Supervision Beyond Human Annotations for Learning Visual Representations

Carl Doersch

October 19, 2015

Machine Learning Department
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

Thesis Committee:
Alexei A. Efros, Co-chair
Abhinav Gupta, Co-chair
Ruslan Salakhutdinov
Trevor Darrell, University of California, Berkeley

Thesis proposal submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Machine Learning

©Carl Doersch, 2015
Abstract

Understanding an image involves recognizing objects, textures, and scene properties such that the scene contents can be linked with outside knowledge. While this type of recognition and association comes effortlessly to humans for most images, it has been extremely difficult to implement in machines. The current best approaches rely heavily on statistical machine learning. For example, in object recognition, one typically divides the set of all recognizable objects into categories, and then builds a classifier which maps thousands or millions of example objects to these categories. Each time a new example is labeled by a human annotator, the classifier is updated, if necessary, to give the correct label.

The study of supervised machine learning techniques in vision generally aims to maximize the accuracy of the resulting classifier on exactly the same task for which labeled examples are provided. However, is it possible that classifiers can, in some cases, learn to perform more than just the tasks where labels are available? In this thesis, we aim to images labeled with labels different than those we are interested in, or even those without labels to build a representation that can then be applied to different tasks. We show that these algorithms are not only possible, but they can work using labels that are much less expensive than the ones currently in use. Such algorithms also enable computers to tackle problems where the correct answer is ambiguous and hence impossible to annotate accurately.

To this end, we explore two broad classes of supervision. The first is weak image-level supervision, which we exploit to train mid-level discriminative patch classifiers. For example, given a dataset of imagery only labeled with GPS coordinates, we train patch classifiers to fire only within Paris images, and find that the resulting classifiers actually correspond to distinctive architectural elements of Paris. The second requires information about images other than the pixels themselves, and instead relies on regularities in image layout. Specifically, we train our algorithm to predict the context around a given patch, and demonstrate that this forces the algorithm to recognize objects and recall the layout of their parts.
Contents

1 Introduction and Summary 1

2 Related Work 6
   2.1 Weakly-Supervised Methods 7
   2.2 Unsupervised Methods 7

3 Weakly Supervised Discriminative Patch Discovery 10
   3.1 Introduction 10
   3.2 Mode Seeking on Density Ratios 11
   3.3 Optimizing the objective 14
   3.4 Better Adaptive Bandwidth via Inter-Element Communication 15
   3.5 Evaluation via Purity-Coverage Plot 17
   3.6 Scene Classification 20
   3.7 Conclusion 22

4 Object Discovery by Learning to Predict Context 23
   4.1 Introduction 23
   4.2 Overview 24
   4.3 Algorithm 26
   4.4 Results 36
   4.5 Baseline algorithm for unsupervised keypoint prediction 40
   4.6 Conclusion 41

5 Deep Visual Representation Learning by Unsupervised Context Prediction 42
   5.1 Introduction 42
   5.2 Learning Visual Context Prediction 44
   5.3 Experiments 47
   5.4 Proposed: Leveraging the 2D structure of objects 53

6 Proposed: From Image Classification to Image Generation 55

7 Conclusion 57
Chapter 1

Introduction and Summary

Given an image, humans can perform many tasks which remain difficult for computers. For example, the average person could find all the dogs, estimate the pose of humans, determine whether the image was taken indoors, or say exactly which pixels are on chairs, and in all cases far outperform machines. This is true despite a significant advantage that the machines have: training data which is given in exactly the same format as the testing data. For example, a machine trained to place bounding boxes around dogs is given hundreds of examples where humans have already placed boxes around all instances of dogs. Humans, on the other hand, learn from a noisy childhood experience. It is true that parents occasionally name objects and valuable experience can be gained through touch and other senses, there are certainly neither bounding boxes nor segmentations given. Inside the human brain there must be some representation of the visual world which is trained from whatever information is available during childhood, and yet is flexible enough to apply to the huge variety of visual tasks that humans may encounter.

Interestingly, work concurrent with this thesis has already suggested that vast performance improvements can be obtained by training a visual representation on a task different than one’s goal. For instance, the current state-of-the-art approach in the Pascal object detection challenge begins by ‘pre-training’ a deep neural network [65] to perform the 1000-way classification task posed by ImageNet [19]. The trained representation is then transferred to object detection, which involves re-training the network to classify whether a given image region depicts one of the 20 Pascal object categories [44]. It is perhaps not so surprising that the network can be retrained to classify the Pascal objects, since many Pascal objects are contained in ImageNet. However, it is perhaps surprising that the network can be readily trained to classify all regions that do not contain one of these categories as “background”. Hence, it seems the pretraining training has resulted in some form of general-purpose representation. Can this approach be extended? ImageNet contains 1.3 million images, each of which is labeled with one of 1,000 categories. Collecting it was non-trivial, and expanding it by an order of magnitude or two would be expensive. Hence, we seek a method to pre-train a representation using labels that are even less expensive than the ImageNet categorization challenge.

It turns out that the pre-training task does not necessarily need to be a useful one. In the text domain, for example, it is possible to train a model to predict the words in the context (i.e., a few words before and/or after) of a given word. The resulting representations are useful for other tasks involving language interpretation [3, 16, 80, 86]. Here the context
Figure 1.1: Given only a large image database tagged with GPS coordinates, our algorithm is able to automatically discover the geographically-informative elements (patch clusters to the right of each photo) that help in capturing its “look and feel”. On the left, the emblematic street sign, a balustrade window, and the balcony support are all very indicative of Paris, while on the right, the neoclassical columned entryway sporting a balcony, a Victorian window, and, of course, the cast iron railing are very much features of London.

The prediction task is just a “pretext” to force the model to learn a good representation. We argue that such “pretext” tasks exist for vision as well. The important characteristic of a pretraining task is that solving the task must require the algorithm to build a representation which is more generally useful. Naturally, higher-level tasks such as ImageNet classification could serve as a pretext task, since they require understanding objects. The goal of this thesis, however, is to show that far less expensive labels can be used to develop pretext tasks that lead to useful representations.

**Thesis Statement:** There exist pretext tasks for which labels are simple and inexpensive to obtain, yet can be used to train image representations that are useful for more than just solving the pretext task. We aim to find such tasks to demonstrate their potential for replacing existing, labor-intensive approaches for generating training data. We also aim to understand what information is ultimately captured in these representations, which we accomplish through visualization and through applying the representations to real-world tasks.

**Overview**

1.0.1 Weakly Supervised Discriminative Patch Discovery

We begin by examining weakly-supervised learning, where we assume that our training images have an associated label that is relatively inexpensive and not directly informative. A motivating example for this problem setup is the analysis of the styles of cities. It is straightforward to download large datasets of images taken in different cities; for example, we can easily obtain millions of such labeled images from Google Street View. Given this data, we could use standard approaches to directly train a classifier which can distinguish between cities. What is less obvious, however, is how to produce the sort of output that might be useful to an architect, urban historian, or graphics designer, which explains what in the image gives it the distinctive look and connects it to other scenes elsewhere in the city. An annotation like that shown in Figure 1.1 is far more useful for these purposes, but providing training examples is not merely labor-intensive, it is ambiguous. How large should the bounding boxes extend on the facade when the elements of the architecture tend to occur together? Does one window having a larger railing than another indicate a sub-category of window that should be clustered separately? These are questions that the
computer should be able to answer automatically.

We aim to group patches into visual elements that capture the important properties of the data (in the case of Paris, they capture style). These elements are 1) frequent, i.e., there are many instances throughout the city, and 2) discriminative, i.e., they tell us at a glance what city we are looking at. We can implement this idea by training a detector (i.e. a patch classifier) which fires only in one city, and yet fires on a visually visually coherent set of patches containing as many instances as possible from that city. Of course, building such a classifier requires training data, which is non-trivial to obtain when we do not begin with any bounding boxes for the elements.

**Visual Element Discovery as Discriminative Mode Seeking (Chapter 3):** This chapter details a formulation which can simultaneously train a classifier which captures a visual element and discovers the set of patches which can be used as training examples. We pose visual element discovery as discriminative mode seeking, drawing connections to the well-known and well-studied mean-shift algorithm. Given a weakly-labeled image collection, our method discovers visually-coherent patch clusters that are maximally discriminative with respect to the labels. One advantage of our formulation is that it requires only a single pass through the data. We also propose the Purity-Coverage plot as a principled way of experimentally analyzing and evaluating different visual discovery approaches, and demonstrate visual element discovery within a dataset of Street View images. We also evaluate our method on the task of scene classification, demonstrating strong performance on the MIT Scene-67 dataset.

**1.0.2 Unsupervised Learning**

Next, we turn to representation learning when no annotation is available beyond the pixels themselves. Unsupervised learning has proven extremely difficult in the visual domain because even images that look similar to humans may actually have vast differences at the pixel level. In the weakly-supervised geographic discovery algorithm above, we assumed that patches belong together only if they come from the same city, giving us some criterion to optimize. Without these labels, how can we tell whether a set of patches belongs together? The surprising answer is that the image context surrounding the patch can be treated as a sort of weak label. A set of patches belongs together if we can share information between them to predict the context surrounding each one. This, it turns out, can serve as the basis for converting the unsupervised learning problem into a supervised learning one, where the algorithm learns to predict the context given a patch.

**Object Discovery by Learning to Predict Context (Chapter 4):** In this chapter, 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 (i.e. generative) 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.
Deep Visual Representation Learning by Unsupervised Context Prediction (Chapter 5):
This chapter explores the use of spatial context as a source of free and plentiful supervisory signal for training a deep visual representation. Given only a large, unlabeled image collection, we extract random pairs of patches from each image in the collection and train a discriminative model to predict their relative position within the image. We argue that doing well on this task will require the model to learn about the layout of visual objects and object parts. We demonstrate that the feature representation learned using this within-image context prediction task is indeed able to capture visual similarity across images. For example, this representation allows us to perform unsupervised visual discovery of objects like cats, people, and even birds from the Pascal VOC 2011 detection dataset. Furthermore, we show that the learned features, when used as pre-training for the R-CNN object detection pipeline, provide a significant boost over random initialization on Pascal object detection, resulting in state-of-the-art performance among algorithms which use only Pascal-provided training set annotations.

Proposed: We have already shown that predicting the position of two patches relative to each other leads to a useful representation, but the representation cannot distinguish between objects, which have a specific layout of parts, and “stuff,” for which there are no specific parts. Hence, much of the representation seems to represent texture gradients and straight edges, rather than semantics. We propose to build a representation which infers and segments out objects, and then predicts patch locations relative to each other only within the object. We propose a formulation which can predict both patch locations and segmentations without labels for either, by using within-segment consistency as a cue for the correctness of both the segmentations and of the location predictions.

From Image Classification to Image Generation (Chapter 6): Much of the work in this thesis visualizes representations by clustering patches, but we have found that as representations become more invariant, producing coherent clusters becomes more difficult due to variation in nuisance factors like pose and lighting. Furthermore, in large datasets, nearest-neighbor search can become too expensive. In Chapter 4, we presented a generative model of images. While the main purpose of this model was to produce probability values, for the sake of statistical hypothesis testing for cluster membership, we also showed an example of generating a new car image from a set of input cars. This kind of visualization useful for understanding a visual representation than clustering, but we propose that it can be extended to be even more informative. For example, we can imagine examining Paris style by producing exaggerated Paris style in an image, or producing an image which is artificially free of factors inconsistent with Paris style. Such algorithms may also have graphics applications, since we might be able to produce a Paris image with a particular layout specified via a non-Paris photograph. However, many challenges remain in this domain, because it is not currently known how to generate realistic images with deep networks.

Proposed: We propose to build a model to generate images by first training several models from either weak supervision or no supervision. The unsupervised model, trained in a generative manner, will capture arbitrary correlations in the world, and aid in producing coherent global structure and realistic local image features. The supervised models, on the other hand, will be trained on tasks such as predicting style, scene categories, or object categories. These models can contribute to image generation because they can quantify the extent to which an image exhibits a given semantic trait. We can then force an image
to display more of a trait by backpropagating the signal from these models. This gives us control over the kind of images that the generative model creates, and also lets us visualize what visual patterns are most important for the trained models.
Chapter 2

Related Work

The idea of an image representation which exists between the input pixels and higher-level cognitive tasks has been a common theme in both computer vision and the study of physiological vision. David Marr, for instance, proposed a 2.5D sketch as a mid-level representation between low-level (e.g. edges and lines) and high-level (i.e. full 3D shape) representations of scenes [79], and Fukushima’s Neocognitron called for the perception of object parts before assembling them into full objects [41]. These theories are appealing because they let us explain the complex function of vision in terms of a series of simpler operations, and they allow us to re-use of computation (e.g. a ⊢ detector can be used to detect both a “B” and an “F”). However, the correct approach to computer implementations of these mid-level representations has remained a matter of controversy.

The bulk of recent work on this topic assumes that these representations should be learned from data rather than handcrafted. For instance, LeCun et al. [68] used discriminative learning (in their case, backpropagation) to build a multi-level hierarchical representation of handwritten digits, where the features were tuned specifically to aid in recognizing digit categories. Subsequent works aimed to provide more constraint on the mid-level representation, to improve statistical efficiently when learning the vast complexity of the visual world. Learning-based implementations of Pictorial Structures [33,37], for example, assume that the mid-level representation is a collection of parts, and that the higher-level reasoning is a relatively simple assembly of these parts with penalties for deformation. However, in the past few years, the dominant approaches have largely avoided explicit representation of parts [65,105]. These “Deep ConvNet” approaches, which are in many ways similar to the ConvNets of the 80s [68], are the current winners in many important challenges, including ImageNet classification [65] and PASCAL object detection [44]. Essential in making a high-performance representation, however, is the existence of a large, labeled dataset such as ImageNet [19]. The deep representations are first pre-trained to classify more than a million images into one of 1000 categories. In the case of Pascal object detection, for example, these representations are then re-trained to classify whether a bounding box depicts an object or background. The object detection dataset is smaller than the original classification dataset, but the representation draws on its previous experience with classification in order to generalize better. The effectiveness of pre-training on weaker labels like the image-level category annotations is promising, but raises the question: how weak can the labels be? Much work has focused on the kinds of supervision that might be effective for representation learning, with varying levels of success. Here, we divide these methods into two broad types: those
which assume some auxiliary information is provided alongside each image, which we call *weakly-supervised learning*, and those which assume no information is provided besides the pixels (which may include multiple video frames). We acknowledge, however, that the distinction might be somewhat artificial, since weak labels are often available without any human annotation effort, and pixels in the context of an image region might also be seen as a sort of weak label for that region.

### 2.1 Weakly-Supervised Methods

It is natural to ask whether algorithms that can classify well can also localize well, even when no localization labels are present. In the weakly-supervised localization task, the input is a set of images with image-level labels of whether an object category is present anywhere, and the goal is to produce either bounding-box detections for the object that is labeled, or even segmentations [11, 15, 85, 89, 103, 107, 123]. Other works on cosegmentation [9, 58, 62, 97] and colocalization [20, 59, 114] make even weaker assumptions about the annotations given, e.g., that they were returned from a web search and hence may not even contain the object of interest, or that negatives are not available. These works do, however, assume that the images correspond to categories and that these categories are well represented within the images (e.g. are often large and central in the image), which aids in the learning process.

Another approach is to turn to the web and collect any information that can be scraped along with images, and use them as labels. For example, text [5, 102], user tags [15, 54, 133], or GPS coordinates [64, 75, 92, 101] are all easily available and provide cues for some important aspects of images. When the labels are noisy or only provided for a subset of images, semi-supervised learning may be employed to propagate or correct the labels [27, 35, 71, 104]. Recent efforts have attempted to summarize visual information across the web into a database of concepts that can make sense of it altogether [8, 21].

The work in this thesis is part of yet another line of research on weak supervision that focuses on discriminative patch mining [22, 25, 29, 60, 106], which has emphasized weak supervision as a means of object discovery. These methods generally only assume that the objects of interest are only somewhat correlated with the labels, and do not assume that they are ever large in the images. They have also emphasized the utility of learning representations of patches (i.e. object parts) before learning full objects and scenes. For example, our work in [25] trains detectors to fire only on images from particular geographic locations, but the actual goal that they achieve by this is the discovery of much more specific elements of urban architecture.

### 2.2 Unsupervised Methods

Approaches which rely solely on unlabeled images and videos have a long history in vision science, and not simply because they may greatly reduce labeling effort. Many of the earliest unsupervised approaches were motivated by biological vision, where the classic notion of a ‘label’—i.e. semantic categories—seems unrealistic for organisms that must rely solely on sense data. One of the most prominent lines of research is temporal coherence learning [38, 51, 53, 81, 122, 125, 129, 135]. Here, the idea is simply that objects do not appear and disappear randomly in video, nor does scene layout change rapidly. Even though the appearance may change as objects deform, lighting changes, and so on, many important scene properties will
change slowly. Hence, the visual representation should also change slowly between frames of video. This effect has been demonstrated in physiological experiments [73], [121], and recent work has shown that it can provide improvements on realistic image datasets when training data is limited [125]. Interestingly, many of these approaches avoid formulating a global objective function, and instead use learning rules that can be evaluated on the level of individual neurons.

Another way to think of a good image representation is as the latent variables of an appropriate generative model. An ideal generative model of natural images would both generate images according to their natural distribution, and be concise in the sense that it would seek common causes for different images and shares information between them. A key difficulty, however, is that inferring the latent structure given an image is intractable for even relatively simple models. To deal with these computational issues, a number of works, such as the wake-sleep algorithm [52], contrastive divergence [50], deep Boltzmann machines [100], and Bayesian nonparametric models [111] use sampling to perform approximate inference. Others strengthen the independence assumptions in the model [128]. More recently, variational Bayesian methods for approximate inference have been proposed [63], [96]. These methods have shown promising performance on smaller datasets such as handwritten digits [50], [52], [63], [96], [100], but none have proven effective for high-resolution natural images.

Another way to look at the goal of unsupervised representation learning is that it aims to learn an embedding (i.e. a feature vector for each image) where images that are semantically similar are close, while semantically different ones are far away. For instance, cars should be similar to other cars, red sports cars should be even more similar to other red sports cars, and so on. To build such a representation without supervision, one approach is to create a supervised “pretext” task, with the hope that this embedding will also be useful for real-world tasks. For example, denoising autoencoders [4], [119] use reconstruction from noisy data as a pretext task; to tell the difference between noise and signal, the algorithm must connect images to other images with similar objects, in order to remove the parts of the image that don’t fit the shared appearance patterns. Sparse autoencoders also use reconstruction as the pretext task, in conjunction with a sparsity penalty on the representation [88]. To learn deeper (and more non-linear) visual representations, one commonly used approach is to stack such sparse autoencoders [67], [69]. We are aware of only one such stacked sparse autoencoder model that has been applied to full-scale images [67]; while the results were promising, it required around a million CPU hours and reported only three discovered objects. Part of the problem with reconstruction-based algorithms is that many low-level phenomena, like stochastic textures, are surprisingly difficult to reconstruct accurately. This means that it’s often hard to even measure whether a model is generating images well.

A related pretext task is “context prediction.” A strong tradition for this kind of task already exists in the text domain, where “skip-gram” [80] models have been shown to generate useful word representations. The idea is to train a model (e.g. a deep network) to predict, from a single word, the $n$ preceding and $n$ succeeding words. In principle, similar reasoning could be applied in the image domain, a kind of visual “fill in the blank” task, but again one runs into the problem of determining whether the predictions are correct. To address this, [78] predicts the appearance of an image region by consensus voting of the transitive nearest neighbors of its surrounding regions. A number of approaches attempt to model contextual pixels directly [66], [115], but these models tend to focus on low-level textures. Hence, one of the chapters in this thesis attempts to simultaneously estimate how difficult the features are to predict, thereby focusing the algorithm on complex shapes rather
than simple textures [23], and another focuses on choosing between predictions rather than forming novel ones [24].

Another line of work in unsupervised learning from images aims to discover object categories using hand-crafted features and various forms of clustering (e.g. [98, 108] learned a generative model over bags of visual words). Such representations lose shape information, and will readily discover clusters of, say, foliage. A few subsequent works have attempted to use representations more closely tied to shape [70, 91], but relied on contour extraction, which is difficult in complex images. Many other approaches [31, 46, 61] focus on defining similarity metrics which can be used in more standard clustering algorithms; [95], for instance, re-casts the problem as frequent itemset mining. Geometry may also be used to for verifying links between images [12, 48, 92], although this can fail for deformable objects.
Chapter 3

Weakly Supervised Discriminative Patch Discovery

3.1 Introduction

A simple representation of visual information is a set of patch clusters, which are generally mined from a large database of images. To represent a novel image, each patch in the image simply be assigned to one of these clusters, and any learned information can be transferred. Perhaps the most popular implementation of this idea is “visual words” \([109]\), which are obtained by unsupervised clustering \((k\text{-means})\) of local features (SIFT) over a large dataset. However, “visual words” is a very low-level representation, mostly capturing local edges and corners (\([98]\) notes that “visual letters” or “visual phonemes” would have been a more accurate term). Part of the problem is that the local SIFT features are relatively low-dimensional \((128)\), and might not be powerful enough to capture anything of higher complexity. However, switching to a more descriptive feature (e.g., 2,000-dimensional HOG) causes \(k\)-means to produce visually poor clusters due to the curse of dimensionality \([25]\).

Several approaches \([25,29,56,60,74,106,113,127]\) have proposed mining visual data for discriminative mid-level visual elements, i.e., entities which are more informative than “visual words,” and more frequently occurring and easier to detect than high-level objects. Most such approaches require some form of weak per-image labels, e.g., scene categories \([60]\) or GPS coordinates \([25]\) (but can also run unsupervised \([106]\)), and have been recently used for tasks including image classification \([60,106,127]\), object detection \([29]\), visual data mining \([25,74]\), action recognition \([56]\), and geometry estimation \([39]\). But how are informative visual elements to be identified in the weakly-labeled visual dataset? The idea is to search for clusters of image patches that are both 1) representative, i.e., frequently occurring within the dataset, and 2) visually discriminative. Unfortunately, algorithms for finding patches that fit these criteria remain rather ad-hoc and poorly understood, and often do not even directly optimize these criteria. Hence, our goal in this work is to quantify the terms “representative” and “discriminative,” and show that a formulation which draws inspiration from the well-known, well-understood mean-shift algorithm can produce visual elements that are more representative and discriminative than those of previous approaches.

Mining visual elements from a large dataset is difficult for a number of reasons. First, the search space is huge: a typical dataset for visual data mining has tens of thousands of
images, and finding something in an image (e.g., finding matches for a visual template) involves searching across tens of thousands of patches at different positions and scales. To make matters worse, patch descriptors tend to be on the order of thousands of dimensions; not only is the curse of dimensionality a constant problem, but we must sift through terabytes of data. And we are searching for a needle in a haystack: the vast majority of patches are actually uninteresting, either because they are rare (e.g., they may contain multiple random things in a configuration that never occurs again) or they are redundant due to the overlapping nature of patches. This suggests the need for an online algorithm, because we wish to discard much of the data while making as few passes through the dataset as possible.

The well-known mean-shift algorithm [10, 17, 40] has been proposed to address many of these problems. The goal of mean-shift is to find the local maxima (modes) of a density using a sample from that density. Intuitively, mean-shift initializes each cluster centroid to a single data point, then iteratively 1) finds data points that are sufficiently similar to each centroid, and, 2) averages these data points to update the cluster centroid. In the end, each cluster generally depends on only a tiny fraction of the data, thus eliminating the need to keep the entire dataset in memory.

However, there is one issue with using classical mean-shift to solve our problem directly: it only finds local maxima of a single, unlabeled density, which may not be discriminative. But in our case, we can use the weak labels to divide our data into two different subsets ("positive" (+) and "negative" (−)) and seek visual elements which appear only in the "positive" set and not in the "negative" set. That is, we want to find points in feature space where the density of the positive set is large, and the density of the negative set is small. This can be achieved by maximizing the well-studied density ratio \( p_+/(x)/p_-(x) \) instead of maximizing the density. While a number of algorithms exist for estimating ratios of densities (see [142] for a review), we did not find any that were particularly suitable for finding local maxima of density ratios. Hence, the first contribution of our paper is to propose a discriminative variant of mean-shift for finding visual elements. Similar to the way mean-shift performs gradient ascent on a density estimate, our algorithm performs gradient ascent on the density ratio (section 3.2). When we perform gradient ascent separately for each element as in standard mean-shift, however, we find that the most frequently-occurring elements tend to be over-represented. Hence, section 3.4 describes a modification to our gradient ascent algorithm which uses inter-element communication to approximate common adaptive bandwidth procedures. Finally, in section 3.5 we demonstrate that our algorithms produce visual elements which are more representative and discriminative than previous methods, and in section 3.6 we show they significantly improve performance in scene classification.

### 3.2 Mode Seeking on Density Ratios

Our goal is to extract discriminative visual elements by finding the local maxima of the density ratio. However, one issue with performing gradient ascent directly on standard density ratio estimates is that common estimators tend to use a fixed kernel bandwidth, for example:

\[
\hat{r}(x) \propto \sum_{i=1}^{n} \theta_i K(||x - x_i||/h)
\]
Distance: 2.58 2.92 3.07 3.10 3.16
Distance: 1.01 1.13 1.13 1.15 1.17

Figure 3.1: The distribution of patches in HOG feature space is very non-uniform and absolute distances cannot be trusted. We show two patches with their 5 nearest-neighbors from the Paris Street View dataset [25]; beneath each nearest neighbor is its distance from query. Although the nearest neighbors on the left are visually much better, their distances are more than twice those on the right, meaning that the actual densities of the two regions will differ by a factor of more than $2^d$, where $d$ is the intrinsic dimensionality of patch feature space. Since this is a 2112-dimensional feature space, we estimate $d$ to be on the order of hundreds.

where $\hat{r}$ is the ratio estimate, the parameters $\theta_i \in \mathbb{R}$ are weights associated with each datapoint, $K$ is a kernel function (e.g., a Gaussian), and $h$ is a globally-shared bandwidth parameter. The bandwidth defines how much the density is smoothed before gradient ascent is performed, meaning these estimators assume a roughly equal distribution of points in all regions of the space. Unfortunately, absolute distances in HOG feature space cannot be trusted, as shown in Figure 3.1: any kernel bandwidth which is large enough to work well in the left example will be too large to work well in the right. One way to deal with the non-uniformity of the feature space is to use an adaptive bandwidth [18]: that is, different bandwidths are used in different regions of the space. However, previous algorithms are difficult to implement for large data in high-dimensional spaces; [18], for instance, requires a density estimate for every point used in computing the gradient of their objective, because their formulation relies on a per-point bandwidth rather than a per-cluster bandwidth. In our case, this is prohibitively expensive. While approximations exist [43], they rely on approximate nearest neighbor algorithms, which work for low-dimensional spaces ($\leq 48$ dimensions in [43]), but empirically we have found poor performance in HOG feature space (> 2000 dimensions). Hence, we take a different approach which we have tailored for density ratios.

We begin by using a result from [10] that classical mean-shift (using a flat kernel) is equivalent to finding the local maxima of the following density estimate:

$$
\sum_{i=1}^{n} \max(b - d(x_i, w), 0)
$$

(3.1)

In standard mean-shift, $d$ is the Euclidean distance function, $b$ is a constant that controls the kernel bandwidth, and $z(b)$ is a normalization constant. Here, the flat kernel has been replaced by its shadow kernel, the triangular kernel, using Theorem 1 from [10]. We want to maximize the density ratio, so we simply divide the two density estimates. We allow an adaptive bandwidth, but rather than associating a bandwidth with each datapoint, we compute it as a function of $w$ which depends on the data.

$$
\frac{\sum_{i=1}^{n_{pos}} \max(B(w) - d(x_i^+, w), 0)}{\sum_{i=1}^{n_{neg}} \max(B(w) - d(x_i^-, w), 0)}
$$

(3.2)

Where the normalization term $z(b)$ is cancelled. This expression, however, produces poor estimates of the ratio if the denominator is allowed to shrink to zero; in fact, it can produce arbitrarily large but spurious local maxima. Hence, we define $B(w)$ as the value of $b$ which satisfies:
\[
\sum_{i=1}^{n_{\text{neg}}} \max(b - d(x_i^-, w), 0) = \beta
\]  

(3.3)

Where \( \beta \) is a constant analogous to the bandwidth parameter, except that it directly controls how many negative datapoints are in each cluster. Note the value of the sum is strictly increasing in \( b \) when it is nonzero, so the \( b \) satisfying the constraint is unique. With this definition of \( B(w) \), we are actually fixing the value of the denominator of (3.2) (We include the denominator here only to make the ratio explicit, and we will drop it in later formula). This approach makes the implicit assumption that the distribution of the negatives captures the overall density of the patch space. Note that if we assume the denominator distribution is uniform, then \( B(w) \) becomes fixed and our objective is identical to fixed-bandwidth mean-shift.

Returning to our formulation, we must still choose the distance function \( d \). In high-dimensional feature space, [94] suggests that normalized correlation provides a better metric than the Euclidean distance commonly used in mean-shift. Formulations of mean-shift exist for data constrained to the unit sphere [7], but again we must adapt them to the ratio setting. Surprisingly, replacing the Euclidean distance with normalized correlation leads to a simpler optimization problem. First, we mean-subtract and normalize all datapoints \( x_i \) and rewrite (3.2) as:

\[
\sum_{i=1}^{n_{\text{pos}}} \max(w^\top x_i^+, b, 0) \ \text{s.t.} \ \sum_{i=1}^{n_{\text{neg}}} \max(w^\top x_i^-, b, 0) = \beta
\]

\[
\|w\|^2 = 1
\]

(3.4)

Where \( B(w) \) has been replaced by \( b \) as in equation (3.3), to emphasize that we can treat \( B(w) \) as a constraint in an optimization problem. We can further rewrite the above equation as finding the local maxima of:

\[
\sum_{i=1}^{n_{\text{pos}}} \max(w^\top x_i^+, b, 0) - \lambda \|w\|^2 \ \text{s.t.} \ \sum_{i=1}^{n_{\text{neg}}} \max(w^\top x_i^-, b, 0) = \beta
\]

(3.5)

Note that (3.5) is equivalent to (3.4) for some appropriate rescaling of \( \lambda \) and \( \beta \). It can be easily shown that multiplying \( \lambda \) by a constant factor does not change the relative location of local maxima, as long as we divide \( \beta \) by that same factor. Such a re-scaling will in fact result in re-scaling \( w \) by the same value, so we can choose a \( \lambda \) and \( \beta \) which makes the norm of \( w \) equal to 1.

After this rewriting, we are left with an objective that looks curiously like a margin-based method. Indeed, the negative set is treated very much like the negative set in an SVM (we penalize the linear sum of the margin violations), which follows [105]. However, unlike [106], which makes the ad-hoc choice of 5 positive examples, our algorithm allows each cluster to select the optimal number of positives based on the decision boundary. This is somewhat reminiscent of unsupervised margin-based clustering [77,131].

Mean-shift prescribes that we initialize the procedure outlined above at every datapoint. In our setting, however, this is not practical, so we instead use a randomly-sampled subset.

\footnote{Admittedly this means that the norm of \( w \) has an indirect effect on the underlying bandwidth: specifically if the norm of \( w \) is increased, it has a similar effect as a proportional decrease in \( \beta \) in (3.4). However, since \( w \) is roughly proportional to the density of the positive data, the bandwidth is only reduced when the density of positive data is high.}
We run this as an online algorithm by breaking the dataset into chunks and then mining, one chunk at a time, for patches where \( w^\top x - b > -\epsilon \) for some small \( \epsilon \), akin to “hard mining” for SVMs. We perform gradient ascent after each mining phase. An example result for this algorithm is shown in Figure 3.2 and we include further results below.

### 3.3 Optimizing the objective

Algorithm 1 gives a summary of our optimization procedure. We begin by sampling a set of patches from the positive dataset, and initialize our \( w_j \) vectors as the features for these patches. We initialize \( b_j \) to 0. For simplicity of notation in this section, we append \( b_j \) to \( w_j \) and append a \(-1\) to each feature vector \( x \). We can then “mine” through a set of images for patches where \( w_j^\top x > 0 \) for some \( j \). In practice, it greatly improves computational efficiency to have a separate round of mining initially on a small set of negative images, where we only update \( b_j \) to satisfy the constraint of (3.10).

After a round of mining on a single chunk of the data (including positives and negatives), we set the \( \alpha \)'s according to the procedure described in section 3.4. We must then optimize the following:

\[
\sum_{i=1}^{n_{pos}} \alpha_{i,j} \max(w_j^\top x_i^+, 0) - \lambda \sum_{j=1}^{m} \|[w_j]_{1:d}\|^2 \quad \text{s.t.} \quad \sum_{i=1}^{n_{neg}} \max(w_j^\top x_i^-, 0) \leq \beta \tag{3.6}
\]

Here, \( d \) is the data dimensionality, and \([\cdot]_{1:d} \) selects the first \( d \) components of the vector such that the bias term is excluded. Note that we can replace the \( = \) with a \( \leq \) in the constraint because it does not affect the solution; a decrease in \( b \) will always increase the objective, and hence the inequality constraint will always be tight at the solution. With this modification, it is straightforward to show that the constraint defines a convex set. At first glance, Expression (3.6) seems quite difficult to optimize, as we are maximizing a non-concave function. It is unlikely that a convex relaxation will be useful either, because different elements correspond to different local maxima of the objective. In practice, however, we can approximately optimize (3.6) directly, and do so efficiently. First, note that locally the function is a simple quadratic on an affine subspace, as long as \( w_j \) remains in a neighborhood where the sign of \( w_j^\top x \) does not change for any \( x \). Hence, we perform a form of projected gradient descent; pseudocode is given in the optimize function of Algorithm 1. We first compute the gradient of (3.6) and then find its projection \( \nabla \) onto the current affine subspace, i.e., the space defined by:

\[
\nabla^\top \sum_{i=1}^{n_{neg}} x_i^+ I(w_j^\top x_i^+ > 0) = 0 \tag{3.7}
\]

where \( I \) is the indicator function. This means that small updates in the direction \( \nabla \) will not result in constraint violations. Next, we perform a line search on \( w + t\nabla \), where \( t \) is the step size that we search over:

\[
t^* = \arg\max_t \sum_{i=1}^{n_{pos}} \alpha_{i,j} (w_j + t\nabla)^\top x_i^+ I(w_j^\top x_i^+ \geq 0) - \lambda \|[w_j + t\nabla]_{1:d}\|^2 \tag{3.8}
\]

This is a simple quadratic that can be solved analytically. If the maximum \( t^* \) of the line search does not cause \( w_j^\top x \) to change for any \( x \), then we accept this maximum, set
\[ w_j = w_j + t^* \nabla, \] and iterate. Otherwise, we set \( t \) equal to a pre-determined fixed constant, and update. If the step causes \( w_j^\top x_i^- \) to change sign for some \( x_i^- \), however, then we will no longer satisfy the constraint in (3.6). Ideally, we would orthogonally project \( w_j \) onto the constraint set, but finding the correct orthogonal projection is computationally expensive.

Hence, we approximate the projection operator with gradient descent (with respect to \( w_j \)) on the expression:

\[
\sum_{i=1}^{n_{neg}} \max(w_j^\top x_i^-, 0) - \beta
\]

This procedure is shown in the `satisfyConstrains` function of Algorithm 1. This function is piecewise linear, so gradient descent can be performed very efficiently. If the path of gradient descent is a straight line (i.e. for no \( x \) does \( w^\top x \) change sign) then this will be a proper projection, but otherwise it is an approximation. In practice we run the optimization on a fixed computational budget for each element, since in practice we find that learning more elements is more useful than optimizing individual elements more exactly.

### 3.4 Better Adaptive Bandwidth via Inter-Element Communication

Implicit in our formulation thus far is the idea that we do not want a single mode, but instead many distinct modes which each corresponds to a different element. In theory, mode-seeking will find every mode that is supported by the data. In practice, clusters often drift from weak modes to stronger modes, as demonstrated in Figure 3.2 (middle). One way to deal with this is to assign smaller bandwidths to patches in dense regions of the space [18], e.g., the window railing on row 1 of Figure 3.2 (middle) would hopefully have a smaller bandwidth and hence not match to the sidewalk barrier. However, estimating a bandwidth for every datapoint in our setting is not practical, so we seek an approach which only requires one pass through the data. Since patches in regions of the feature space with high density ratio will be members of many clusters, we want a mechanism that will reduce their bandwidth. To accomplish this, we extend the standard local (per-element) optimization of mean-shift into a joint optimization among the \( m \) different element clusters.

Specifically, we control how a single patch can contribute to multiple clusters by introducing a sharing weight \( \alpha_{i,j} \) for each patch \( i \) that is contained in a cluster \( j \), akin to soft-assignment in EM GMM fitting. Returning to our formulation, we maximize (again with respect to the \( w \)'s and \( b \)'s):

\[
\sum_{i=1}^{n_{pos}} \sum_{j=1}^{m} \alpha_{i,j} \max(w_j^\top x_i^+ - b_j, 0) - \lambda \sum_{j=1}^{m} ||w_j||^2 \text{ s.t. } \forall j \sum_{i=1}^{n_{neg}} \max(w_j^\top x_i^- - b_j, 0) = \beta \quad (3.10)
\]

Where each \( \alpha_{i,j} \) is chosen such that any patch which is a member of multiple clusters gets a lower weight. (3.10) also has a natural interpretation in terms of maximizing the “representativeness” of the set of clusters: clusters are rewarded for representing patches that are not represented by other clusters. But how can we set the \( \alpha \)'s? One way is to set \( \alpha_{i,j} = \max(w_j^\top x_i^+ - b_j, 0) / \sum_{k=1}^{m} \max(w_k^\top x_i^+ - b_k, 0) \), and alternate between setting the \( \alpha \)'s and optimizing the \( w \)'s and \( b \)'s at each iteration. Intuitively, this algorithm would be much like EM, alternating between softly assigning cluster memberships for each datapoint and
Algorithm 1: Discriminative Mode Seeking Pseudocode

Data: $I^+, I^{-}$: positive and negative image sets
Initialize $W = [w_1, ..., w_m]$ as random patches from positive images, with the last (bias) row 0;
Initialize $B = [b_1, ..., b_m]$ by running $W$ on a subset of $I^-$ and finding $b$'s that satisfy 3.3;
Set the last row of $W$ equal to $B$;
Distribute $I^+$ and $I^-$ evenly into $L$ sets, $I_1, ..., I_L$;
for $l ← 1$ to $L$ do
  Mine for patches $x$ in $I_l$ for which any of $W^T x > 0$
  for $j ← 1$ to $m$ do
    $X ←$ the set of $x$ for which $w_j^T x > 0$
    $[w_j] ←$ optimize($w_j, X$)
  end
end

Function optimize($w, X$)

$X^+, X^- ←$ Positive and negative examples from $X$, respectively;
while not converged and not timed out do
  $\nabla ← \sum_{x \in X^+, w_j^T x > 0} x - 2 \ast \lambda \|w\|_{1,0} \|x\|$; // Gradient of objective
  $\Pi ← \sum_{x \in X^-, w_j^T x > 0} x$; // Gradient of constraint
  $\nabla ← (\Pi \nabla^T \Pi) / \|\Pi\|^2$; // Project $\nabla$ to be orthogonal to $\Pi$
  $w ← w + t \ast \nabla$; // take a step of size $t$ (see text)
  $w ←$ satisfyConstraints($w, X^-$);
end
return $w$;

Function satisfyConstraints($w, X^-$)

while constraint is not satisfied do
  $\Pi ←$ sum of $x \in X^-$ where $w^T x > 0$; // Gradient of constraint
  $\delta ← \min \delta$ such that the sign of $(w - \delta \ast \Pi)^T x$ changes for some $x \in X^-$;
  if some $\delta_0 < \delta$ makes $(w - \delta_0 \ast \Pi)$ satisfy the constraint then
    $\delta ← \delta_0$;
  end
  $w ← w - \delta \ast \Pi$;
end
return $w$;

then optimizing each cluster. However, this goes against our mean-shift intuition: if two patches are really instances of the same element, then clusters initialized from those two points should converge to the same mode and not “compete” with one another. So, our heuristic is to first cluster the elements. Let $C_j$ be the assigned cluster for the $j$’th element. Then we set
Figure 3.2: Left: without competition, the algorithm from section 3.2 correctly learns a street lamp element. Middle: The same algorithm trained on a sidewalk barrier, which is too similar to the very common “window with railing” element, which takes over the cluster. Right: with the algorithm from section 3.4, the window gets down-weighted and the algorithm can learn the sidewalk barrier.

\[
\alpha_{i,j} = \max\left(\frac{w_j^\top x_i^+ - b_j}{\max(w_j^\top x_i^+ - b_j, 0) + \sum_{k=1}^m I(C_k \neq C_j) \max(w_k^\top x_i^+ - b_k, 0)}\right) \quad (3.11)
\]

In this way, any “competition” from elements that are too similar to each other is ignored. To obtain the clusters, we perform agglomerative (UPGMA) clustering on the set of element clusters, using the negative of the number of overlapping cluster members as a “distance” metric.

In practice, however, it is extremely rare that the exact same patch is a member of two different clusters; instead, clusters will have member patches that merely overlap with each other. Our heuristic deal with this is to compute a quantity \(\alpha'_{i,j,p}\) which is analogous to the \(\alpha_{i,j}\) defined above, but is defined for every pixel \(p\). Then we compute \(\alpha_{i,j}\) for a given patch by averaging \(\alpha'_{i,j,p}\) over all pixels in the patch. Specifically, we compute \(\alpha_{i,j}\) for patch \(i\) as the mean over all pixels \(p\) in that patch of the following quantity:

\[
\alpha'_{i,j,p} = \frac{\max(w_j^\top x_i^+ - b_j, 0)}{\max(w_j^\top x_i^+ - b_j, 0) + \sum_{x \in Ov(p)} \sum_{k=1}^m I(C_k \neq C_j) \max(w_k^\top x_i^+ - b_k, 0)} \quad (3.12)
\]

Where \(Ov(p)\) denotes the set of features for positive patches that contain the pixel \(p\).

It is admittedly difficult to analyze how well these heuristics approximate the adaptive bandwidth approach of [18], and even there the setting of the bandwidth for each datapoint has heuristic aspects. However, empirically our approach leads to improvements in performance as discussed below, and suggests a potential area for future work.

### 3.5 Evaluation via Purity-Coverage Plot

Our aim is to discover visual elements that are maximally representative and discriminative. To measure this, we define two quantities for a set of visual elements: **coverage** (which captures representativeness) and **purity** (which captures discriminativeness). Given a held-out test set, visual elements will generate a set of patch detections. We define the coverage of this set of patches to be the fraction of the pixels from the positive images claimed by at least one patch. We define the purity of a set as the percentage of the patches that share the same label. For an individual visual element, of course, there is an inherent trade-off between purity and coverage: if we lower the detection threshold, we cover more pixels but
Figure 3.3: Purity-coverage graph for our algorithm and baselines. In each plot, purity measures the accuracy of the element detectors, whereas coverage captures how often they fire. Curves are computed over the top 25 (left) and 200 (right) elements. Higher is better.

Figure 3.4: Coverage versus the number of elements used in the representation. On the left we keep only the detections with a score higher than the score of the detector’s first error (i.e. purity 1). On the right, we lower the detection threshold until the elements are 90% pure. Note: this is the same purity and coverage measure for the same elements as Figure 3.3, just plotted differently.

also increase the likelihood of making mistakes. Hence, we can construct a purity-coverage curve for a set of elements, analogous to a precision-recall curve. We could perform this analysis on any dataset containing positive and negative images, but [25] presents a dataset which is particularly suitable. The goal is to mine visual elements which define the look and feel of a geographical locale, with a training set of 2,000 Paris Street View images and 8,000 non-Paris images, as well as 2,999 of both classes for testing. Purity-coverage curves for this dataset are shown in Figure 3.3.

To plot the curve for a given value of purity $p$, we rank all patches by $w^T x - b$ independently for every element, and select, for a given element, all patches up until the last point where the element has the desired purity. We then compute the coverage as the union of patches selected for every element. Because we are taking a union of patches, adding more
elements can only increase coverage, but in practice we prefer concise representations, both for interpretability and for computational reasons. Hence, to compare two element discovery methods, we must select exactly the same number of elements for both of them. Different works have proposed different heuristics for selecting elements, which would make the resulting curves incomparable. Hence, we select elements in the same way for all algorithms, which approximates an “ideal” selection for our measure. Specifically, we first fix a level of purity (95%) and greedily select elements to maximize coverage (on the testing data) for that level of purity. Hence, this ranking serves as an oracle to choose the “best” set of elements for covering the dataset at that level of purity. While this ranking has a bias toward large elements (which inherently cover more pixels per detection), we believe that it provides a valuable comparison between algorithms. Our purity-coverage curves are shown in Figure 3.3 for the 25 and 200 top elements, respectively. We can also slice the same data differently, fixing a level of purity for all elements and varying the number of elements, as shown in Figure 3.4.

**Baselines:** We included five baselines of increasing complexity. Our goal is not only to analyze our own algorithm; we want to show the importance of the various components of previous algorithms as well. We initially train 20,000 visual elements for all the baselines, and select the top elements using the method above. The simplest baseline is “Exemplar LDA,” proposed by [47]. Each cluster is represented by a hyperplane which maximally separates a single seed patch from the negative dataset learned via LDA, i.e. the negative distribution is approximated using a single multivariate Gaussian. To show the effects of re-clustering, “LDA Retrained” takes the top 5 positive-set patches retrieved in Exemplar LDA (including the initial patch itself), and repeats LDA, separating those 5 from the negative Gaussian. This is much like the well-established method of “query expansion” for retrieval, and is similar to [60] (although they use multiple iterations of query expansion). Finally, “LDA Retrained 5 times” begins with elements initialized via the LDA retraining method, and re-trains the LDA classifier, each time throwing out the previous top 5 used to train the previous LDA, and selecting a new top 5 from held-out data. This is much like the iterative SVM training of [25], except that it uses LDA instead of an SVM. Finally, we include the algorithm of [25], which is a weakly supervised version of [106], except that knn is being used for initialization instead of kmeans. The iterations of retraining clearly improve performance, and it seems that replacing LDA with an SVM also gives improvement, especially for difficult elements.

**Implementation details:** We use the same patch descriptors described in [25] and whiten them following [47]. We mine elements using the online version of our algorithm, with a chunk size of 1000 (200 Paris, 800 non-Paris per batch). We set $\beta = \lambda = t/500$ where $t$ is the iteration number, such that the bandwidth increases proportional to the number of samples. We train the elements for about 200 gradient steps after each chunk of mining. To compute $\alpha_{i,j}$ for patch $i$ and detector $j$, we actually use scale-space voxels rather than pixels, since a large detection can completely cover a small detection but not vice versa. Hence, the set of scale-space voxels covered is a 3D box, the width of the bounding box by its height (both discretized by a factor of 8 for efficiency) by 5, covering exactly one “octave” of scale space (i.e. $\log_2(\sqrt{\text{width} \times \text{height}}) \times 5$ through $\log_2(\sqrt{\text{width} \times \text{height}}) \times 5 + 4$). For experiments without inter-element communication, we simply set $\alpha_{i,j}$ to $0.1$. Finally, to reduce the impact of highly redundant textures, we divide $\alpha_{i,j}$ divided by the total number of detections for element $j$ in the image containing $i$. Source code will be available online.
Figure 3.5: For each correctly classified image (left), we show four elements (center) and heatmap of the locations (right) that contributed most to the classification.

### 3.6 Scene Classification

Finally, we evaluate whether our visual element representation is useful for scene classification. We use the MIT Scene-67 dataset [93], where machine performance remains substantially below human performance. For indoor scenes, objects within the scene are often more useful features than global scene statistics [60]: for instance, shoe shops are similar to other stores in global layout, but they mostly contain shoes.

**Implementation details:** We used the original Indoor-67 train/test splits (80 training and 20 testing images per class). We learned 1600 elements per class, for a total of 107,200
Table 3.1: Results on MIT 67 scenes

<table>
<thead>
<tr>
<th>Method</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>ROI + Gist</td>
<td>26.05</td>
</tr>
<tr>
<td>MM-scene</td>
<td>28.00</td>
</tr>
<tr>
<td>DPM [89]</td>
<td>30.40</td>
</tr>
<tr>
<td>CENTRIST</td>
<td>36.90</td>
</tr>
<tr>
<td>Object Bank [72]</td>
<td>37.60</td>
</tr>
<tr>
<td>RBoW [90]</td>
<td>37.93</td>
</tr>
<tr>
<td>D-Patches [106]</td>
<td>38.10</td>
</tr>
<tr>
<td>D-Parts [113]</td>
<td>44.84</td>
</tr>
<tr>
<td>BoP [60]</td>
<td>46.10</td>
</tr>
<tr>
<td>miSVM [74]</td>
<td>46.40</td>
</tr>
<tr>
<td>Ours (no inter-element, §3.2)</td>
<td>50.15</td>
</tr>
<tr>
<td>Ours (§3.4)</td>
<td>51.40</td>
</tr>
<tr>
<td>DPM [89]</td>
<td>50.40</td>
</tr>
<tr>
<td>BoP-IFV [60]</td>
<td>51.40</td>
</tr>
<tr>
<td>Ours+IFV</td>
<td>54.03</td>
</tr>
<tr>
<td>RBoW [90]</td>
<td>54.40</td>
</tr>
<tr>
<td>MMDL [127]</td>
<td>55.15</td>
</tr>
<tr>
<td>Ours+IFV</td>
<td>56.87</td>
</tr>
</tbody>
</table>

Figure 3.6: Each of these images was misclassified by the algorithm, and the heatmaps explain why. For instance, it may not be obvious why a corridor would be classified as a staircase, but we can see (top right) that the algorithm has identified the railings as a key staircase element, and has found no other staircase elements the image.

elements, following the procedure described above. We include right-left flipped images as extra positives. 5 batches were sufficient, as this dataset is smaller. We also used smaller descriptors: 6-by-6 HOG cells, corresponding to 64-by-64 patches and 1188-dimensional descriptors. We again select elements by fixing purity and greedily selecting elements to maximize coverage, as above. However, rather than defining coverage as the number of pixels (which is biased toward larger elements), we simply count the detections, penalizing for overlap: we penalize each individual detection by a factor of \(1/(1 + n_{overlap})\), where \(n_{overlap}\) is the number of detections from previously selected detectors that a given detection overlaps with. We select 200 top elements per class. To construct our final feature vector, we use a 2-level (1x1 and 2x2) spatial pyramid and take the max score per detector per region, thresholded at \(-0.5\) (since below this value we do not expect the detection scores to be meaningful) resulting in a 67,000-dimensional vector. We average the feature vector for the right and left flips of the image, and classify using 67 one-vs-all linear SVM’s. Note that this differs from [106], which selects only the elements for a given class in each class-specific SVM.

Figure 3.5 shows a few qualitative results of our algorithm. Quantitative results and comparisons are shown in Table 3.1. We significantly outperform other methods based on discriminative patches, suggesting that our training method is useful. We even outperform the Improved Fisher Vector of [60], as well as IFV combined with discriminative patches (IFV+BoP). Finally, although the optimally-performing representation is dense (about 58% of features are nonzero), it can be made much sparser without sacrificing much perfor-
mance. For instance, if we trivially zero-out low-valued features until fewer than 6% are nonzero, we still achieve 60.45% accuracy.

3.7 Conclusion

We developed an extension of the classic mean-shift algorithm to density ratio estimation, showing that the resulting algorithm could be used for element discovery, and demonstrating state-of-the-art results for scene classification. However, there is still much room for improvement in weakly-supervised element discovery algorithms. For instance, our algorithm is limited to binary labels, but image labels may be continuous (e.g., GPS coordinates or dates). Also, our elements are detected based only on individual patches, but images often contain global structures beyond patches.
Chapter 4

Object Discovery by Learning to Predict Context

4.1 Introduction

Proponents of unsupervised representation learning [36, 52, 67, 88] and unsupervised object discovery [31, 46, 61, 70, 91, 97, 98, 108] 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 [118]. Unsupervised object discovery has, however, proven extremely difficult; one state-of-the-art result [67] 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 [22]. 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 [22, 25, 29, 56, 60, 74, 106, 113, 127, 132]. The supervision (e.g., scene labels, GPS coordinates, etc.) gives information about which image regions should be 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 [87] 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 4.1b, 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 4.1c 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.
Figure 4.1: 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.

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 [128]. 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’ [1], 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 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 [28]. The model’s alignment errors are thus corrected before they drift too far.

4.2 Overview

At a high level, our pipeline is similar to algorithms for mid-level patch discovery [25,106], especially in the early stages. Like [25], 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. [25] uses normalized correlation as a distance metric, but we found Exemplar LDA [47], with a Gaussian learned
Figure 4.2: 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 [83]).

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 4.2 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 [32, 36, 52, 111, 124, 128], 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:

$$P(c|p) = \prod_{k=1}^{m} P(c_k|c_1, ..., c_{k-1}, p)$$ (4.1)

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. Factorizations like this have been used modeling texture and low-level image statistics [26, 115, 116], and handwritten digits [66, 84], but we are not aware of attempts to capture higher-level dependencies needed to model objects in high-resolution images. We show that these incremental predictions can be made efficiently and with surprising accuracy in this setting, enough that the resulting likelihoods can be compared between our thing and stuff models.

4.3 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 4.2 $\mathcal{P}$ would be a strict subset of the region outlined in green for all but the first term in the product in Equation 4.1). 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 4.2 $\mathcal{C}$ 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. 4.1) for the thing model can now be written more formally as:
Figure 4.3: 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.

\[ P_T(H_0^C|H_0^P) = \prod_{t=1}^{[C]} P_T(H_0^C_t|H_0^C[1:t-1], H_0^P) \]  \hspace{1cm} (4.2)

We will have a similar factorization for \( P_S \) of the stuff model. In Figure 4.2, the region outlined in red corresponds to \( H_0^C[t] \), and those outlined in green correspond to \( \{H_0^C[1:t-1], H_0^P\} \). We repeat this computation of \( P(H_0^C_t|H_0^C[1:t-1], H_0^P) \) 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 \( \{H_0^C[1:t-1], H_0^P\} \) the “condition” region, and \( H_0^C[t] \) the “prediction” region. We choose the order of \( C \) 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.

### 4.3.1 Stuff Model

To construct the stuff prediction \( P_S(H_0^C[t]|H_0^C[1:t-1], H_0^P) \) (which we will abbreviate as \( P_S^C[t] \)), the simplest approach is to 1) extract a subset of the condition region that is spatially close to \( C[t] \), 2) find nearest neighbors for that subset from a large database of images, and 3) use the context around those retrieved 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 4.3. We begin with a query image (Figure 4.3a), with a condition region (in green) and a prediction region (in red). We assume that we have available a ‘dictionary’ (Figure 4.3b) constructed from a large sample of image patches (Figure 4.3b), 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 4.3b). 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 4.3b), and learn a Gaussian Mixture Model (GMM)
Figure 4.4: 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.

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 4.3b) 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 4.3c. 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 4.3.

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 4.3a). To deal with this, we marginalize out any such cells from the GMM as shown in Figure 4.3d (Hence why 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).

4.3.2 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 \( P_T (H^0_{C[t]} | H^0_{C[1:t−1]}, H^0_P) \) (which we will abbreviate as \( p^T_{C[t]} \)) 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 \( H^0_{C[t]} \) 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.

Predicting from correspondence

Formally, recall that $C$ and $P$ 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 $H^i_i(x, y)$ is as similar as possible to $H^0_C(x, y)$ for all $(x, y)$ in the condition region. Note that we are interested in correspondence for $C[C[t] = (x_t, y_t)$, the prediction region, but we aren’t allowed to access the HOG feature at $H^0_C(x, y)$; 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:

$$f^i(x_t, y_t) = f^i(x^*_t, y^*_t) + (x_t, y_t) - (x^*_t, y^*_t)$$  \hspace{1cm} (4.3)

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 4.4. 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.

$$f^i(x, y) := [\mu^i_{x,y}, \Sigma^i_{x,y}]$$  \hspace{1cm} (4.4)

Thus, $f^i(x, y)$ defines Gaussian distribution over the HOG grid of image $i$. (In Equation 4.3, the addition is only performed on $\mu$: i.e. $f^i(x, y) + (a, b) = [\mu^i_{x,y} + (a, b), \Sigma^i_{x,y}]$). Figure 4.4 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 4.4, it guesses that the prediction region could correspond to panel or to wheel.
Computing the warping $f$

The goal in this section is to optimize the $\mu$ and $\Sigma$ in equation [4.4]. 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 [4.4]), which we parameterize by $\mu_{x,y}^i$ and $\Sigma_{x,y}^i$. We minimize with respect to $\Sigma$ and $\mu$:

$$E(\Sigma, \mu) = \sum_{i,x,y} c_{x,y} \Phi(H_{x,y}^0, \mu_{x,y}^i, \Sigma_{x,y}^i) + \lambda \sum_{i,x,y} \sum_{(x',y') \in N(x,y)} \Psi(\mu_{x,y}^i, \Sigma_{x,y}^i, \mu_{x',y'}^i, \Sigma_{x',y'}^i)$$

(4.5)

Here, $\Phi$ rewards $\mu_{x,y}^i$ and $\Sigma_{x,y}^i$ for mapping $H_{x,y}^0$ to similar HOG cells in image $i$. $\Psi$ 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.

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 $p_T^i/(p_T^i + p_S^i)$. We use $c_{x,y}$ to weight these probabilities. However, $p_T^i$ and $p_S^i$ 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:

$$L_{z,y}^T = \prod_{(u,v) \in \{C[1:t-1], T\}} (p_{u,v}^T)^{\delta([u,v]-[x,y])}; \quad L_{z,y}^S = \prod_{(u,v) \in \{C[1:t-1], P\}} (p_{u,v}^S)^{\delta([u,v]-[x,y])}$$

(4.6)

We then compute:

$$c_{x,y} = L_{z,y}^T / (L_{z,y}^T + L_{z,y}^S)$$

(4.7)

$p_T^i$ and $p_S^i$ are, respectively, $p_T^i$ and $p_S^i$ 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 $\sigma$), normalized such that $\sum_{(u,v) \in \{C[1:t-1], P\}} \rho([u,v]-[x,y]) = 1$.

Computing $\Phi$: the unary potentials

Intuitively, our definition of the unary potentials $\Phi$ 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 $i$, $\Sigma_{x,y}$ will grow to explain as much as it can. Mathematically:

$$\Phi(H_{x,y}^0, \mu_{x,y}^i, \Sigma_{x,y}^i) = -\sum_{(u,v)} \log \mathcal{N}((u,v); \mu_{x,y}^i, \Sigma_{x,y}^i) * \mathcal{N}(H_{u,v}^i; H_{x,y}^0, \Sigma_H) + \gamma \mathcal{N}(H_{u,v}^i; \mu_H, \Sigma_H))$$

(4.8)

Here, $\mathcal{N}(\cdot; \mu, \Sigma)$ represents a multivariate normal PDF with mean $\mu$ and variance $\Sigma$, $\gamma$ is a regularization constant (set to 100 in our experiments), and $\mu_H, \Sigma_H$ 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 \( \mu_{x,y} \) is explained by a Gaussian in HOG space with mean \( H_{x,y}^0 \). The term involving \( \mu \) 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.

**Computing \( \Psi \): the pairwise potentials**  The pairwise potentials \( \Psi \) 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:

\[
\Psi(\mu_{x,y}^i, \Sigma_{x,y}^i, \mu_{x',y'}^i, \Sigma_{x',y'}^i) = D\left( \mathcal{N}(\cdot; \mu_{x,y}^i, \Sigma_{x,y}^i), \mathcal{N}(\cdot; \mu_{x',y'}^i - [(x', y') - (x, y)], \Sigma_{x',y'}^i) \right)
\]

where \( D \) is the KL-divergence between the two distributions.

**Improving affine invariance: pairwise potentials \( \Omega_{x,y} \)**  The spatial priors defined in Equation 4.9 often work 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 \( \lambda \) (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 \( \alpha_{x,y}^i \) at each HOG cell in the query image (\( \alpha_{x,y}^i \) is represented with a 2-by-2 affine transformation matrix). Then we minimize:

\[
E'(\Sigma, \mu, \alpha) = E(\Sigma, \mu) + \lambda' \sum_{x,y,i} \Omega_{x,y,i}(\Sigma_i, \mu_i, \alpha_{x,y}^i)
\]

In this equation:

\[
\Omega_{x,y,i}(\Sigma_i, \mu_i, \alpha_{x,y}^i) = \sum_{\left[ (x', y'), (x'', y'') \right] \in L_i^{H}(x, y)} D(\mathcal{N}(\cdot; \mu_{x',y'}^i, \Sigma_{x',y'}^i), \mathcal{N}(\cdot; \mu_{x'',y''}^i - [(x'', y'') - (x', y')], \alpha_{x,y}^i, \Sigma_{x'',y''}^i))
\]

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 \( \alpha_{x,y}^i \) 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 \( \Psi \), 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 \( \Phi \), so we get this approximate affine invariance essentially for free.
Optimization for the warping f

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 4.8. 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:

$$
\zeta_{i,x,y,u,v} = \frac{N((u,v); \mu_{i,x,y}, \Sigma_{i,x,y}) \cdot \mathcal{N}(H_{i,x,y}; H_{0,x,y}, \Sigma_H)}{N((u,v); \mu_{x,y}, \Sigma_{x,y}) \cdot \mathcal{N}(H_{0,x,y}; H_{0,x,y}, \Sigma_H) + \gamma \mathcal{N}(H_{i,x,y}; H_{0,x,y}, \Sigma_H)}
$$

(4.12)

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 $\alpha_{i,x,y}$.

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

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

(4.13)

Note that all of the edges summed over in Equation 4.11 are either vertical or horizontal, so we may optimize the rows of $\alpha$ separately, in each case focusing either on vertical or horizontal edges. The only term in the KL divergence that matters is the one involving $\mu$.

Hence, we can write the objective over the first row of $\alpha$ as:

$$
\frac{\lambda'}{2} \sum_{[(x',y'),(x'',y'')]} \left( \mu_{x',y'} - \mu_{x'',y''} - \alpha_{x',y'} \right) \left( \Sigma_{x',y'}^{-1} + (\Sigma_{x'',y''}^{-1})^{-1} \right) \left( \mu_{x',y'} - \mu_{x'',y''} - \alpha_{x',y'} \right)^\top
$$

(4.14)

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):

$$
c_{x,y} \sum_{u,v} \zeta_{i,x,y,u,v} \left( \mu_{i,x,y} - (u,v) \right) \left( \Sigma_{x,y}^{-1} \right) \left( \mu_{i,x,y} - (u,v) \right)^\top
$$

(4.15)

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

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

(4.16)
When symmetrizing the KL-divergence, the gradient is:

\[
\frac{\partial}{\partial L} \text{Equation 4.13} = (\mu_{x,y} - \mu_{x',y'}) - \alpha [(x, y) - (x', y')] \right) \ast \left( (\Sigma_{x,y}^{-1}) + (\Sigma_{x',y'}^{-1}) \right) \ast \left( (\mu_{x,y} - \mu_{x',y'}) - \alpha [(x, y) - (x', y')] \right)^\top
\]  

(4.17)

\(L^{-1}([(x, y), (x', y')])\) 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 \(\Sigma\) 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 with respect to \(\Sigma\) is:

\[
\frac{\partial}{\partial \Sigma} \left( \sum_{x,y} \xi(x,y) = \frac{1}{2} \left( \text{tr} \left( \Sigma_{x,y}^{-1} \Sigma_{x',y'}^{-1} + \Sigma_{x,y}^{-1} \Sigma_{x,y}^{-1} \right) - 2k \right) \right)
\]

(4.18)

We add to this the gradient with respect to the higher order terms. We’ll start with the terms in the KL divergence (Equation 4.13) that don’t involve \(\mu\):

\[
\frac{1}{2} \sum_{(x', y') \in N(x, y)} \left( \text{tr} \left( \Sigma_{x,y}^{-1} \Sigma_{x',y'}^{-1} + \Sigma_{x,y}^{-1} \Sigma_{x',y'}^{-1} \right) - 2k \right).
\]

(4.19)

Here, \(b\) counts the number of terms where such a KL divergence occurs that includes \(\Sigma_{x',y'}\), i.e. \(b = \lambda + \lambda' + L^{-1}([(x, y), (x', y')])\). Note that the final term from Equation 4.13 cancels when symmetrizing the KL-divergence. The gradient of this is:

\[
\frac{1}{2} \sum_{(x', y') \in N(x, y)} \left( \Sigma_{x,y}^{-1} \Sigma_{x',y'}^{-1} + \Sigma_{x,y}^{-1} \Sigma_{x',y'}^{-1} \right)
\]

(4.20)

Finally, the term involving \(\mu\) in Equation 4.13 is essentially the same as Equation 4.16 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:

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

(4.21)

Where:

\[
\Xi'(\alpha) = (\Sigma_{x,y}^{-1})^{-1} \ast \left( (\mu_{x,y} - \mu_{x',y'}) - \alpha [(x, y) - (x', y')] \right)^\top \ast \left( (\mu_{x,y} - \mu_{x',y'}) - \alpha [(x, y) - (x', y')] \right) \ast (\Sigma_{x,y})^{-1}
\]

(4.22)

**Making Predictions from the Warping**

This correspondence \(f\) 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 \(H^1 \ldots H^{n-1}\) that the prediction region potentially corresponds to.
\[
p_{C[t]} = N(H_{C[t]}^0; \mu_{C[t]}^I, \Sigma_{C[t]}^f)
\]

where

\[
\mu_{C[t]}^I = \sum_{i,u,v} \eta_{C[t],u,v}^I H_{u,v}^i; \quad \Sigma_{C[t]}^f = \sum_{i,u,v} \eta_{C[t],u,v}^I (H_{u,v}^i - \mu_{C[t]}^I)(H_{u,v}^i - \mu_{C[t]}^I)^T
\]

Here, \(\sum_{i,u,v} \eta_{C[t],u,v}^I = 1\). There are two components of this weighting: \(\eta_{C[t],u,v}^I = \omega_{C[t]}^I, u,v\omega_{C[t]}^I\). The first is based on the spatial correspondence \(f\), and is defined as \(w_{C[t],u,v}^I = N((u,v); \mu_{x_t,y_t}^I, \Sigma_{x_t,y_t}^I)\) 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 \(\omega_{C[t]}^I\) 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 \(C[t]\).

**Computing \(\omega_{C[t]}^I\)** Intuitively, our use of \(\omega_{C[t]}^I\) in Equation 4.24 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-image predictions in a manner similar to how we obtain predictions for the full model: i.e., we estimate a likelihood \(P_{C[t]}^I(H_{C[t]}^0; [H_{C[t]}^0, H_{C[t]}^0, H_{C[t]}^0])\) which we will abbreviate as \(g_{C[t]}^I\) 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:

\[
g_{C[t]}^I = P_{C[t]}^I(H_{C[t]}^0; [H_{C[t]}^0, H_{C[t]}^0, H_{C[t]}^0]) = N(H_{C[t]}^0; \theta_{C[t]}^I, \Sigma_{C[t]}^f)
\]

Where

\[
\theta_{C[t]}^I = \sum_{u,v} u_{C[t],u,v}^I H_{u,v}^i
\]

Note the similarities between the prediction made by one expert and the prediction made by the full thing model. \(\Sigma_{C[t]}^f\) is actually identical to Equation 4.23. i.e. it is estimated using data from all images, \(\theta_{C[t]}^I\) is identical to \(\mu_{C[t]}^I\) in Equation 4.23 except that it integrates data from only a single image \(i\). Given these per-image predictions \(g_{C[t]}^I\), we assume that any expert which predicted well for cells near to \(C[t]\) will also do a good job predicting the cell \(C[t]\). We use Bayes rule to estimate the probability of each expert given the data: i.e. set \(\omega_{C[t]}^I\) 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 \(C[t] = (x_t, y_t)\), we compute the following product of data likelihoods:

\[
\omega_{C[t]}^I \propto \prod_{(x,y) \in \{C[t], C[t-1], \Phi\}} g_{(x,y)}^I \rho((x,y) - (x_t, y_t))
\]
Figure 4.5: 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 \cite{120}.

Here, \( \rho \) is the same as in Equation 4.6. We normalize \( \omega^i_C[t] \) such that \( \sum_i \omega^i_C[t] = 1 \) (since our goal is to compute a relative weighting between images), and we cap \( \omega^i_C[t] \) 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” \( U_i \) of a given patch \( i \) as:

\[
U_i = \sum_t \omega^i_C[t] \ast c_C[t]
\]

(4.28)

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 4.23) exclusively, or it ‘mimics’ the stuff model exactly (i.e. it sets its conditional distribution \( P_T(h|H^0_{C[1:t-1]}, H^0_P) \) equal to \( P_S(h|H^0_{C[1:t-1]}, H^0_P) \) for all \( h \)). Note that when the thing model mimics the stuff model, \( p^T_C[t] \) will be equal to \( p^S_C[t] \), and hence the value of \( H^0_C \) 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 \( C[t] \) corresponds to regions in other images that were predicted poorly.

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 \( C[t] \) corresponds to cells in other images that were predicted poorly. For (1), recall that \( c_{x,y} \) (Equation 4.7) 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 \( \pi_{u,v}^{i} \) be the probability of the thing model for image at location \((u,v)\) in that image (i.e., \( \hat{p}_u^T/v \) computed while predicting image \( i \), without allowing mimicry). We can integrate these values as

$$\beta_{C[t]} = \sum_i \omega_{C[t]}^i \sum_{u,v} w_{C[t],u,v}^i \pi_{u,v}^i$$

(4.29)

Here, the sum over \( i \) sums over only those images \( i \) where prediction has already been computed; \( w_{C[t],u,v}^i \) is defined as in equation 4.24 and \( \omega_{C[t]}^i \propto \omega_{C[t]}^i \) as defined in equation 4.24 but re-normalized to sum to 1 over the smaller set of images. We mimic if \( \beta_{C[t]} \times c_{C[t]} < 0.3 \), a threshold we determined empirically. If we haven’t yet performed verification on any other images, we simply set \( \beta_{C[t]} = 0.5 \).

### 4.3.3 From Patch Prediction to Object Discovery

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 Equation 4.28), 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.

### 4.4 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.

#### 4.4.1 Quantitative Results: Purity-Coverage on PASCAL VOC 2007

Following the experimental setup of [106], 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 [22], 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 [106], we follow the previous section and plot purity and coverage on the same graph, since we find it makes the methods more comparable.) The result is shown in Figure 4.6 left. We slightly outperform [106], especially for the high purity regime (we get nearly 10% extra coverage before we make our first mistake). However, note that [106] 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 [110]. Specifically, given a selection \( \chi \) of clusters, let \( s_{i,j}^{\chi} \) be the log likelihood ratio for the \( j \)’th highest-scoring patch in image \( i \) out of all clusters contained in \( \chi \). We greedily select clusters to include in \( \chi \) to maximize

\[
\sum_{i,j} 2^{-j} s_{i,j}^{\chi},
\]

i.e. exponentially discounting the scores of patches from the same image.
Figure 4.7: 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.
Figure 4.8: 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.

4.4.2 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 4.7 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 4.3.2) to their locations in the image. Black borders indicate either the edge of the image, or the display cut off. Figure 4.8 shows a few examples of discovered regions that correspond to 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 4.4.1 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 [25]. Our full ranking is available online at: http://graphics.cs.cmu.edu/projects/contextPrediction/

4.4.3 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 [49]. 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 $cc_{[t]} \ast w_{[t],u,v} \ast s$, 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 $\text{thing}$ over likelihood of $\text{thing} + \text{likelihood of stuff}$) for the patch, and $cc_{[t],x,y}$ 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 $\text{thing}$ 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 [49] isn’t suitable in our situation, since they only mea-
Figure 4.9: Intuition behind our baseline keypoint prediction algorithm. See text for details.

sure 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 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 4.6 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. Chance performance is well below 1% precision. Figure 4.10 shows the raw output for one image, after thresholding the confidence of keypoint transfers.

4.5 Baseline algorithm for unsupervised keypoint prediction

Exemplar-LDA based baseline 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 4.9. 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
Figure 4.10: 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.

to the LDA score of the retrieved patch. In Figure 4.9, 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 4.9, 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.

4.6 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 $P(H^0_{c[t]}|H^0_{c[1:t-1]}; H^0_p)$ 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.
Chapter 5

Deep Visual Representation Learning by Unsupervised Context Prediction

5.1 Introduction

Recently, new computer vision methods have leveraged large datasets of millions of labeled examples to learn rich, high-performance visual representations [65]. Yet efforts to scale these methods to truly Internet-scale datasets (i.e. hundreds of billions of images) are hampered by the sheer expense of the human annotation required. A natural way to address this difficulty would be to employ unsupervised learning, which aims to use data without any annotation. Unfortunately, despite several decades of sustained effort, unsupervised methods have not yet been shown to extract useful information from large collections of full-sized, real images. After all, without labels, it is not even clear what should be represented. How can one write an objective function to encourage a representation to capture, for example, objects, if none of the objects are labeled?

Interestingly, in the text domain, context has proven to be a powerful source of automatic supervisory signal for learning representations [3, 16, 80, 86]. Given a large text corpus, the idea is to train a model that maps each word to a feature vector, such that it is easy to predict the words in the context (i.e., a few words before and/or after) given the vector. This converts an apparently unsupervised problem (finding a good similarity metric between words) into a “self-supervised” one: learning a function from a given word to the words surrounding it. Here the context prediction task is just a “pretext” to force the model to learn a good word embedding, which, in turn, has been shown to be useful in a number of real tasks, such as semantic word similarity [80].

This chapter aims to provide a similar “self-supervised” formulation for image data: a supervised task involving predicting the context for a patch. Our task is illustrated in Figures 5.1 and 5.2. We sample random pairs of patches in one of eight spatial configurations, and present each pair to a machine learner, providing no information about the patches’ original position within the image. The algorithm must then guess the position of one patch relative to the other. Our underlying hypothesis is that doing well on this task requires understanding scenes and objects, i.e. a good visual representation for this task.
will need to extract objects and their parts in order to reason about their relative spatial location. “Objects,” after all, consist of multiple parts that can be detected independently of one another, and which occur in a specific spatial configuration (if there is no specific configuration of the parts, then it is “stuff” [1]). We present a ConvNet-based approach to learn a visual representation from this task. We demonstrate that the resulting visual representation is good for both object detection, providing a significant boost on PASCAL VOC 2007 compared to learning from scratch, as well as for unsupervised object discovery / visual data mining. This means, surprisingly, that our representation generalizes across images, despite being trained using an objective function that operates on a single image at a time. That is, instance-level supervision appears to improve performance on category-level tasks.

Unsupervised representation learning can also be formulated as learning an embedding (i.e. a feature vector for each image) where images that are semantically similar are close, while semantically different ones are far apart. One way to build such a representation is to create a supervised “pretext” task such that an embedding which solves the task will also be useful for other real-world tasks. For example, denoising autoencoders [4, 119] use reconstruction from noisy data as a pretext task: the algorithm must connect images to other images with similar objects to tell the difference between noise and signal. Sparse autoencoders also use reconstruction as a pretext task, along with a sparsity penalty [88], and such autoencoders may be stacked to form a deep representation [67, 69]. (however, only [67] was successfully applied to full-sized images, requiring a million CPU hours to discover just three objects). We believe that current reconstruction-based algorithms struggle with low-level phenomena, like stochastic textures, making it hard to even measure whether a model is generating well.

Another pretext task, and the focus of this chapter, is “context prediction.” A strong tradition for this kind of task already exists in the text domain, where “skip-gram” [80] models have been shown to generate useful word representations. The idea is to train a model (e.g. a deep network) to predict, from a single word, the \( n \) preceding and \( n \) succeeding words. In principle, similar reasoning could be applied in the image domain, a kind of visual “fill
Figure 5.2: The algorithm receives two patches in one of these eight possible spatial arrangements, without any context, and must then classify which configuration was sampled.

in the blank” task, but, again, one runs into the problem of determining whether the predictions themselves are correct [23], unless one cares about predicting only very low-level features [26, 66, 115]. To address this, [78] predicts the appearance of an image region by consensus voting of the transitive nearest neighbors of its surrounding regions. Our previous work in chapter 4 explicitly formulates a statistical test to determine whether the data is better explained by a prediction or by a low-level null hypothesis model.

The key problem that these approaches must address is that predicting pixels is much harder than predicting words, due to the huge variety of pixels that can arise from the same semantic object. In the text domain, one interesting idea is to switch from a pure prediction task to a discrimination task [16, 86]. In this case, the pretext task is to discriminate true snippets of text from the same snippets where a word has been replaced at random. A direct extension of this to 2D might be to discriminate between real images vs. images where one patch has been replaced by a random patch from elsewhere in the dataset. However, such a task would be trivial, since discriminating low-level color statistics and lighting would be enough. To make the task harder and more high-level, in this chapter, we instead classify between multiple possible configurations of patches sampled from the same image, which means they will share lighting and color statistics, as shown on Figure 5.2.

5.2 Learning Visual Context Prediction

We aim to learn an image representation for our pretext task, i.e., predicting the relative position of patches within an image. We employ Convolutional Neural Networks (ConvNets), which are well known to learn complex image representations with minimal human feature design. Building a ConvNet that can predict a relative offset for a pair of patches is, in principle, straightforward: the network must feed the two input patches through several convolution layers, and produce an output that assigns a probability to each of the eight spatial configurations (Figure 5.2) that might have been sampled (i.e. a softmax output).
Figure 5.3: Our architecture for pair classification. Dotted lines indicate shared weights. ‘conv’ stands for a convolution layer, ‘fc’ stands for a fully-connected one, ‘pool’ is a max-pooling layer, and ‘LRN’ is a local response normalization layer. Numbers in parentheses are kernel size, number of outputs, and stride (fc layers have only a number of outputs). The LRN parameters follow [65]. All conv and fc layers are followed by ReLU nonlinearities, except fc9 which feeds into a softmax classifier.

Note, however, that we ultimately wish to learn a feature embedding for individual patches, such that patches which are visually similar (across different images) would be close in the embedding space.

To achieve this, we use a late-fusion architecture shown in Figure 5.3: a pair of AlexNet-style architectures [65] that process each patch separately, until a depth analogous to fc6 in AlexNet, after which point the representations are fused. For the layers that process only one of the patches, weights are tied between both sides of the network, such that the same fc6-level embedding function is computed for both patches. Because there is limited capacity for joint reasoning—i.e., only two layers receive input from both patches—we expect the network to perform the bulk of the semantic reasoning for each patch separately. When designing the network, we followed AlexNet where possible.

To obtain training examples given an image, we sample the first patch uniformly, without any reference to image content. Given the position of the first patch, we sample the second patch randomly from the eight possible neighboring locations as in Figure 5.2.

5.2.1 Avoiding “trivial” solutions

When designing a pretext task, care must be taken to ensure that the task forces the network to extract the desired information (high-level semantics, in our case), without taking “trivial” shortcuts. In our case, low-level cues like boundary patterns or textures continuing between patches could potentially serve as such a shortcut. Hence, for the relative prediction task, it was important to include a gap between patches (in our case, approximately half the patch width). Even with the gap, it is possible that long lines spanning neighboring patches could give away the correct answer. Therefore, we also randomly jitter each patch location by up to 7 pixels (see Figure 5.2).

However, even these precautions are not enough: we were surprised to find that, for
Figure 5.4: Examples of patch clusters obtained by nearest neighbors. The query patch is shown on the far left. Matches are for three different features: fc6 features from a random initialization of our architecture, AlexNet fc7 after training on labeled ImageNet, and the fc6 features learned from our method. Queries were chosen from 1000 randomly-sampled patches. The top group is examples where our algorithm performs well; for the middle AlexNet outperforms our approach; and for the bottom all three features work well.

some images, another trivial solution exists. We traced the problem to an unexpected culprit: chromatic aberration. Chromatic aberration arises from differences in the way the lens focuses light at different wavelengths. In some cameras, one color channel (commonly green) is shrunk toward the image center relative to the others [6, p. 76]. A ConvNet, it turns out, can learn to localize a patch relative to the lens itself (see Section 5.3.2) simply by detecting the separation between green and magenta (red + blue). Once the network learns the absolute location on the lens, solving the relative location task becomes trivial. To deal with this problem, we experimented with two types of pre-processing. One is to shift green and magenta toward gray (‘projection’). Specifically, let $a = [-1, 2, -1]$ (the ‘green-magenta
We can recover image layout automatically. Cannot recover layout with color removed.

Figure 5.5: We trained a network to predict the absolute \((x, y)\) coordinates of randomly sampled patches. Far left: input image. Center left: extracted patches. Center right: the location the trained network predicts for each patch shown on the left. Far right: the same result after our color projection scheme. Note that the far right patches are shown after color projection; the operation’s effect is almost unnoticeable.

color axis’ in RGB space). We then define \(B = I - a^T a/(aa^T)\), which is a matrix that subtracts the projection of a color onto the green-magenta color axis. We multiply every pixel value by \(B\). An alternative approach is to randomly drop 2 of the 3 color channels from each patch (‘color dropping’), replacing the dropped colors with Gaussian noise (standard deviation \(\sim 1/100\) the standard deviation of the remaining channel). For qualitative results, we show the ‘color-dropping’ approach, but found both performed similarly; for the object detection results, we show both results.

**Implementation Details:** We use Caffe [57], and train on the ImageNet [19] 2012 training set (1.3M images), using only the images and discarding the labels. First, we resize each image to between 150K and 450K total pixels, preserving the aspect-ratio. From these images, we sample patches at resolution 96-by-96. For computational efficiency, we only sample the patches from a grid like pattern, such that each sampled patch can participate in as many as 8 separate pairings. We allow a gap of 48 pixels between the sampled patches in the grid, but also jitter the location of each patch in the grid by \(-7\) to \(7\) pixels in each direction. We preprocess patches by (1) mean subtraction (2) projecting or dropping colors (see above), and (3) randomly downsampling some patches to as little as 100 total pixels, and then upsampling it, to build robustness to pixelation. When applying simple SGD to train the network, we found that the network predictions would degenerate to a uniform prediction over the 8 categories, with all activations for fc6 and fc7 collapsing to 0. This meant that the optimization became permanently stuck in a saddle point where it ignored the input from the lower layers (which helped minimize the variance of the final output), and therefore that the net could not tune the lower-level features and escape the saddle point. Hence, Our final implementation employs batch normalization [55], which forces the network activations to vary across examples. We also find that high momentum values (e.g. .999) accelerated learning. For experiments, we use a ConvNet trained on a K40 GPU for approximately four weeks.

5.3 Experiments

We first demonstrate the network has learned to associate semantically similar patches, using simple nearest-neighbor matching. We then apply the trained network in two domains.
Table 5.1: Results on VOC-2007. R-CNN performance with our unsupervised pre-training is 5% MAP better than training from scratch, but still 8% below pre-training with ImageNet label supervision.

First, we use the model as “pre-training” for a standard vision task with only limited training data: specifically, we use the VOC 2007 object detection. Second, we evaluate visual data mining, where the goal is to start with an unlabeled image collection and discover object classes. Finally, we analyze the performance on the layout prediction “pretext task” to see how much is left to learn from this supervisory signal.

5.3.1 Nearest Neighbors

Recall our intuition that training should assign similar representations to semantically similar patches. In this section, our goal is to understand which patches our network considers similar. We begin by sampling random 96x96 patches, which we represent using fc6 features (i.e. we remove fc7 and higher shown in Figure 5.3, and use only one of the two stacks). We find nearest neighbors using normalized correlation of these features. Results for some patches (selected out of 1000 random queries) are shown in Figure 5.4. For comparison, we repeated the experiment using fc7 features from AlexNet trained on ImageNet (obtained by upsampling the patches), and using fc6 features from our architecture but without any training (random weights initialization). As shown in Figure 5.4, the matches returned by our feature often capture the semantic information that we are after, matching AlexNet in terms of semantic content (in some cases, e.g. the car wheel, our matches capture pose better). Interestingly, in a few cases, random (untrained) ConvNet also does reasonably well.

5.3.2 Aside: Learnability of Chromatic Aberration

We noticed in early nearest-neighbor experiments that some patches retrieved match patches from the same absolute location in the image, regardless of content, because those patches displayed similar aberration. To further demonstrate this phenomenon, we trained a network to predict the absolute \((x, y)\) coordinates of patches sampled from ImageNet. While the overall accuracy of this regressor is not very high, it does surprisingly well for some images: for the top 10% of images, the average (root-mean-square) error is .255, while chance performance (always predicting the image center) yields a RMSE of .371. Figure 5.5 shows one such result. Applying the proposed “projection” scheme increases the error on the top 10% of images to .321.
5.3.3 Object Detection

Previous work on the Pascal VOC challenge [30] has shown that pre-training on ImageNet (i.e., training a ConvNet to solve the ImageNet challenge) and then “fine-tuning” the network (i.e., re-training the ImageNet model for PASCAL data) provides a substantial boost over training on the Pascal training set alone [2, 44]. However, as far as we are aware, no works have shown that unsupervised pre-training on images can provide such a performance boost, no matter how much data is used.

Since we are already using a ConvNet, we adopt the current state-of-the-art R-CNN pipeline [44]. R-CNN works on object proposals that have been resized to 227x227. Our algorithm, however, is aimed at 96x96 patches. We find that downsampling the proposals to 96x96 loses too much detail. Instead, we adopt the architecture shown in Figure 5.6. As above, we use only one stack from Figure 5.3. Second, we resize the convolution layers to operate on inputs of 227x227. This results in a pool5 that is 7x7 spatially, so we must convert the previous fc6 layer into a convolution layer (which we call conv6) following [76]. Note our conv6 layer has 4096 channels, where each unit connects to a 3x3 region of pool5. A conv layer with 4096 channels would be quite expensive to connect directly to a 4096-dimensional fully-connected layer. Hence, we add another layer after conv6 (called conv6b), using a 1x1 kernel, which reduces the dimensionality to 1024 channels (and adds a nonlinearity). Finally, we feed the outputs through a pooling layer to a fully connected layer (fc7) which in turn connects to a final fc8 layer which feeds into the softmax. We fine-tune this network according to the procedure described in [44] (conv6b, fc7, and fc8 start with random weights), and use fc7 as the final representation. We do not use bounding-box regression, and take the appropriate results from [44] and [2].

Table 5.1 shows our results. Our architecture trained from scratch (random initialization) performs slightly worse than AlexNet trained from scratch. However, our pre-training makes up for this, boosting the from-scratch number by 6% MAP, and outperforms an AlexNet-style model trained from scratch on Pascal by over 5%. This puts us about 8% behind the performance of R-CNN pre-trained with ImageNet labels [44]. This is the best result we are aware of on VOC 2007 without using labels outside the dataset. We ran additional baselines initialized with batch normalization, but found they performed worse than the ones shown.

To understand the effect of various dataset biases [118], we also performed a preliminary experiment pre-training on a randomly-selected 2M subset of the Yahoo/Flickr 100-million Dataset [117], which was collected entirely automatically. The performance after fine-tuning is slightly worse than Imagenet, but there is still a considerable boost over the from-scratch model. We also performed a preliminary experiment with a VGG-style [105]...
Figure 5.7: Object clusters discovered by our algorithm. The number beside each cluster
does its ranking, determined by the fraction of the top matches that geometrically ver-
ified. For all clusters, we show the raw top 7 matches that verified geometrically. The full
ranking is available on our project webpage.

5.3.4 Visual Data Mining

Visual data mining [25, 92, 95, 106], or unsupervised object discovery [46, 98, 108], aims to
use a large image collection to discover image fragments which happen to depict the same
semantic objects. Applications include dataset visualization, content-based retrieval, and
tasks that require relating visual data to other unstructured information (e.g. GPS coor-
dinates [25]). For automatic data mining, our approach from section 5.3.1 is inadequate:
although object patches match to similar objects, textures match just as readily to similar
textures. Suppose, however, that we sampled two non-overlapping patches from the same
object. Not only would the nearest neighbor lists for both patches share many images, but
within those images, the nearest neighbors would be in roughly the same spatial configu-
ration. For texture regions, on the other hand, the spatial configurations of the neighbors
would be random, because the texture has no global layout.

To implement this, we first sample a constellation of four adjacent patches from an im-

age (we use four to reduce the likelihood of a matching spatial arrangement happening by chance). We find the top 100 images which have the strongest matches for all four patches, ignoring spatial layout. We then use a type of geometric verification \cite{13} to filter away the images where the four matches are not geometrically consistent. Because our features are more semantically-tuned, we can use a much weaker type of geometric verification than \cite{13}. Finally, we rank the different constellations by counting the number of times the top 100 matches geometrically verify.

**Implementation Details:** To compute whether a set of four matched patches geometrically verifies, we first compute the best-fitting square $S$ to the patch centers (via least-squares), while constraining that side of $S$ be between $2/3$ and $4/3$ of the average side of the patches. We then compute the squared error of the patch centers relative to $S$ (normalized by dividing the sum-of-squared-errors by the square of the side of $S$). The patch is geometrically verified if this normalized squared error is less than 1. When sampling patches do not use any of the data augmentation preprocessing steps (e.g. downsampling). We use the color-dropping version of our network.

We applied the described mining algorithm to Pascal VOC 2011, with no pre-filtering of images and no additional labels. We show some of the resulting patch clusters in Figure 5.7. The results are visually comparable to our previous work in chapter 4, although we discover a few objects that were not found in chapter 4, such as monitors, birds, torsos, and plates of food. The discovery of birds and torsos—which are notoriously deformable—provides further evidence for the invariances our algorithm has learned. We believe we have covered all objects discovered in chapter 4 with the exception of (1) trusses and (2) railroad tracks without trains (though we do discover them with trains). For some objects like dogs, we discover more variety and rank the best ones higher. Furthermore, many of the clusters shown in chapter 4 depict gratings (14 out of the top 100), whereas none of none of the ones here do (though two of our top hundred depict diffuse gradients). As in chapter 4, we often re-discover the same object multiple times with different viewpoints, which accounts for most of the gaps between ranks in Figure 5.7. The main disadvantages of this algorithm relative to chapter 4 are 1) some loss of purity, and 2) that we cannot currently determine an object mask automatically (although one could imagine dynamically adding more sub-patches to each proposed object).

To ensure that our algorithm has not simply learned an object-centric representation due to the various biases \cite{118} in ImageNet, we also applied our algorithm to 15,000 Street View images from Paris (following \cite{25}). The results in Figure 5.8 show that our representation captures scene layout and architectural elements. For this experiment, to rank clusters, we use the de-duplication procedure originally proposed in \cite{25}.

**Quantitative Results**

As part of the qualitative evaluation, we applied our algorithm to the subset of Pascal VOC 2007 selected in \cite{106}: specifically, those containing at least one instance of bus, dining table, motorbike, horse, sofa, or train, and evaluate via a purity coverage curve following chapter 4. We select 1000 sets of 10 images each for evaluation. The evaluation then sorts the sets by purity: the fraction of images in the cluster containing the same category. We generate the curve by walking down the ranking. For each point on the curve, we plot average purity of all sets up to a given point in the ranking against coverage: the fraction of images in the dataset that are contained in at least one of the sets up to that point. As shown in Figure 5.9, we have gained substantially in terms of coverage, suggesting increased invariance...
Figure 5.8: Clusters discovered and automatically ranked via our algorithm (§5.3.4) from the Paris Street View dataset.

for our learned feature. However, we have also lost some highly-pure clusters compared to chapter 4—which is not very surprising considering that our validation procedure is considerably simpler.

**Implementation Details:** We initialize 16,384 clusters by sampling patches, mining nearest neighbors, and geometric verification ranking as described above. The resulting clusters are highly redundant. The cluster selection procedure in chapter 4 relies on a likelihood ratio score that is calibrated across clusters, which is not available to us. To select clusters, we first select the top 10 geometrically-verified neighbors for each cluster. Then we iteratively select the highest-ranked cluster that contributes at least one image to our coverage score. When we run out of images that aren’t included in the coverage score, we choose clusters to cover each image at least twice, and then three times, and so on.

5.3.5 **Accuracy on the Relative Prediction Task Task**

Can we improve the representation by further training on our relative prediction pretext task? To find out, we briefly analyze classification performance on pretext task itself. We sampled 500 random images from Pascal VOC 2007, sampled 256 pairs of patches from each, and classified them into the eight relative-position categories from Figure 5.2. This gave an accuracy of 38.4%, where chance performance is 12.5%, suggesting that the pretext task is quite hard (indeed, human performance on the task is similar). To measure possible overfitting, we also ran the same experiment on ImageNet, which is the dataset we used for training. The network was 39.5% accurate on the training set, and 40.3% accurate on the validation set (which the network never saw during training), suggesting that little overfitting has occurred.

One possible reason why the pretext task is so difficult is because, for a large fraction of
patches within each image, the task is almost impossible. Might the task be easiest for image regions corresponding to objects? To test this hypothesis, we repeated our experiment using only patches sampled from within Pascal object ground-truth bounding boxes. We select only those boxes that are at least 240 pixels on each side, and which are not labeled as truncated, occluded, or difficult. Surprisingly, this gave essentially the same accuracy of 39.2%, and a similar experiment only on cars yielded 45.6% accuracy. So, while our algorithm is sensitive to objects, it is almost as sensitive to the layout of the rest of the image.

5.4 Proposed: Leveraging the 2D structure of objects

While the representations learned from our relative-prediction pretext task seem to show a surprising amount of semantic information, they clearly capture less than those trained with semantic labels. What is missing? From the deep network’s perspective, recognizing more objects can only lead to better performance on the pretext task. One possibility is that the network, in addition to learning objects, also learns semantically meaningless cues that nevertheless improve performance on the pretext task. Chromatic aberration is an example that certainly occurs, but we hypothesize that the network is learning to represent texture gradients, converging parallel lines, and other low-level cues. These may be a problem because the networks we use have finite capacity (i.e. finite VC dimension), and hence the network may choose to learn this nuisance information instead of learning object semantics, if doing so will result in better performance on the pretext task. Even if we expand the network to give it higher capacity, this nuisance information will still be represented when we attempt to transfer the representation to another task, and the curse of dimensionality means that we may require more training data to transfer or fine-tune the representation.

To test this hypothesis, we aim to update the pretext task such that it focuses the network on regions that contain objects, so that simple texture gradients, lines, and scene statistics are less helpful compared to object semantics. In a sense, we want to avoid the patch pairings that are too difficult and waste space in the representation. In section 5.3.4, we argued...
that objects can be distinguished from “stuff” because they are made of parts, each of which corresponds to one specific location on the object. To leverage this, we wish to modify the loss function such that only parts that can be localized to specific 2D coordinates on an object contribute to lowering the loss.

To accomplish this, instead of localizing patches relative to one another as before, we will instead force the network to produce a single 2D coordinate for each patch, without reference to any other patches in the image. Note that such a prediction can, in theory, lead to exactly the same kind of predictions we have already done: subtracting the predictions of two patches will yield a prediction of the relative offset between the patches. We can then verify the predicted offset by measuring whether it is consistent with the actual offset between a pair of patches. This leads to a simple objective function: we train the network so that the predicted coordinates for different patches are consistent with the actual offset between them. Note that this means that the origin of the coordinate system can be arbitrary: we only measure whether the offsets are consistent with the truth. The difficulty, however, is that it is not possible to create one coordinate system that can be consistently predicted across an entire image. To illustrate this, say that we have a person to the left of a car in an image. A coordinate system which is consistent and high scoring across the whole image may, for example, give a low x-coordinate to the person and high x-coordinate to patches in the car. If we switch the locations of the person and car, however, the predictions that were good before are now bad. In fact, if we measure squared error—which penalizes large errors much more harshly than small ones—the overall loss may be worse than if the network had simply predicted the same x-coordinate everywhere.

Hence, we propose to separate different objects through an implicit segmentation, and only penalize errors between patches that occur on the same object. The implementation is straightforward: before the network predicts the location of two patches in a pair, it first produces a confidence score, which measures how likely it is that its prediction for the pair will be correct. The loss will be positive if the relative prediction is worse than some threshold and negative if the relative prediction is better than the threshold. The loss will be multiplied by the network’s confidence in its prediction. Hence, if the patches come from two separate objects, the network can easily zero out the loss, and neither gain nor lose. However, if the patches are on the same object, the network can produce a high confidence score, and be rewarded for a correct prediction.

Evaluation for this algorithm will be similar to the evaluations earlier in this chapter: the algorithm should produce better nearest neighbors for patches depicting object parts, and should lead to higher performance on standard vision tasks like Pascal VOC.
Chapter 6

Proposed: From Image Classification to Image Generation

As demonstrated in Figure 4.5, image representations can be better understood and visualized by generating new images. Newly generated imagery demonstrates what the representation captures: arguably, a representation which can be used to generate novel, photorealistic scenes containing an object must capture everything about that object’s appearance. Of course, such a model would have immediate applications in computer graphics. Photorealistic image generation, however, has thus far remained in the domain of physics-based systems, due to the incredible complexity of scenes and the extremely nonlinear nature of the imaging process.

Most existing generative models have been limited to relatively simple domains like handwritten digits or street numbers. Some interesting exceptions, however, can generate textures which are perceptually similar to realistic textures, given similar textures as input [42]. “Deep Dream” can hallucinate features of complex objects in existing images [82]. These approaches, however, do not rely on pure generative modeling; instead, they begin with networks that were trained discriminatively. They establish some objective function over these discriminative representations, and backpropagate that objective function to the pixels, repeatedly updating the image until the desired pattern appears. These works currently have severe limitations, partially due to the fact that discriminative representations pick up on specific features rather than the full object. For instance, Gatys et al. [42] requires a seed texture because it needs to first builds a complex, high-dimensional representation of that texture’s statistics. Furthermore, while Deep Dream [82] can generate object and scene fragments, it does not generate images that are overall coherent. However, in this work, we will argue that the basic algorithm in these works can be extended to include additional terms that enhance image realism using unsupervised data.

These two works maximize remarkably different objective functions when adjusting the images. Gatys et al. [42] maximizes the similarity of the image’s texture representation to the representation of the seed texture, and Deep Dream [82] simply maximizes the norm of the output features. Yet these two objectives could easily be combined: we simply backpropagate the gradients from both to the same image.

Naturally, the objective functions used to generate the image are very important in determining the final image. In a preliminary experiment, added two additional terms to
Figure 6.1: Two images synthesized by our algorithm, starting from noise, with no reference images other than those used for training the deep networks. Left, the algorithm maximized the likelihood of the scorpion category from ImageNet, and right the red fox category.

the objective of Deep Dream [82]: 1) maximizing the likelihood that GoogLeNet assigns to the presence of one particular object category, and 2) a term which aims to force the conv4-level features of GoogLeNet to have similar statistics to the statistics seen in conv4 representations of natural images. To measure 2, we built a simple denoising autoencoder of GoogLeNet’s conv4 features, using ImageNet images as training data. In order to update the image at generation time, we first encoded the image using GoogLeNet’s conv4 features, then passed them through the denoising autoencoder to obtain reconstructed “denoised” conv4 features. We then treated the difference between the current conv4 features and the reconstruction as an error signal. We found that the original Deep Dream objective function [82] (i.e. maximizing the norm of the conv4-level features) was still useful because it seemed to drive the network toward regions of the space that it understood (which is important since discriminative models don’t necessarily represent everything). Example images generated starting from random noise are shown in Figure 6.1.

For graphics applications, we aim to give the user as much control as possible while still generating realistic images. This means having a large number of objective functions which factorize the different aspects of images that the user may care about. For example, we might factorize a street scene into style and content: i.e. have one objective which captures scene layout (content) and another which captures the architectural style of the city where the image was taken (style). This would allow a user to hold the content constant in an image while altering the style, i.e., make a scene look like a different city. We argue that both of these can be trained from weak supervision: e.g., a model trained to predict city labels to capture style, and another trained for the relative patch prediction of Chapter 5 to capture layout.
Chapter 7

Conclusion

In this thesis, we have proposed and analyzed several methods for learning visual representations from images collections where the labels are either weak or nonexistent. We have shown that image-level labels like GPS tags or scene categories allow us to build such representations, and also that context can serve as a similarly informative supervisory signal when no labels are present. In the future, we aim to learn even more powerful representations, first by focusing the algorithm on coherent objects, and second by extending them to fully generative models.

Completed Work

We have introduced methods for learning detectors for smaller “visual elements” like objects and object parts given weak image-level labels like GPS tags and scene categories, and shown that they can be used for both dataset visualization and for scene classification [22]. We have also shown that image context can be used to cluster objects [23] and even learn deep representations that can be used for other tasks like object detection [24].

Proposed Work

We propose two novel directions. First, we aim to further improve algorithms for learning visual representations from within-image context. We will pose a new pretext task where the network must infer and segments out objects, and then predict a single, absolute location for each patch, using a coordinate system that is specific to the object depicted in the patch. Second, we propose a generative model which will enable visualization of deep representations and the synthesis of novel imagery for graphics applications. The generative model will trained through weak supervisory signals to be aware of scene attributes like style and content, and will be trained through unsupervised learning to produce overall realistic images.

Tentative Timeline

1. Fall 2015: Unsupervised representation learning with a novel pretext task
2. Winter/Spring 2016: Generative modeling
3. May 2016: Defend
Bibliography


