupervised object discovery on all PASCAL VOC 2007 images containing a dining table, horse, motorbike, bus, train, or sofa. We evaluate the quality of the discovered objects

using the purity-coverage curve [14], which we compute in three steps: 1) for each of our discovered patch clusters, we select the top 10 patches (as scored by the likelihood ratio value); 2) we compute the purity of each cluster using these 10 patches, according to the majority VOC label in the cluster; 3) we sort the clusters by purity. To obtain the k'th point on the purity-coverage curve, we plot the average purity for the first through k'th clusters in this ranking versus the total number of images that contribute at least one patch to the clusters 1 through k. (Note that, unlike [15], we follow [14] and plot purity and coverage on the same graph, since we find it makes the methods more comparable.) The result is shown in Figure 6 left. We slightly outperform [15], especially for the high purity regime (we get nearly 10% extra coverage before we make our first mistake). However, note that [15] is not completely unsupervised as it requires a "natural world" dataset, which typically contains a somewhat different distribution of visual data than the "discovery" dataset, providing an implicit, albeit very weak, supervision. Our method significantly outperforms other, truly unsupervised methods.

Implementation details : We start with over 10,000 randomly-sampled image patches at multiple resolutions to initialize the clusters. Cluster verification is relatively computationally expensive, so in practice we terminate the verification of each cluster as soon as it appears to be producing low scores. We start by verifying a single patch for each cluster, and we kill the half of the clusters with low *thing* likelihood. We repeat this procedure iteratively, doubling the number of verifications before killing the worst half in terms of *thing* likelihood (keeping at least 1,000 at the end). We end when we have run 31 verifications per surviving element. To choose the elements to kill, one approach is to kill those with the lowest score. However, this can lead to many duplicates and poor coverage. Instead, we use a greedy selection procedure reminiscent of [34] Specifically, given a selection χ of clusters, let $s^{\chi}_{i,j}$ be the log likelihood ratio for the j'th highest-scoring patch in image i out of all clusters contained in χ. We greedily select clusters to include in χ to maximize $\sum_{i,j} 2^{-j} s^{\chi}_{i,j}$, i.e. exponentially discounting the scores of patches from the same image.

4.2 Qualitative Results: Object Discovery on PASCAL VOC 2011

Next, we turn to the full PASCAL VOC 2011 (Train+Val) dataset, which contains more than 11,000 images. We are aware of no other unsupervised object discovery algorithm which can handle the full PASCAL VOC dataset without labels or subsampling. Figure 7 shows some of our qualitative results. In the left column, we show our automatically-generated rank for the discovered object. Center, we show the initialization for each of these clusters: the top 6 patches retrieved using Exemplar LDA. Right we show the top 6 regions after verification. The masks visualize which HOG cells the algorithm believes contain the object: specifically, we map our Bayesian confidence scores $\beta * c_{x,y}$ (See Appendix A) to their locations in the image. Black borders indicate either the edge of the image, or the display cut off. Figure 8 shows a few examples of discovered regions that correspond to

Fig. 7. Examples of regions discovered in PASCAL VOC 2011. Left: object rank. Center: initial top 6 patches from Exemplar LDA. Right: top 6 verifications. See text for details.

Fig. 8. Typical failure cases from our algorithm. These "objects" appear in our final top 50. Most likely this is a failure of the background model, which does not produce a high enough likelihood for these particular self-similar textures.

self-similar textures rather than objects, which is the most common failure mode of our algorithm. *Implementation Details:* To rank the discovered objects, we use the same procedure as in Section 4.1 applied to the full PASCAL VOC 2011 to discover the top 1,000 clusters. To make a better visualization, we also perform an additional de-duplication step following [16]. Our full ranking is available online at: `http://graphics.cs.cmu.edu/projects/contextPrediction/`.

4.3 Keypoint annotation transfer

Finally, we demonstrate the quality of our discovered intra-cluster correspondences by applying our method to the problem of keypoint annotation using the car annotations on PASCAL 2011 from [35]. The goal is to predict keypoint locations in an unlabeled image using other labeled images, in our case, without knowing it is a car. To perform transfer, we begin with the 1,000 objects discovered from PASCAL VOC 2011 above. For a given test image, we first use Exemplar-LDA to find which of the 1,000 clusters fire in this image. For each detection, we perform our context-based verification. Each verification uses the top 20 predictor images according to verification score, and we transfer keypoints from all 20 images using f. We make the assumption that each keypoint occurs only once per instance, so we score each keypoint and take, for each label, the keypoint with maximum score. We compute this score as $c_{\mathcal{C}[t]} * w^i_{\mathcal{C}[t],u,v} * s$, where the points (x, y) and (u, v) are the points that the keypoint was transferred to and from, respectively, s is the overall verification probability (likelihood of *thing* over likelihood of *thing* + likelihood of *stuff*) for the patch, and $c_{\mathcal{C}[t],x,y}$ is the per-point confidence computed in Appendix A. If multiple verifications happen for the same image (i.e. multiple patches are detected) and the regions considered to be *thing* overlap (intersection over union greater than .2), then the keypoints predicted for those verifications are merged and de-duplicated so that there is only one prediction per label.

Unfortunately, the evaluation of [35] isn't suitable in our situation, since they only measure keypoint accuracy conditioned on correctly-predicted bounding boxes. Since our algorithm may discover partial correspondences (e.g. only the wheel), we wish to evaluate on keypoints directly. We consider a keypoint prediction "correct" if there is a ground-truth keypoint of the same label within a distance less than 10% of the max dimension of the associated bounding box. We penalize repeated keypoint detections. Each predicted keypoint has a confidence score, so we can compute a precision-recall curve. For reasons of computation time, we only evaluate on images containing at least one car that's larger than 150 pixels on its minimum dimension (476 images total), and predict keypoints in a leave-one-out paradigm. Note that while we measure performance only on

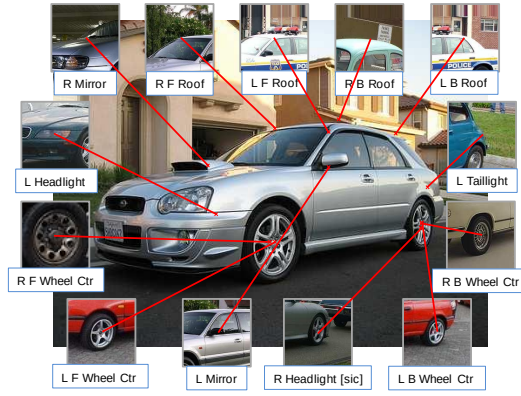

Fig. 9. An example of keypoint transfer. For each predicted keypoint, we show a patch from the image where that keypoint was transferred from. These correspondences are discovered without any supervision. Note that multiple different labels are predicted for the wheel centers. This happens because our algorithm finds all wheels somewhat visually similar, and proposes as many correspondences as possible.

these images, we allow our algorithm to find correspondence in all images, even those containing no cars. Labels were never used to find or score correspondences (though we assume at most one instance of each keypoint label per object, which helps us de-duplicate). Figure 6 shows our precision-recall curve. Admittedly our recall is low, but note that out of 1,000 clusters, few correspond to cars (on the order of 50). For comparison, we include a baseline that uses Exemplar-LDA directly, explained in Appendix D. Chance performance is well below 1% precision. Figure 9 shows the raw output for one image, after thresholding the confidence of keypoint transfers.

5 Conclusion

In this work, we have presented a method for validating whether a cluster of patches depicts a coherent visual concept, by measuring whether the correspondence provided by that cluster helps to predict the context around each patch. However, many questions remain about the best ways to implement and use the prediction models presented here. For instance, can we predict color in a prediction region conditioned on the color of the condition region? If so, color may become an important cue in detecting these regions. Texture and even brightness information may be similarly useful. Another extension may be to treat $P(H^0_{\mathcal{C}[t]}|H^0_{\mathcal{C}[1:t-1]}, H^0_{\mathcal{P}})$ as a more classical machine learning problem: estimating conditional distributions is, after all, a classical *discriminative* learning setup. Our algorithm did not use standard discriminative learning algorithms due to a lack of reliable training data, but in a supervised setting our *thing* model might be replaced with a far simpler regression model trained discriminatively.

Acknowledgements: We thank Shiry Ginosar, Yong Jae Lee, Dinesh Jayaraman, and anonymous reviewers for suggesting improvements to the writing. This work was partially supported by an Amazon Web Services grant, a Google Research grant, and ONR MURI N000141010934. This research was conducted with Government support under and awarded by DoD, Air Force Office of Scientific Research, National Defense Science and Engineering Graduate (NDSEG) Fellowship, 32 CFR 168a.

References

1. Hinton, G.E., Dayan, P., Frey, B.J., Neal, R.M.: The "wake-sleep" algorithm for unsupervised neural networks. Proceedings. IEEE (1995)
2. Olshausen, B.A., et al.: Emergence of simple-cell receptive field properties by learning a sparse code for natural images. Nature (1996)
3. Le, Q.V.: Building high-level features using large scale unsupervised learning. In: ICASSP. (2013)
4. Fergus, R., Perona, P., Zisserman, A.: Object class recognition by unsupervised scale-invariant learning. In: CVPR. (2003)
5. Sivic, J., Russell, B.C., Efros, A.A., Zisserman, A., Freeman, W.T.: Discovering objects and their location in images. In: ICCV. (2005)
6. Russell, B.C., Freeman, W.T., Efros, A.A., Sivic, J., Zisserman, A.: Using multiple segmentations to discover objects and their extent in image collections. In: CVPR. (2006)
7. Lee, Y.J., Grauman, K.: Foreground focus: Unsupervised learning from partially matching images. IJCV (2009)
8. Payet, N., Todorovic, S.: From a set of shapes to object discovery. In: ECCV. (2010)
9. Kim, G., Faloutsos, C., Hebert, M.: Unsupervised modeling of object categories using link analysis techniques. In: CVPR. (2008)
10. Grauman, K., Darrell, T.: Unsupervised learning of categories from sets of partially matching image features. In: CVPR. (2006)
11. Faktor, A., Irani, M.: clustering by composition–unsupervised discovery of image categories. In: ECCV 2012. (2012)
12. Rubinstein, M., Joulin, A., Kopf, J., Liu, C.: Unsupervised joint object discovery and segmentation in internet images. In: CVPR. (2013)
13. Torralba, A., Efros, A.A.: Unbiased look at dataset bias. In: CVPR. (2011)
14. Doersch, C., Gupta, A., Efros, A.A.: Mid-level visual element discovery as discriminative mode seeking. In: NIPS. (2013)
15. Singh, S., Gupta, A., Efros, A.A.: Unsupervised discovery of mid-level discriminative patches. In: ECCV. (2012)
16. Doersch, C., Singh, S., Gupta, A., Sivic, J., Efros, A.A.: What makes Paris look like Paris? SIGGRAPH (2012)
17. Endres, I., Shih, K., Jiaa, J., Hoiem, D.: Learning collections of part models for object recognition. In: CVPR. (2013)
18. Jain, A., Gupta, A., Rodriguez, M., Davis, L.: Representing videos using mid-level discriminative patches. In: CVPR. (2013)
19. Juneja, M., Vedaldi, A., Jawahar, C.V., Zisserman, A.: Blocks that shout: Distinctive parts for scene classification. In: CVPR. (2013)
20. Li, Q., Wu, J., Tu, Z.: Harvesting mid-level visual concepts from large-scale internet images. In: CVPR. (2013)
21. Sun, J., Ponce, J.: Learning discriminative part detectors for image classification and cosegmentation. In: ICCV. (2013)
22. Wang, X., Wang, B., Bai, X., Liu, W., Tu, Z.: Max-margin multiple-instance dictionary learning. In: ICML. (2013)
23. Zeiler, M.D., Fergus, R.: Visualizing and understanding convolutional neural networks. arXiv preprint arXiv:1311.2901 (2013)
24. Oliva, A., Torralba, A.: The role of context in object recognition. Trends in cognitive sciences (2007)

25. Weber, M., Welling, M., Perona, P.: Unsupervised learning of models for recognition. In: CVPR. (2000)
26. Adelson, E.H.: On seeing stuff: the perception of materials by humans and machines. In: Photonics West 2001-Electronic Imaging, International Society for Optics and Photonics (2001)
27. Efros, A.A., Leung, T.K.: Texture synthesis by non-parametric sampling. In: ICCV. (1999)
28. Munroe, R.: xkcd, a webcomic of romance, sarcasm, math and language. Creative Commons Attribution-Noncommercial (2014)
29. Hariharan, B., Malik, J., Ramanan, D.: Discriminative decorrelation for clustering and classification. In: ECCV. (2012)
30. Fei-Fei, L., Fergus, R., Perona, P.: Learning generative visual models from few training examples: An incremental bayesian approach tested on 101 object categories. CVIU (2007)
31. Wang, G., Zhang, Y., Fei-Fei, L.: Using dependent regions for object categorization in a generative framework. In: CVPR. (2006)
32. Sudderth, E.B., Torralba, A., Freeman, W.T., Willsky, A.S.: Learning hierarchical models of scenes, objects, and parts. In: ICCV. (2005)
33. Vondrick, C., Khosla, A., Malisiewicz, T., Torralba, A.: HOG-gles: Visualizing object detection features. In: ICCV. (2013)
34. Song, H.O., Girshick, R., Jegelka, S., Mairal, J., Harchaoui, Z., Darrell, T.: On learning to localize objects with minimal supervision. In: ICML. (2014)
35. Hejrati, M., Ramanan, D.: Analyzing 3d objects in cluttered images. In: NIPS. (2012)

A Computing the warping f

The goal in this section is to optimize the μ and Σ in equation 4. Recall that, for each location (x, y) in the condition region of the image, we have an associated Gaussian distribution over possible correspondences in image i (Equation 4), which we parameterize by $\mu^i_{x,y}$ and $\Sigma^i_{x,y}$. We minimize with respect to Σ and μ:

$$E(\Sigma, \mu) = \sum_{i,x,y} c_{x,y} \Phi(H^0_{x,y}, \mu^i_{x,y}, \Sigma^i_{x,y}) + \lambda \sum_{i,x,y} \sum_{(x',y') \in N(x,y)} \Psi(\mu^i_{x,y}, \Sigma^i_{x,y}, \mu^i_{x',y'}, \Sigma^i_{x',y'}) \tag{7}$$

Here, Φ rewards $\mu^i_{x,y}$ and $\Sigma^i_{x,y}$ for mapping $H^0_{x,y}$ to similar HOG cells in image i. Ψ encourages adjacent cells in the query image to map to adjacent cells in image i (i.e. the mapping should be smooth). $N(x, y)$ denotes the neighbors above, below, left, and right of (x, y). $c_{x,y}$ captures the probability that a given point (x, y) is a part of the object. That is, we penalize feature mismatches more in regions that are likely to contain the object.

A.1 Computing $c_{x,y}$: the probability that (x, y) is *thing*

We already have *thing* and *stuff* likelihoods computed for cells in the condition region, and a probability computed as $p^T_{x,y}/(p^T_{x,y} + p^S_{x,y})$ is exactly the sort of weighting we want to use for $c_{x,y}$. However, p^T and p^S tend to be quite noisy for individual cells, so we smooth them. Mathematically, we use Bayes rule to integrate the per-cell likelihoods across a small region, thereby estimating the posterior probability that (x, y) is *thing*. We compute the likelihoods for each model as follows:

$$L^T_{x,y} = \prod_{(u,v) \in \{\mathcal{C}[1:t-1], \mathcal{P}\}} \left(\hat{p}^T_{u,v}\right)^{\rho([u,v]-[x,y])} \quad ; \quad L^S_{x,y} = \prod_{(u,v) \in \{\mathcal{C}[1:t-1], \mathcal{P}\}} \left(p^S_{u,v}\right)^{\rho([u,v]-[x,y])} \tag{8}$$

We then compute $c_{x,y} = L^T_{x,y}/(L^T_{x,y} + L^S_{x,y})$. $\hat{p}^T$ and $\hat{p}^S$ are, respectively, p^T and p^S without allowing mimicry, and $\rho(v)$ weights the samples in the condition region such that the likelihoods of nearby points matter more. In our implementation, we set $\rho(v) \propto \mathcal{N}(v; 0, \sigma)$, a Gaussian weighting (isotropic, with mean 0 and variance σ), normalized such that $\sum_{(u,v) \in \{\mathcal{C}[1:t-1], \mathcal{P}\}} \rho([u, v] - [x, y]) = 1$.

A.2 Computing Φ: the unary potentials

Intuitively, our definition of the unary potentials Φ is that we try to use the query image to explain as much of the the predictor images as possible. This means that, whenever a feature (x, y) in the query image matches to multiple features in image i, $\Sigma^i_{x,y}$ will grow to explain as much as it can. Mathematically:

$$\Phi(H^0_{x,y}, \mu^i_{x,y}, \Sigma^i_{x,y}) = \\ -\sum_{u,v} \log \left[\mathcal{N}((u, v); \mu^i_{x,y}, \Sigma^i_{x,y}) * \mathcal{N}(H^i_{u,v}; H^0_{x,y}, \Sigma_H) + \gamma \mathcal{N}(H^i_{u,v}; \mu_H, \Sigma_H)\right] \tag{9}$$

Here, $\mathcal{N}(\cdot; \mu, \mathbf{\Sigma})$ represents a multivariate normal PDF with mean μ and variance $\mathbf{\Sigma}$, γ is a regularization constant (set to 100 in our experiments), and μ_H, Σ_H are the empirical mean and covariance of 31-dimensional HOG cell feature vectors across the full dataset. Intuitively this is a mixture model to explain all HOG cells in image i, where as much data as possible near $\mu^i_{x,y}$ is explained by a Gaussian in HOG space with mean $H^0_{x,y}$. The term involving μ_H provides a background distribution over HOG cells that can explain any HOG cell somewhat well, and prevents cells in image i that have no good matches in the query image from dominating the optimization.

A.3 Computing Ψ: the pairwise potentials

The pairwise potentials Ψ enforce smoothness of the correspondence. If two neighboring HOG cells are offset by some displacement in the query image, then the gaussians representing their correspondences in image i should be offset by the same displacement. We set:

$$\Psi(\mu^i_{x,y}, \Sigma^i_{x,y}, \mu^i_{x',y'}, \Sigma^i_{x',y'}) = \mathcal{D}\left(\mathcal{N}(\cdot, \mu^i_{x,y}, \Sigma^i_{x,y}), \mathcal{N}(\cdot, \mu^i_{x',y'} - [(x',y') - (x,y)], \Sigma^i_{x',y'})\right) \tag{10}$$

where $\mathcal{D}$ is the KL-divergence between the two distributions.

A.4 Improving affine invariance: pairwise potentials $\Omega_{x,y}$

The definition given in Section A.3 often works well, but it can fail if there is a large change in scale, rotation, or pose between our objects. Hence, we wish to give less penalty to warps that are locally affine, even if the affine transformation is large. To accomplish this, we reduce the value of λ (but don't eliminate it entirely, since we don't want to allow arbitrarily large affine transformations), and define a new term which directly penalizes the departure from an affine transformation at each location in f. We define a local affine transformation $\alpha^i_{x,y}$ at each HOG cell in the query image ($\alpha^i_{x,y}$ is represented with a 2-by-2 affine transformation matrix). Then we minimize:

$$E'(\Sigma, \mu, \alpha) = E(\Sigma, \mu) + \lambda' \sum_{x,y,i} \Omega_{x,y,i}(\Sigma^i, \mu^i, \alpha^i_{x,y}) \tag{11}$$

In this equation:

$$\Omega_{x,y,i}(\Sigma, \mu, \alpha_{x,y}) = \sum_{[(x',y'),(x'',y'')] \in L^H(x,y)} \mathcal{D}(\mathcal{N}(\cdot; \mu^i_{x',y'}, \Sigma^i_{x',y'}), \mathcal{N}(\cdot; \mu^i_{x'',y''} - [(x'',y'') - (x',y')]\, \alpha^i_{x,y}, \Sigma^i_{x'',y''}) \tag{12}$$

Here, $L(x,y)$ represents the edges of a 5-by-5 lattice centered at (x,y) (where each lattice point represents a HOG cell in the query image). $L(x,y)$ defines the

region in the query image over which the local affine transformation $\alpha_{x,y}$ applies. In defining $L(x, y)$, we make edges directed and include edges that are symmetries of each other; i.e. $L(3, 3)$ contains both $[(1, 1), (1, 2)]$ and $[(1, 2), (1, 1)]$; $|L(x, y)|$ will thus contain 80 different edges. Note that each term in the sum over grid edges is essentially the same penalty as Ψ, except that we specify that the displacement between the Gaussians should be defined by the affine transformation. While this may seem like a large number of terms in the sum, in practice we find that the time for optimization is still dominated by Φ, so we get this approximate affine invariance essentially for free.

A.5 Optimization

We optimize E' using generalized EM and coordinate descent. The E-step computes a weighting for cells in each of the predictor images, using Equation 9. That is, for each cell in the predictor images, we compute the likelihood that the correspondence to the query image is responsible for explaining that cell. This weight is computed as:

$$\zeta^i_{x,y,u,v} = \frac{\mathcal{N}((u, v); \mu^i_{x,y}, \Sigma^i_{x,y}) * \mathcal{N}(H^i_{u,v}; H^0_{u,v}, \Sigma_H)}{\mathcal{N}((u, v); \mu^i_{x,y}, \Sigma^i_{x,y}) * \mathcal{N}(H^i_{u,v}; H^0_{u,v}, \Sigma_H) + \gamma\mathcal{N}(H^i_{u,v}; \mu_H, \Sigma_H)} \quad (13)$$

All other variables are updated in the M-step. We can minimize the objective with respect to a single $\mu^i_{x,y}$ or $\alpha^i_{x,y}$ (keeping all other variables fixed), in closed form; the contribution of these variables to the overall objective is quadratic. Hence, it is convenient to use coordinate descent for the M-step, where each descent optimizes a single $\mu^i_{x,y}$ or $\alpha^i_{x,y}$. For each update of $\mu^i_{x,y}$, we also make an update to $\Sigma^i_{x,y}$ according to ordinary gradient desecent.

Updating $\alpha_{x,y}$ The KL divergence between two Gaussians may be written:

$$\frac{1}{2}\left(\mathrm{tr}\left(\Sigma_1^{-1}\Sigma_0\right) + (\mu_1 - \mu_0)\,\Sigma_1^{-1}(\mu_1 - \mu_0)^\top - k - \ln\left(\frac{\det\Sigma_0}{\det\Sigma_1}\right)\right). \quad (14)$$

Note that all of the edges summed over in Equation 12 are either vertical or horizontal, so we may optimize the rows of α separately, in each case focusing either on vertical or horizontal edges. The only term in the KL divergence that matters is the one involving μ. Hence, we can write the objective over the first row of α as:

$$\frac{\lambda'}{2} \sum_{[(x',y'),(x'',y'')]\in L^H(x,y)} \left(\mu^i_{x',y'} - \mu^i_{x'',y''} - \alpha^1_{x,y}\right)\left((\Sigma^i_{x',y'})^{-1} + (\Sigma^i_{x'',y''})^{-1}\right)\left(\mu^i_{x',y'} - \mu^i_{x'',y''} - \alpha^1_{x,y}\right)^\top \quad (15)$$

This is a standard quadratic form which we can minimize. Here, L^H includes only increasing horizontal edges (i.e., edges of the form $[(x, y), (x + 1, y)]$).

Updating $\mu_{x,y}$ For a fixed covariance matrix, the unary term can be computed as (up to a constant offset):

$$c_{x,y} \sum_{u,v} \zeta^i_{x,y,u,v} \left(\mu^i_{x,y} - (u,v)\right) \left(\Sigma^i_{x,y}\right)^{-1} \left(\mu^i_{x,y} - (u,v)\right)^\top \tag{16}$$

For the higher-order terms, we again only care about the part of the KL divergence that depends on μ. It can be computed as:

$$\frac{1}{2} \sum_{x',y' \in N(x,y)} \left[\lambda \Xi(I) + \sum_{x'',y'' \in L^{-1}([(x,y),(x',y')])} \lambda' \Xi(\alpha_{x'',y''}) \right] \tag{17}$$

Where

$$\Xi(\alpha) = \left((\mu^i_{x,y} - \mu^i_{x',y'}) - \alpha\left[(x,y) - (x',y')\right]\right) * \left((\Sigma^i_{x',y'})^{-1} + \left(\Sigma^i_{x'',y''}\right)^{-1}\right) * \left((\mu^i_{x,y} - \mu^i_{x',y'}) - \alpha\left[(x,y) - (x',y')\right]\right)^\top \tag{18}$$

$L^{-1}([(x,y),(x',y')])$ is the set of all points where $[(x,y),(x',y')] \in L(x'',y'')$. This is likewise a quadratic form which we minimize.

Updating $\Sigma_{x,y}$ Unfortunately, the expression for Σ keeping all other variables constant does not result in an expression that we are aware can be solved in closed form. Hence, we compute its gradient here. The gradient for the unary term wrt. $\Sigma^i_{x,y}$ is:

$$c_{x,y} \sum_{u,v} \zeta^i_{x,y,u,v} - \Sigma^{-1}_{x,y} + \left(\Sigma^i_{x,y}\right)^{-1} (u,v)^\top (u,v) \left(\Sigma^i_{x,y}\right)^{-1} \tag{19}$$

We add to this the gradient with respect to the higher order terms. We'll start with the terms in the KL divergence (Equation 14) that don't involve μ:

$$b \sum_{(x',y') \in N(x,y)} \frac{1}{2} \left(\mathrm{tr}\left((\Sigma^i_{x,y})^{-1} \Sigma^i_{x',y'} + (\Sigma^i_{x',y'})^{-1} \Sigma^i_{x,y}\right) - 2k\right). \tag{20}$$

Here, b counts the number of terms where such a KL divergence occurs that includes $\Sigma^i_{x,y}$, i.e. $b = \lambda + \lambda' |L^{-1}([(x,y),(x',y')])|$. Note that the final term from Equation 14 cancels when symmetrizing the KL-divergence. The gradient of this is:

$$\frac{b}{2} \sum_{(x',y') \in N(x,y)} (\Sigma^i_{x,y})^{-1} \Sigma^i_{x',y'} (\Sigma^i_{x,y})^{-1} + (\Sigma^i_{x',y'})^{-1} \tag{21}$$

Finally, the term involving μ in Equation 14 is essentially the same as Equation 17, except that the $\Sigma_{x'',y''}$ can be dropped since it is a constant that can be factored out of the rest. The gradient is:

$$\frac{1}{2} \sum_{x',y' \in N(x,y)} \left[\lambda \Xi'(I) + \sum_{x'',y'' \in L^{-1}([(x,y),(x',y')])} \lambda' \Xi'(\alpha_{x'',y''}) \right] \tag{22}$$

Where:

$$\begin{aligned}\Xi'(\alpha) = \left(\Sigma^i_{x,y}\right)^{-1} * \left((\mu^i_{x,y} - \mu^i_{x',y'}) - \alpha\left[(x,y) - (x',y')\right]\right)^\top \\ * \left((\mu^i_{x,y} - \mu^i_{x',y'}) - \alpha\left[(x,y) - (x',y')\right]\right) * \left(\Sigma^i_{x,y}\right)^{-1}\end{aligned} \tag{23}$$

B Computing $\omega^i_{\mathcal{C}[t]}$

Intuitively, our use of $\omega^i_{\mathcal{C}[t]}$ in Equation 6 is similar treating each predictor image as an "expert" in a mixture-of-experts paradigm. Of course, a mixture-of-experts paradigm assumes that we can force each expert i to produce a prediction separately, so that we can give greater weight to those experts that predict well. We obtain these per-expert predictions in a manner similar to how we obtain predictions for the full model: i.e., we estimate a likelihood $P^i_T(H^0_{\mathcal{C}[\tau]}|H^0_{\mathcal{C}[1:\tau-1]}, H^0_{\mathcal{P}})$ (which we will abbreviate as $g^i_{\mathcal{C}[\tau]}$) for all $\tau < t$, using only data from image i. This single image, however, will contain too little data to do a good job estimating the covariance of each conditional gaussian. Thus, we set:

$$g^i_{\mathcal{C}[\tau]} = P^i_T(H^0_{\mathcal{C}[\tau]}|H^0_{\mathcal{C}[1:\tau-1]}, H^0_{\mathcal{P}}) = \mathcal{N}(H^0_{\mathcal{C}[\tau]}; \theta^i_{\mathcal{C}[\tau]}, \Sigma^f_{\mathcal{C}[\tau]}) \tag{24}$$

Where

$$\theta^i_{\mathcal{C}[\tau]} = \sum_{u,v} w^i_{\mathcal{C}[\tau],u,v} H^i_{u,v} \tag{25}$$

Note the similarities between the prediction made by one expert and the prediction made by the full *thing* model. $\Sigma^f_{\mathcal{C}[\tau]}$ is actually identical to Equation 5, i.e. it is estimated using data from all images. $\theta^i_{\mathcal{C}[\tau]}$ is identical to $\mu^f_{\mathcal{C}[\tau]}$ in Equation 5 except that it integrates data from only a single image i. Given these per-image predictions $g^i_{\mathcal{C}[\tau]}$, we assume that any expert which predicted well for cells near to $\mathcal{C}[t]$ will also do a good job predicting the cell $\mathcal{C}[t]$. We use Bayes rule to estimate the probability of each expert given the data: i.e. set $\omega^i_{\mathcal{C}[t]}$ equal to the posterior probability of image i versus the other images. Under a uniform prior, the posterior is simply the re-normalized likelihood of the data under the different models. To compute the likelihoods for each model at location $\mathcal{C}[t] = (x_t, y_t)$, we compute the following product of data likelihoods:

$$\omega^i_{\mathcal{C}[t]} \propto \prod_{(x,y)\in\{\mathcal{C}[1:t-1],\mathcal{P}\}} \left(g^i_{(x,y)}\right)^{\rho([x,y]-[x_t,y_t])} \tag{26}$$

Here, ρ is the same as in Equation 8. We normalize $\omega^i_{\mathcal{C}[t]}$ such that $\sum_i \omega^i_{\mathcal{C}[t]} = 1$ (since our goal is to compute a relative weighting between images), and we cap $\omega^i_{\mathcal{C}[t]}$ at $1/3$ to prevent any single image from dominating the prediction. When deciding which element to hold out and predict during our object discovery pipeline, compute the "usage score" of a given patch i as $\sum_t \omega^i_{\mathcal{C}[t]} * c_{\mathcal{C}[t]}$.

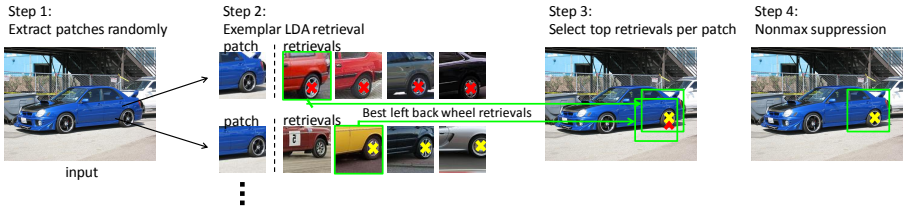

Fig. 10. Intuition behind our baseline keypoint prediction algorithm. See text for details.

C Computing when to mimic the *stuff* model

We have two criteria to determine when the *thing* model mimics the *stuff* model: (1) We measure how well the *thing* model has predicted the query image in cells near the current prediction region, and (2) we measure whether the *thing* model believes $\mathcal{C}[t]$ corresponds to cells in other images that were predicted poorly. For (1), recall that $c_{x,y}$ (Appendix A.1) is the Bayesian estimate of the probability that (x, y) is a part of the *thing*; this can be used without modification. To compute (2), assuming we have already performed prediction on another image i, let $\pi^i_{u,v}$ be the probability of the *thing* model for image at location (u, v) in that image (i.e., $\hat{p}^T_{u,v}/(\hat{p}^T_{u,v} + \hat{p}^S_{u,v})$ computed while predicting image i, without allowing mimicry). We can integrate these values as

$$\beta_{\mathcal{C}[t]} = \sum_i \hat{\omega}^i_{\mathcal{C}[t]} \sum_{u,v} w^i_{\mathcal{C}[t],u,v} \pi^i_{u,v} \tag{27}$$

Here, the sum over i sums over only those images i where prediction has already been computed; $w^i_{\mathcal{C}[t],u,v}$ is defined as in equation 6, and $\hat{\omega}^i_{\mathcal{C}[t]} \propto \omega^i_{\mathcal{C}[t]}$ as defined in equation 6 but re-normalized to sum to 1 over the smaller set of images. We mimic if $\beta_{\mathcal{C}[t]} * c_{\mathcal{C}[t]} < .3$, a threshold we determined empirically. If we haven't yet performed verification on any other images, we simply set $\beta_{\mathcal{C}[t]} = .5$.

D Baseline algorithm for unsupervised keypoint prediction

We are aware of no previous work which is designed for intra-category keypoint prediction without any knowledge of object class labels, so we have developed our own. This is a non-trivial mining task, and we have found that even performing above chance involves considerable design complexity. Mid-level visual elements (trained using exemplar LDA) are one simple and well-understood way to obtain an initial correspondence, and we have found that this approach can be extended for keypoint prediction. The intuition is shown in Figure 10. In step 1, we sample 50 random patches from the image where we want to make predictions. Some of these patches will hopefully contain the pixels that correspond to each ground-truth keypoint that needs to be predicted (in the figure, we show two sampled patches containing the left back wheel). Next, we want to predict keypoints within each sampled patch. To do this, in step 2, we retrieve matching

patches for each sampled patch using exemplar LDA, and in step 3, we select the top keypoints for each label out of this retrieved set. That is, at the end of step 3, we have at most one instance of each keypoint label per sampled patch. Each keypoint is scored according to the LDA score of the retrieved patch. In Figure 10, for example, we retrieve several other patches containing left back wheels. Note that we retrieve one patch containing a front wheel rather than a rear wheel, which will result in a front wheel prediction for this patch (which we don't show in the figure). Finally, in step 4, we perform non-maximum suppression across the sampled patches. For each keypoint label, we go through all patches containing a prediction for that keypoint, in order of LDA-based score for the keypoint. Each time we find an instance of the label of interest in a patch p, we suppress all keypoints in lower-ranked patches that occur within the bounding box for p. In Figure 10, the red keypoint is less confident than the yellow keypoint, and occurs within the bounding box associated with the yellow keypoint, so it is suppressed. We continue selecting the top non-suppressed keypoints, until we run out. In practice this approach performs far above chance, but it often fails to match the more difficult keypoints (e.g., it matches wheels well, but harder keypoints like headlights or roofs must often be extrapolated from the wheels, and these will be missed). However, since this baseline algorithm samples a huge number of patches per image (far more than in the core algorithm in this paper), the baseline can achieve higher recall.

E Visualization of our algorithm progression

To provide additional intuition about our algorithm, Figure E shows our algorithm verifying a single correspondence.

(a) Initial correspondence. Left is the query image; right are 3 out of the 20 predictor images.

Round 1

(b) The state of the *thing* model after making predictions for one set of prediction regions (the HOG cells surrounding the initial patch). On the left, we show $\log(P^T_{C[t]}/P^S_{C[t]})$ (i.e. each square represents the likelihood ratio for one HOG cell). On the right we visualize the foreground model's estimated correspondence between the prediction image and the predictor images. (Note that the warping f actually allows uncertainty and is not defined between cells. Hence, we perform smoothing in this visualization, which can lead to a slight fisheye effect; see our online code for details).

Round 2

(c) The same state after another round of prediction. Note that certain structure is easy for the *thing* model to predict (the wheel, the corners of the roof frame, parts of the bumper) whereas other parts are difficult (notably the headlights, which can have a wide range of appearance across cars)

Round 12

(d) At round 12, the prediction algorithm reaches the back wheel. Note that up until the wheel, the *thing* model has been predicting roughly as well as the *stuff* model, because the horizontal texture is easy for both. However, the *stuff* model cannot predict the wheel.

Final

(e) After 24 rounds the algorithm terminates, because it the *thing* model no longer believes it can predict more of the image well. The predictions themselves are quite noisy, but summing over a large region, it is clear that the foreground model has predicted better than the background model across the car.

Fig. 11. A visualization of the progress of our algorithm while verifying a single correspondence.