Supervision Beyond Manual Annotations for Learning Visual Representations

Carl Doersch

April 2016
CMU-ML-16-102

Machine Learning Department
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA

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

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Machine Learning

This research was sponsored by a Google Fellowship, an NDSEG Fellowship, National Science Foundation grant number IIS0905402, and Department of the Air Force contract number FA865012C7212

Keywords: pretext tasks, self-supervised learning, computer vision, unsupervised learning, weakly-supervised learning, context, unsupervised object discovery, visual data mining, visual summarization, computational geography

Acknowledgements

First and foremost, I would like to thank my advisors Alyosha Efros and Abhinav Gupta. Both have been an invaluable source of insight, ideas, and practical guidance from beginning to end. They have consistently put in their time and energy through both successes and failures. I'm thankful that I had the chance to work with them, and I'm proud of the work we have done together over the last 6 years.

I would also like to thank Erik Learned-Miller, Tai-Sing Lee, and David Plaut, who worked with me as a fresh undergraduate, taught me the basics of research, and shaped my career as a vision scientist. I thank Florian Schroff, Taehee Lee, and Hartwig Adam for mentoring me during my internship at Google. I have also had the privilege of working and discussing with many other researchers, including my thesis committee members Trevor Darrell and Ruslan Salakhutdinov, as well as Josef Sivic, Martial Hebert, and Jitendra Malik.

I have also been fortunate to be surrounded by many supportive friends and colleagues at CMU, UC Berkeley, and elsewhere. I would like to thank Abhinav Shrivastava, David Fouhey, Jacob Walker, Philipp Krähenbül, Shiry Ginosar, Saurabh Singh, Tinghui Zhou, Yong Jae Lee, Ishan Misra, Ed Hsiao, Scott Satkin, Jun-Yan Zhu, and Tomasz Malisiewicz for many helpful discussions (Abhinav especially for lending me a bed after my apartment caught fire). A special thanks goes to everyone who helped me revitalize Student Pugwash at CMU, including Kunal Ghosh, Katy McKeough, Rob Macedo, Joe Vukovich, and Maddi Brumbaugh. Thanks also to James Carver, Joel Lu, and Derek Brown for lasting support and friendship.

Finally, I thank my parents Bob and Candace, and my sister Karen. I couldn't have done this without your many years of love and support.

Abstract

For both humans and machines, understanding the visual world requires relating new percepts with past experience. We argue that a good visual representation for an image should encode what makes it similar to other images, enabling the recall of associated experiences. Current machine implementations of visual representations can capture some aspects of similarity, but fall far short of human ability overall. Even if one explicitly labels objects in millions of images to tell the computer what should be considered similar—a very expensive procedure—the labels still do not capture everything that might be relevant.

This thesis shows that one can often train a representation which captures similarity beyond what is labeled in a given dataset. That means we can begin with a dataset that has uninteresting labels, or no labels at all, and still build a useful representation. To do this, we propose to using *pretext tasks*: tasks that are not useful in and of themselves, but serve as an excuse to learn a more general-purpose representation. The labels for a pretext task can be inexpensive or even free. Furthermore, since this approach assumes training labels differ from the desired outputs, it can handle output spaces where the correct answer is ambiguous, and therefore impossible to annotate by hand.

The thesis explores two broad classes of supervision. The first is weak image-level supervision, which is exploited to train mid-level discriminative patch classifiers. For example, given a dataset of street-level imagery labeled only with GPS coordinates, patch classifiers are trained to differentiate one specific geographical region (e.g. the city of Paris) from others. The resulting classifiers each automatically collect and associate a set of patches which all depict the same distinctive architectural element. In this way, we can learn to detect elements like balconies, signs, and lamps without annotations. The second type of supervision requires no information about images other than the pixels themselves. Instead, the algorithm is trained to predict the *context* around image patches. The context serves as a sort of weak label: to predict well, the algorithm must associate similar-looking patches which also have similar contexts. After training, the feature representation learned using this within-image context indeed captures visual similarity across images, which ultimately makes it useful for real tasks like object detection and geometry estimation.

Contents

Chapter 1

Introduction and Overview

Computer vision has begun to make substantial progress: for instance, recent work showed computers can classify internet images into one of 1,000 categories as well as people [80], which was far from true even five years ago. But do computers really *understand* images? Consider the image in Figure 1.1. Can you recognize the city where it was taken? Given images like this one, people familiar with this city can tell that it's Paris, even though most have never been explicitly trained to distinguish between cities. So how do people do this? One important answer is that small details like street signs, balconies, or window railings look different in Paris than they do in other cities, but within Paris they mostly have a common style. What differentiates Paris style from other cities is hard to describe with the sort of labels that human annotators could provide easily: even though the windows in Figure 1.1 are clearly similar, it is not clear that all "Paris windows" fit into a single category. We argue that much of a human's understanding of images is not well captured by the segments, categories, and bounding boxes the vision community tends to use as training data. Hence, in this work, we seek an alternative way for computers to learn visual concepts. We expect such an alternative *should* exist because humans seemingly learn to see without millions of labels, and certainly with no bounding boxes or segmentations. How can people guide their learning when they have no clearly-defined task? And can machines do the same thing?

Our ultimate goal is machine understanding of images, which means endowing computers with visual representations that are informative, robust, and versatile. That is, the representations should capture all relevant information in a scene, as much information as a human would. They should not fail to recognize objects, attributes, places, and other useful scene properties due to variations in appearance from lighting, pose, occlusion, and so forth. And finally, they should organize the information in a way that enables as many tasks as possible, including tasks where little or no training data is available. While we might theoretically build representations like this with huge manually-labeled datasets describing exactly the representation we want, we expect that such detailed annotations would be prohibitively expensive. We wish to avoid defining and labeling the representation by hand, inspired by the simple fact that humans and animals do not seem to require it.

Hence, in this work, we examine how other sources of supervision besides manual annotation might be used to learn a representation. The idea is that, given the right task, the computer can learn *on its own* to represent useful semantic properties of the visual world. A good task, we argue, will require semantic understanding as a prerequisite, even if the outputs needed for the task are not themselves semantic. We call such tasks "pretext tasks",

Figure 1.1: How do our minds represent a city scene? People who have been to Paris can often pick out discriminative elements of the architecture like these, despite never being explictly educated to do so. Hence, a first step in our work is to investigate how a computer might learn this kind of "visual element" from a large database of street-level images, when the only label given is the city name.

since we do not necessarily care about solving the tasks themselves, but rather, the tasks are an excuse for learning something else. The key contributions of this thesis are an analysis of several different pretext tasks, as well as algorithms that train useful representations from those tasks, and methods for doing useful work with the resulting representations[1]. Specifically:

- Our first pretext task challenges the computer to predict image-level GPS tags. We show that, in response, the computer learns specific and recognizable "Visual Elements" of the style of cities, which summarize the style in ways that are easy for humans to interpret.
- We use scene categorization as a pretext task for learning objects and other discriminative elements of indoor scenes, and demonstrate how a discriminative extension of the classic Mean Shift algorithm [60] can be used to learn such elements.
- Our final pretext task requires the computer to predict the context around patches. We show that this task encourages the algorithm to learn to detect objects, since objects put strong constraints on what can occur in the surrounding context.
- We train deep neural networks using unlabled image context, and demonstrate that this "unsupervised" learning can result in improvements on real image tasks like object detection and geometry estimation.

1.1 What is a Representation?

Before we describe learning procedures, we should first describe the desired result: a working visual representation. We argue that it contains much more than the bounding boxes,

[1]These contributions were originally reported in [39–41, 43, 44].

keypoints, and segmentations which are the usual target outputs of computer vision systems! Describing what exactly humans do with visual information has fascinated philosophers for centuries. For example, David Hume wrote in his *Treatise of Human Nature* in 1740 [91]:

> The first circumstance, that strikes my eye, is the great resemblance betwixt our impressions and ideas in every other particular, except their degree of force and vivacity. The one seem to be in a manner the reflexion of the other; so that all the perceptions of the mind are double, and appear both as impressions and ideas. When I shut my eyes and think of my chamber, the ideas I form are exact representations of the impressions I felt; nor is there any circumstance of the one, which is not to be found in the other. In running over my other perceptions, I find still the same resemblance and representation. Ideas and impressions appear always to correspond to each other.

Here, *impressions* refers, roughly, to sensory data, whereas *ideas* or *representations* are something that exists purely in the mind, which appear automatically even when the mind is not performing any specific task, and persist after the eyes are closed. But what is contained in this representation? Hume continues:

> Upon a more accurate survey I find I have been carried away too far by the first appearance, and that I must make use of the distinction of perceptions into simple and complex, to limit this general decision, that all our ideas and impressions are resembling. I observe, that many of our complex ideas never had impressions, that corresponded to them, and that many of our complex impressions never are exactly copied in ideas. I can imagine to myself such a city as the New Jerusalem, whose pavement is gold and walls are rubies, though I never saw any such. I have seen Paris; but shall I affirm I can form such an idea of that city, as will perfectly represent all its streets and houses in their real and just proportions?

Representations, it seems, are not mere copies of the visual world; rather, the visual world has been transformed, and information has been lost. This seems reasonable: if we want to recognize that a window is constructed in the style of Paris windows, we do not need to remember every blemish in the stone of every Paris window we have seen. But does this transformation accomplish something more than simple compression of information stored in our memories? Hume argues that the purpose of the ideas (the basic unit of our representation) is that it enables us to form associations:

> The qualities, from which this association arises, and by which the mind is after this manner conveyed from one idea to another, are three, viz. RESEMBLANCE, CONTIGUITY in time or place, and CAUSE and EFFECT. I believe it will not be very necessary to prove, that these qualities produce an association among ideas, and upon the appearance of one idea naturally introduce another. It is plain, that in the course of our thinking, and in the constant revolution of our ideas, our imagination runs easily from one idea to any other that resembles it, and that this quality alone is to the fancy a sufficient bond and association.

Hume's statement here seems to pre-suppose that a good representation makes apparent the resemblance (or similarity) between different objects and scenes we see. For example, it is likely readily apparent that the Paris windows shown in Figure 1.1 have something

Figure 1.2: Given a large database of Google Street View images, we might hope that awnings, traffic lights, and pedestrians would be grouped close together in feature space. However, here we show nearest neighbors in the highly successful HOG [34] feature space for 3 such patches (the query patch is on the left).

in common, even if they are not identical. This kind of association is necessary for recognizing that past memories are relevant when, for example, we notice that a building we haven't seen before uses the Paris architectural style. Admittedly, Hume did not have the tools to explore *what* about the mind's representation allows us to infer "resemblance," and thus this term (as well as "contiguity" and "cause and effect") remain poorly defined throughout the treatise. Today, however, we know that defining visual resemblance is much more difficult than we would expect given how effortlessly the human brain to computes it. Even after decades of work, we can apply one of the the best hand-designed similarity metrics (HOG [34]) to a large dataset of image patches and fail associate some patches with others that are semantically similar, as shown in Figure 1.2. The incredible difficulty of programming a notion of similarity suggests that perhaps we need to take a different approach than defining it by hand.

One likely answer is that similarity metrics, the resulting associations between different visual experiences, and the representation in general are all learned from data. That is, we don't need to define similarity or the representation explicitly in code, but rather, we need to define some rule that allows us to optimize a representation until it gives the kind of associations we want. State-of-the-art work in this field—deep neural networks—currently learns almost every aspect of its representation [111], even low-level image filters that can replace hand-designed features like the HOG features described above. This has led to remarkable improvements in performance in tasks like large-scale image classification [35]. However, the key in this approach is that human annotators explicitly tell the computer which kinds of images must be associated, by dividing the visual world into 1000 different categories of objects, and providing more than a million pre-labeled images to the algorithm. Hence, this result is not only expensive in terms of human labor; it also makes strong assumptions about what we care to label in images. Furthermore, it can't explain how learning happens in humans, since humans do not need such large collections of labeled images.

Interestingly, work concurrent with this thesis has shown that a deep neural network representations [111] trained to perform the large-scale classification task posed by ImageNet [35] are useful for a variety of visual tasks, such as detecting and localizing ob-

jects [66], segmentation [130], image retrieval [68], geometry estimation [219], and others. This result suggests that the deep representation is capturing something beyond the categories that it is initially trained on. Why does this happen? One hypothesis is that in order to classify between different object categories, the deep net learns to partially solve many other problems and extract other types of information, because doing so is useful for the final classification. For example, to learn to differentiate between different breeds of dogs, it helps to be able to localize discriminative parts like the head, so that those parts can be compared directly. As a side effect, this capability might be what helps R-CNN [66] localize objects when built on top of the representation learned from ImageNet. So, if the representation and the task are not tied so closely together, then are there other tasks which prompt learning algorithms to discover useful concepts beyond what is labeled?

1.2 Our Approach

This thesis aims to begin to address the question of whether there exists other ways to train strong visual representations besides giving the computer exactly the label we want. The repercussions are twofold. First, we want to train good representations with less human effort. After all, 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. Even if we don't ultimately discard all manual annotation, we might be able to learn more effective representations with cheap or free annotations that augment or even replace the expensive ones. Second, and more philosophically, we want to understand what makes a good representation. That is, we want to understand the circumstances that allow machine learning algorithms to generate general-purpose representations, which might ultimately even contribute to understanding how humans learn to see.

What tasks can prompt an algorithm to learn a good representation? The space of potential tasks is enormous: for example, given any function that we might compute on images, we can train an algorithm to approximate that function, or to approximate its inverse (Denoising Autoencoders [9, 208], for example, take the second approach). To try to narrow down the set of possible tasks, we take inspiration from Hume: the idea is that a good representation produces associations between ideas, i.e., associations between similar objects and similar scenes. Therefore a good task should be solved by that kind of association. Specifically, a good task will require similar outputs (predictions) for *semantically* similar images, whereas semantically different images should require different (or at least uncorrelated) predictions. In order to generalize well on such a task, the algorithm will thus learn to associate the semantically similar images: this allows it to transfer the correct prediction from one image to other similar images. It is not immediately obvious that there really exist *non-semantic* (i.e. easy-to-label) tasks where semantically similar images have semantically similar predictions. However, recent work in both psychology and computer science suggests that they might. In psychology, it has been shown that young children learn words more quickly if those words predict something about the context, i.e., they co-occur with another distinctive environmental cue [174]. Thus, environment prediction might be seen as a pretext task: a child learns that a given word—which may sound very different coming from different speakers—always means the same thing because it predicts something consistent about the environment. Recent results in the natural language processing have shown similar results: notably, that the context around words can train useful word repre-

sentations [5,28,137,151]. For instance, we can task the computer with predicting the words in the context of (i.e., a few words before and/or after) a given word. The learned representations are then useful for other tasks involving natural language interpretation [137]. In this case, context prediction serves pretext task to cause the model to associate related words, since they have similar contexts.

We argue that such pretext tasks exist for vision as well, and that some of them can be posed to the computer with minimal annotation effort. In this thesis, we divide such tasks broadly into *weakly supervised* methods, which use image-level labels, and *unsupervised* methods, which assume no information is available in an image dataset other than the pixels themselves.

Weakly Supervised Learning

Our work begins with the goal of learning to associate visually similar patches that are somehow informative, and therefore form useful building blocks for representations. One way to associate semantically similar images is that they often have similar metadata on the web. For example, assuming we have GPS tags for images, images with nearby GPS coordinates will often have something visually in common. In this thesis, we first aim to build a computer system that understands the look and feel of cities, and associates patches depicting similar components of city architecture. This work is inspired by the fact that humans can often point out small, distinctive elements of the architecture of cities they recognize, despite having no apparent source of labels for such elements. It is straightforward to download large datasets of images taken in different cities: for example, we can easily obtain millions of geotagged images from Google Street View. However, it is less clear how a computer can learn to describe and localize *what* is distinctive about the scenes, as people can. Even if we have a strong classifier that can differentiate between cities, it is not clear how to produce the sort of associations that might be useful to an architect, urban historian, or graphics designer, which explains what in the image gives it the distinctive look, and how those elements are connected to other scenes elsewhere in the city. Retrieving patches like those shown in Figure 1.1 would be useful, but annotating training examples would be not only labor-intensive, but also ambiguous for the annotators. How large should the bounding boxes extend on the facade? Does the fact that one window has a larger railing than another indicate a sub-category of window that should be clustered separately? We argue that that the computer should be able to answer this kind of question automatically. Thus, we are faced with a "weakly supervised" pretext task: the labels are only weakly correlated with the kind of representation we want. Our real goal is to learn something much richer than the labels we have.

Our approach is to group image 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 which city is depicted in the patch. We can implement a visual element by training a detector (i.e. a patch classifier) which fires only in one city, and yet fires on a visually coherent set of patches containing as many instances as possible from that city. In this thesis, we detail two separate formulations that enable this kind of visual element discovery without labels.

Our first approach relies on cross-validation clustering [190]. We initialize clusters of patches using simple nearest-neighbors retrieval. Then, for each cluster, we train a classifier to discriminate the top retrievals from one city from everything else. Given this trained

classifier, we can retrieve new top matches and re-train the classifier. We continue this procedure iteratively. An improved classifier improves the retrievals, and improved retrievals in turn improve the classifier. Over time, we find the classifier hones in on a visual concept that occurs frequently in the target city, and can be easily discriminated from similar concepts in other cities.

We find even better results by reformulating the above algorithm into a joint optimization over detectors and cluster memberships. We pose the visual element discovery as discriminative mode seeking, drawing connections to the the well-known and well-studied mean-shift algorithm. Given a weakly-labeled image collection, this method seeks regions of patch feature space where there are many patches from one city, and few patches from other cities.

We develop 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 standard task of scene classification, demonstrating strong performance on the MIT Indoor Scene-67 dataset, as well as several tasks related to visualizing the style of cities, such as understanding the city layouts and connecting similar elements across different cities.

Unsupervised Learning

Next, we turn to representation learning when no annotations are available. Unsupervised learning has been a longstanding goal in computer vision due to the promise of virtually infinite free training data available on the web. However, 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.

First, we aim to simply discover sets of image patches that all depict the same object, much like we tried to find patches depicting the same architectural element above. The ability of an object patch to predict the rest of the object (its context) is used as supervisory signal to help verify which patches belong together. There are three main components of the algorithm. First, we frame unsupervised clustering as a leave-one-out context prediction (i.e. generative) task. Second, we evaluate the quality of context prediction by statistical hypothesis testing between "thing" and "stuff" appearance models, only accepting that a set of patches forms an object if the "thing" model predicts better than the "stuff" model. Third, we use 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. In all, this work demonstrates that context can serve as a way to associate images that contain the same thing, much in the same way that labels do in a standard supervised setup.

Given this success, we next show how to use spatial context as a source of free and plentiful supervisory signal for training a deep visual representation, which efficiently and robustly summarizes the data in a way that is useful for real tasks. 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. Intuitively, doing well on this task will require the model to learn to detect objects and object parts, since without the semantics, it would often be impossible to recover the layout. We demonstrate that the feature representation learned using this within-image context prediction task is indeed able to capture visual similarity across images, and can be used for discovery much like our previous algorithm. Furthermore, we show that the learned representation is useful for standard vision tasks including object detection and geometry estimation. Notably, when used as pre-training for the R-CNN object detection pipeline [66], our features 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.

Chapter 2

Background

Though Hume and others described the basic intuition behind a visual representation centuries ago, it remained difficult to say anything rigorous about how they *work* until relatively recently. Advances have come gradually, both from psychology and neuroscience of animal vision, and more recently from computer science and machine learning. This chapter aims to provide a brief history of some of the important developments which contributed to the scientific community's understanding of visual representations, starting from neuroscience and psychology, and continuing through to the machine representations that are useful for real tasks today. While this is far from a complete account of the history of visual representations, our hope here is to provide historical context for the current work, highlighting works that have contributed to it, and to trace the origins of some of the questions that this thesis contributes to answering.

2.1 Early Theories from Psychology and Neuroscience

The view of the eye as a camera obscura, with a lens projecting an image onto the retina, is generally credited to Johannes Kepler in the early 1600s [211] (although even Da Vinci found parallels between the eye and the camera obscura 100 years earlier). This is, in some sense, the most basic possible representation: an image copied exactly into the body. This understanding of the eye gradually gained acceptance, but the question of what happened to that image afterward remained murky. After all, a flat image is a poor approximation to our perception of a 3D world populated by objects. Hermann von Helmholtz is generally credited with emphasizing the importance of *inference*, a mathematical process which happens unconsciously from an image, to infer properties like depth that are not immediatly present in the image. Helmholtz's theory was largely based on geometry, particularly binocular stereo. The origins of the idea that *learning* is an important part of the development of human representations is difficult to trace, but it appears Helmholtz was at least considering the idea. In his Treatise on Physiological Optics [209], Helmholtz discusses the importance of experience and learning in shaping the outcomes of this "unconscious inference," which was apparently a divisive topic during his time:

> Still to many physiologists and psychologists the connection between the sensation and the conception of the object usually appears to be so rigid and obligatory that they are not much disposed to admit that, to a considerable

> extent at least, it depends on acquired experience, that is, on psychic activity. On the contrary, they have endeavoured to find some mechanical mode of origin for this connection through the agency of imaginary organic structures. With regard to this question, all those experiences are of much significance which show how the judgment of the senses may be modified by experience and by training derived under various circumstances, and may be adapted to the new conditions. Thus, persons may learn in some measure to utilize details of the sensation which otherwise would escape notice and not contribute to obtaining any idea of the object. [...] For example, the spectacle of a person in the act of walking is a familiar sight. We think of this motion as a connected whole, possibly taking note of some of its most conspicuous singularities. But it requires minute attention and a special choice of the point of view to distinguish the upward and lateral movements of the body in a person's gait. We have to pick out points or lines of reference in the background with which we can compare the position of his head. But look through an astronomical telescope at a crowd of people in motion far away. Their images are upside down, but what a curious jerking and swaying of the body is produced by those who are walking about! Then there is no trouble whatever in noticing the peculiar motions of the body and many other singularities of gait; and especially differences between individuals and the reasons for them, simply because this is not the everyday sight to which we are accustomed. On the other hand, when the image is inverted in this way, it is not so easy to tell whether the gait is light or awkward, dignified or graceful, as it was when the image was erect.

Methods to explore what happened to the image after it was absorbed into the retina did not appear until the 1930s, when developments in electrodes made it possible to record the responses of single neurons [77]. In vision, microelectrode studies ultimately led to the discovery of retinal neurons that were sensitive to simple patterns like dots [8], and then to visual cortex neurons which were sensitive to oriented edges with some invariance to spatial position by Hubel and Wiesel [87, 89]. Hubel and Wiesel conjectured a hierarchical representation of neurons, where neurons deeper in the hierarchy were sensitive to increasingly complicated stimuli. This theory was confirmed by later studies which found, for example, hand detectors in inferotemporal cortex [74].

Despite the tremendous progress in understanding the basic contents of the brain's visual representation, there were two fundamental questions that remained unanswered. First, neuroscience could do relatively little to explain *how* the neurons detected the things they detected. The hand detector neuron apparently worked by receiving and assembling messages from neurons which detected the mid-level parts of hands; these in turn received messages from the edge detecting neurons. However, the mid-level "parts" were quite difficult to describe, and the mathematics that a neuron used to combine messages could not be observed directly. Furthermore, neuroscience could do almost nothing to explain how individual neurons decided what they should detect. It was not known how the visual cortex was constructed starting only from DNA, or how experiences like those described by Helmholtz could be incorporated into a learning system.

2.2 Computer Vision For Artificial Intelligence

By the 1960s, computers were powerful enough to begin processing images. Early work largely ignored learning, and instead focused on directly coding algorithms that could infer scene properties that might be useful for robotics. For example, early work at the MIT Artificial Intelligence laboratory aimed to infer 3D geometry starting from straight lines [75, 171]. Subsequent works presented algorithms to infer other properties of objects, such as surface reflectance [114] or more complex 3D shape using shading [86]. Fischler and Eischlager laid the groundwork for more semantic object detection with their Pictorial Structures model [56]. Progress was slow, mostly due to computational constraints. However, David Marr [135] points to another issue, the importance of which was not very widely recognized: the goals were poorly defined. Specifically, he writes regarding the papers that were published in the 1970s:

> There must exist an additional level of understanding at which the character of the information-processing tasks carried out during perception are analyzed and understood in a way that is independent of the particular mechanisms and structures that implement them in our heads. This was what was missing—the analysis of the problem as an information-processing task. Such analysis does not usurp an understanding at the other levels—of neurons or of computer programs—but it is a necessary complement to them [...] [I]f the notion of different types of understanding is taken very seriously, it allows the study of the information-processing basis of perception to be made *rigorous*. It becomes possible, by separating explanations into different levels, to make explicit statements about what is being computed and why and to construct theories stating that what is being computed is optimal in some sense or is guaranteed to function correctly. The ad hoc element is removed, and heuristic computer programs are replaced by solid foundations on which a real subject can be built.

Marr and others argued during the late 1970s that there were three levels of understanding that the field needed to strive for separately. At the highest level, we need to understand the basic goals, i.e., *what* must the system compute in the end. The next level is the algorithm and representation that is used to compute it, and the lowest level is the actual hardware implementation in terms of neurons or lines of code. The problem, Marr argued, was that all algorithms published up to that point had not done enough to separate the algorithm from the desired output, leading to a great deal of confusion about whether a given algorithm was accomplishing its task, and whether or not that task was worth accomplishing. In retrospect, there is certainly a kernel of truth to Marr's argument here, although it is difficult to say whether the field's progress since then is quite what he had in mind. Certainly the modern vision field has gotten better about posing tasks separately from the algorithms that solve them (allowing for rigorous comparisons between algorithm performance, which I believe most would argue has been critical to the success of the field). This separation—and notably the idea that it is non-trivial to define *what task* the algorithm should solve—is of central importance in this thesis, but again our view is probably not quite what Marr had in mind. We aim to show that we can define proxies for the tasks of interest, rather than defining the desired tasks themselves. In some sense, this understanding of the relationships between tasks is yet another level of understanding, above simply understanding what the algorithm should compute.

The separation of the task from the algorithm also paved the way for a development that Marr does not seem to have predicted: namely, that once the task was well defined, machine learning can overtake human programming of the algorithm-level understanding of computer vision. In the early days of computer vision, learning was scarcely mentioned, much less implemented. However, the computational neuroscience community had never forgotten the problem of learning. Even early on, it was clear that the visual processing algorithms wer not hard-coded in DNA. Rather, some form of learning was happening even at the lowest levels of the hierarchy. For instance, it was shown neurons in the visual cortex of cats learned to take input from only one eye if the other eye was sewn shut [90]. The earliest implementations of visual learning actually aimed to be neural models, aimed at understanding how a functioning visual system could develop in a brain. Most notably, Rosenblatt's work on the Perceptron (which, remarkably, he stacked into layers and trained with a rudimentary version of backpropagation [172] in the early 1960s) was motivated largely as a neural model (although it was not applied to vision). Decades later, this work inspired the Neocognitron [61], a model which attempted unsupervised visual representation learning by having neurons memorize and cluster parts of digits. This theory, motivated by Hubel and Wiesel's notions of hierarchy [88], explained complex visual computations in terms of a series of simpler operations, and allowing detected parts to be re-used to detect multiple characters (e.g. a $\vdash$ detector can be used to help detect both a "B" and an "F"). Unfortunately, the Neocognitron's learning method was somewhat ad-hoc and difficult to tune. Hence, their view was not shown to be useful until error backpropagation was reformulated as gradient descent [176], which, in vision, led to performance strong enough to be useful in a real vision task: handwritten digit recognition [117].

Learned representations quickly established a place in computer vision, but these methods still struggled to deal with the complexity of full-resolution natural images. Unlike digits, where tens of thousands of digits did a good job covering the space of variations, it seemed that many orders of magnitude more data would be required to cover the space of natural images. Subsequent works aimed to improve statistical efficiently by providing more constraint on the mid-level representation. For instance, EigenFaces and numerous follow-up works represented faces with principal components analysis [207]. For object recognition, Murase et al. [146] attempted to extend the PCA approach to be more invariant to pose and deformation. Many of these early representations were not particularly hierarchical, often relying directly on pixel-level similarity. The field gradually shifted towards methods which brought back more of the hierarchy, although they tended to rely on hand-engineered low-level features. Most notably, algorithms that extended the pictorial structures approach to representation [56] gradually came to dominate object detection and recognition [15, 52, 113, 230], culminating in the Deformable Parts Model [53] which was highly successful for object detection. For the most successful pictorial structures models, the training was done discriminatively, i.e., the entire pipeline, including the mid-level parts, was optimized to minimize detection error on the training set. However, discriminative training was not universal. In particular, unsupervised representation learning remained dominant in image classification [71, 161, 193]. However, the successful unsupervised learning methods all focused on some kind of clustering or factor analysis which tried to summarize all of the variation in the data, rather than attempting to extract semantically meaningful or descriminative features.

During this period, another dramatic shift was happening besides the change in algorithms: the shift to big datasets and standard benchmarks. After digits, early datasets in face detection quickly led to large performance improvements [173]. In object recognition,

COIL [148] was the first to capture a wide variety of objects (though each category contained only one instance), Caltech-101 [51] increased the complexity of objects, and MSRC [224] and LabelMe [179] put those objects into more complicated scenes, requiring localization in the evaluation as well as classification. PASCAL [49], ImageNet [35, 177], SUN [227], and COCO [129] gradually increased both the complexity of the scenes where the objects occurred and the space of categories that the objects might come from. The result was a gradual progression of richer and richer representations, which were partially propelled by algorithmic advancements, and partially by the demands that the particular datasets placed on the algorithms.

2.3 Present State of the Art

In the past few years, importance of the dataset complexity has been extremely apparent, as deep neural networks have consistently outperformed other methods only once the dataset size is as large as ImageNet or COCO [79,111,189,199]. These "Deep ConvNet" approaches, which are in many ways similar to the ConvNets of the 1980s [117], are the current winners in many important challenges, including ImageNet classification [111], and PASCAL and COCO object detection and segmentation [66, 79, 130]. Besides their strong performance, however, another important contribution of deep networks is to show that importance of a learned representation extends beyond the single task that the representation was trained to perform. When performing object detection on a dataset like PASCAL with limited training examples, it helps to begin with a deep representation trained on the larger ImageNet dataset [66, 111]. This same network transfers to other problems like segmentation [130], geometry estimation [219], pose estimation [16], image retrieval [68], and numerous other tasks [167]. These results suggest that the ImageNet-trained representation is "general-purpose," i.e., it provides a strong and useful prior for other visual learning. Therefore, we argue that it is important to understand how a learning algorithm can result in a general-purpose representation, since we ultimately hope to build representations that are even better than the ones trained from ImageNet.

This leads to two questions. First, under what circumstances does a machine learning algorithm learn to do more than the single task that it is trained on? Second, is it possible to begin with a task that does not appear to be useful by itself, and use it to train a useful representation? These are the core questions that this thesis aims to address, and they are not completely new. Here, we review previous attempts to find visual supervision in unexpected places, most of which have been relatively recent. Much of this work has focused on labels that require little or no human effort to obtain, aspiring to provide cheap, huge datasets to the community. However, little of this work has been influential in the broader field of vision, since so far nothing has seemed likely to outperform more conventional forms of strong supervision.

We consider two broad types of methods: first, those which assume some auxhiliary information is provided alongside each image, which we call *weakly-supervised learning*, and second, those which assume no information is provided besides the pixels, which we call *unsupervised*. We acknowledge, however, that there is disagreement in the field regarding the meaning of these terms. Admittedly the distinction is be somewhat artificial, since weak labels are often available without any human annotation effort, and furthermore, pixels in the context of an image region might also be seen as a sort of "weak label" for that region.

2.3.1 Weakly-Supervised Methods

One approach to weak-label learning is to try to get localization for free. That is, the algorithms start with an image collection that has image-level category labels, and infer bounding boxes or segmentations for objects [22, 27, 149, 157, 184, 191, 216]. These works generally assume relatively strong image-level labels which are collected by hand, and indicate the presence or absence of certain object categories. Other works on cosegmentation [20, 99, 107, 175] and colocalization [37, 100, 200] make assumptions about the annotations that are in some sense weaker; e.g., that the images 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 [10, 183], user tags [69, 93, 95, 101, 140, 228, 234], or GPS coordinates [109, 126, 163, 182] 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 [23, 33, 46, 54, 112, 122, 138, 187]. Recent efforts have attempted to summarize visual information across the web into a database of concepts that can make sense of it altogether [19, 38].

Sometimes cameras come with additional hardware that provide auxhiliary information. For instance, in cars scanning street scenes, it is easy to record the 3D offset between images. Deep networks trained to predict this seem to learn semantic information [3, 97]. Robotic sensors also have information about the robot's interaction with the world, allowing researchers to train visual representation that predict the result of physical interactions [162].

Work in this thesis is partially the foundation of yet another line of research on weak supervision that focuses on discriminative patch mining [6, 12, 32, 39, 43, 48, 59, 96, 102, 119, 125, 168, 190, 197, 212, 218, 222, 232], 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 necessarily 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 more details, see Chapters 3 and 4.

2.3.2 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 [42, 58, 84, 92, 141, 215, 220, 225, 236]. 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 exper-

iments [124, 214], and recent work has shown that it can provide improvements on realistic image datasets when training data is limited [220]. 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, adding to their biological plausibility.

Many other cues are present in video. For instance, motion can reveal the boundaries of objects, which is difficult to predict from static images without semantics. Given patches, a neural network can be trained to predict these motion boundaries, which results in object understanding [128]. A neural network can also be trained to recover the temporal ordering of frames [139] or trajectories in future motions [1].

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 [85], contrastive divergence [83], deep Boltzmann machines [181], and Bayesian nonparametric models [195] use sampling to perform approximate inference. Others strengthen the independence assumptions in the model [223]. More recently, variational Bayesian methods for approximate inference have been proposed [108, 170], and adversarial nets have been proposed as a way to evaluate distributions instead of evaluating single samples [36, 70, 165]. These methods have shown promising performance on smaller datasets such as handwritten digits [83, 85, 108, 170, 181], but none have proven effective for understanding the semantics of 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 try to compress the data using a hierarchical compressor and decompressor—called an autoencoder—with the hope that the network will learn object and scene semantics as a method of storing information more compactly and recovering lost signal. For example, denoising autoencoders [9,208] aim to reconstruct images from noisy versions; 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, in conjunction with a sparsity penalty on the representation [154]. To learn deeper (and more non-linear) visual representations, one commonly used approach is to stack such sparse autoencoders [116, 118]. We are aware of only one such stacked sparse autoencoder model that has been applied to full-scale images [116]; 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 reconstructing images well.

One of the ideas explored in this work is "context prediction." A strong tradition for this kind of task already exists in the text domain, where "skip-gram" [137] 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, [134] 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 [115, 155, 202], 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 [40], and another focuses on choosing between predictions rather than forming novel ones [41]. measure how difficult the features are to predict through a "stuff" model, so that the overall algorithm can focus on patches that are not very "stuff"-like and more "thing"-like [40]. Our view of context as supervisory signal has already inspired a small line of research in deep representation learning from other authors [150, 159], suggesting a bright future for related methods.

A final line of work in unsupervised visual learning aims to discover object categories from unlabeled datasets. These approach tend to use hand-crafted features and various forms of clustering (e.g. [178, 192] learned a generative model over bags of visual words). Early bag-of-words approaches tended to 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 [120, 160], but relied on contour extraction, which is difficult in complex images. Many other approaches [50, 72, 104, 106] focus on defining similarity metrics which can be used in more standard clustering algorithms; Rematas et al. [169], for instance, re-casts the problem as frequent itemset mining. Geometry may also be used to for verifying links between images [24, 81, 163], although this fails for deformable objects.

Chapter 3

Discriminative Patch Discovery: What Makes Paris Look like Paris?

3.1 Introduction

Consider the two photographs in Figure 3.1, both downloaded from Google Street View. One comes from Paris, the other one from London. Can you tell which is which? Surprisingly, even for these nondescript street scenes, people who have been to Europe tend to do quite well on this task. In an informal survey, we presented 11 subjects with 100 random Street View images of which 50% were from Paris, and the rest from eleven other cities. We instructed the subjects (who have all been to Paris) to try and ignore any text in the photos, and collected their binary forced-choice responses (Paris / Not Paris). On average, subjects were correct 79% of the time ($std = 6.3$), with chance at 50% (when allowed to scrutinize the text, performance for some subjects went up as high as 90%). What this suggests is that people are remarkably sensitive to the geographically-informative features within the visual environment. But what are those features? In informal debriefings, our subjects suggested that for most images, a few localized, distinctive elements "immediately gave it away". E.g. for Paris, things like windows with railings, the particular style of balconies, the distinctive doorways, the traditional blue/green/white street signs, etc. were particularly helpful. The human ability to find and appreciate the resemblance between instances of a particular architectural feature—even though they are not identical—is especially remarkable given that most people never explicitly train themselves to discriminate architecture. The fact that it comes so naturally for people to notice—i.e. represent—these elements without a clear supervisory signal suggests that it is a good starting point for our representation learning algorithms. Finding those features can be difficult though, since every image can contain more than $25,000$ candidate patches, and only a tiny fraction will be truly distinctive. Before we can tackle the representation problem, we must first solve the association problem, so we will know what should be similar in the space of representations.

In this chapter, we want to find such local geo-informative features *automatically*, directly from a large database of photographs from a particular place, such as a city. Specifically, given tens of thousands of geo-localized images of some geographic region R, we aim to find a few hundred visual elements that are both: 1) repeating, i.e. they occur often in R, and 2) geographically discriminative, i.e. they occur much more often in R than in R^C. Figure 3.1

Figure 3.1: These two photos might seem nondescript, but each contains hints about which city it might belong to. Given a large image database of a given city, 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 top, the emblematic street sign, a balustrade window, and the balcony support are all very indicative of Paris, while on the bottom, the neoclassical columned entryway sporting a balcony, a Victorian window, and, of course, the cast iron railing are very much features of London.

shows sample output of our algorithm: for each photograph we show three of the most geo-informative visual elements that were automatically discovered. For the Paris scene (left), the street sign, the window with railings, and the balcony support are all flagged as informative.

But why is this topic important? 1) Scientifically, the goal of understanding which visual elements are fundamental to our perception of a complex visual concept, such as a place, is an interesting and useful one. Visual elements may be seen as the basic building blocks of an informative representation, so capturing them using only scene-level labels suggests that we can learn rich representations with little human annotation effort. 2) Our work demonstrates how "visual data mining" be used to visualize datasets in ways that humans can understand. Our work shares this motivation with a number of other recent works that propose ways of finding and visualizing existing image data in better ways, be it selecting candid portraits from a video stream [57], summarizing a scene from photo collections [188], finding iconic images of an object [11], etc. 3) More practically, one possible

future application of the ideas presented here might be to help CG modelers by generating so-called "reference art" for a city. For instance, when modeling Paris for Pixar's *Ratatouille*, the co-director Jan Pinkava faced exactly this problem: "The basic question for us was: 'what would Paris look like as a model of Paris?', that is, what are the main things that give the city its unique look?" [156]. Their solution was to "run around Paris for a week like mad tourists, just looking at things, talking about them, and taking lots of pictures" not just of the Eiffel Tower but of the many stylistic Paris details, such as signs, doors etc. [156](see photos on pp.120–121). But if going "on location" is not feasible, our approach could serve as basis for a detail-centric reference art retriever, which would let artists focus their attention on the most statistically significant stylistic elements of the city. 3) And finally, more philosophically, our ultimate goal is to provide a *stylistic narrative* for a visual experience of a place. Such a narrative, once established, can be related to others in a kind of geo-cultural visual reference graph, highlighting similarities and differences between regions. E.g. one could imagine finding a visual appearance "trail" from Greece, through Italy and Spain and into Latin America. In this work, we only take the first steps in this direction – connecting visual appearance across cities, finding similarities within a continent, and differences between neighborhoods. But we hope that our work might act as a catalyst for research in this new area, which might be called *computational geo-cultural modeling*.

3.2 Related Work on Geo-spatial Visual Data Mining

Considerable prior work has focused specifically on understanding and modeling visual scenes in the context of geography, so here we briefly review the literature. In the field of architectural history, descriptions of urban and regional architectural styles and their elements are well established, e.g. [131,198]. Such local elements and rules for combining them have been used in computer systems for procedural modeling of architecture to generate 3D models of entire cities in an astonishing level of detail, e.g. [144], or to parse images of facades, e.g. [201]. However, such systems require significant manual effort from an expert to specify the appropriate elements and rules for each architectural style.

At the other end of the spectrum, data-driven approaches have been leveraging the huge datasets of geotagged images that have recently become available online. For example, Crandall et al. [31] use the GPS locations of 35 thousand consumer photos from Flickr to plot photographer-defined frequency maps of cities and countries, while Kalogerakis et al. [103] use the locations and relative time-stamps of photos of the same photographer to model world-wide human travel priors. Geo-tagged datasets have also been used for place recognition [18, 109, 182] including famous landmarks [126, 127, 233]. Our work is particularly related to [109, 182], where geotags are also used as a *supervisory signal* to find sets of image features discriminative for a particular place. While these approaches can work very well, their image features typically cannot generalize beyond matching specific buildings imaged from different viewpoints. Alternatively, global image representations from scene recognition, such as GIST descriptor [152] have been used for geo-localization of generic scenes on the global Earth scale [78, 103]. There, too, reasonable recognition performance has been achieved, but the use of global descriptors makes it hard for a human to interpret *why* a given image gets assigned to a certain location.

In contrast, here we propose a discovery method that is weakly constrained by location labels derived from GPS tags, and which is able to mine representative visual elements automatically from a large online image dataset, and recognize instances of them even when

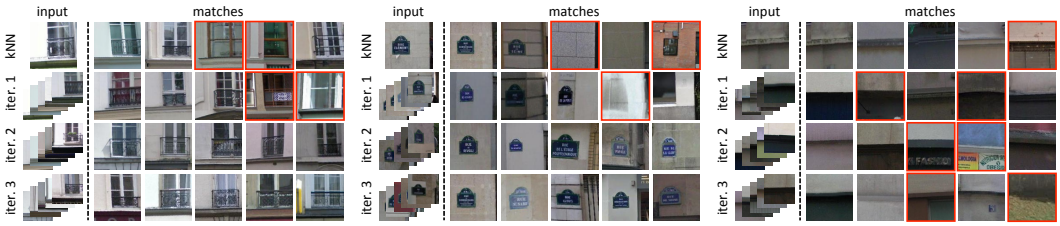


Figure 3.2: Steps of our algorithm for three sample candidate patches in Paris. The first row: initial candidate and its NN matches. Rows 2-4: iterations of SVM learning (trained using patches on left). Red boxes indicate matches outside Paris. Rows show every 7th match for clarity. Notice how the number of not-Paris matches decreases with each iteration, except for rightmost cluster, which is eventually discarded.

they are not geometrically identical. Not only are the resulting visual elements geographically discriminative (i.e. they occur only in a given locale), but they also typically look meaningful to humans, making them suitable for a variety of geo-data visualization applications. The next section describes the data used in this work, followed by the full description of our algorithm.

3.3 The Data

Flickr has emerged as the data-source of choice for most recently developed data-driven applications in computer vision and graphics, including visual geo-location [31, 78, 127]. However, the difficulty with Flickr and other consumer photo-sharing websites for geographical tasks is that there is a strong data bias towards famous landmarks. To correct for this bias and provide a more uniform sampling of the geographical space, we turn to Google Street View – a huge database of street-level imagery, captured as panoramas using specially-designed vehicles. This enables extraction of roughly fronto-parallel views of building facades and, to some extent, avoids dealing with large variations of camera viewpoint.

Given a geographical area on a map, we automatically scrape a dense sampling of panoramas of that area from Google Street View [73]. From each panorama, we extract two perspective images (936x537 pixels), one on each side of the capturing vehicle, so that the image plane is roughly parallel to the vehicle's direction of motion. This results in approximately $10,000$ images per city. For this project, we downloaded 12 cities: Paris, London, Prague, Barcelona, Milan, New York, Boston, Philadelphia, San Francisco, San Paulo, Mexico City, and Tokyo. We have also scraped suburbs of Paris for one experiment.

3.4 Discovering geo-informative elements

Our goal is to discover visual elements which are characteristic of a given geographical locale (e.g. the city of Paris). That is, we seek patterns that are both *frequently occurring* within the given locale, **and** *geographically discriminative*, i.e. they appear in that locale and do not appear elsewhere. Note that neither of these two requirements by itself is enough:

(a) K-Means Clusters using SIFT (Visual Words) (b) K-Means Clusters using HOG

Figure 3.3: (a) k-means clustering using SIFT (visual words) is dominated by low level features. (b) k-means clustering over higher dimensional HOG features produces visually incoherent clusters.

sidewalks and cars occur frequently in Paris but are hardly discriminative, whereas the Eiffel Tower is very discriminative, but too rare to be useful ($< 0.0001\%$ in our data). In this work, we will represent visual elements by square image patches at various resolutions, and mine them from our large image database. Hence, our elements are reminiscent of Poselets [13], but our training procedure avoids the labor-intensive part labeling process that Poselets requires. Instead, we assume only that our dataset can be divided into two parts: (i) the positive set containing images from the location whose visual elements we wish to discover (e.g. Paris); and (ii) the negative set containing images from the rest of the world (in our case, the other 11 cities in the dataset). We assume that many frequently occurring but uninteresting visual patterns (trees, cars, sky, etc.) will occur in both the positive and negative sets, and should be filtered out. Our biggest challenge is that the overwhelming majority of our data is uninteresting, so matching the occurrences of the rare interesting elements is like finding a few needles in a haystack.

One possible way to attack this problem would be to first discover repeated elements and then simply pick the ones which are the most geographically discriminative. A standard technique for finding repeated patterns in data is clustering. For example, in computer vision, "visual word" approaches [193] use k-means clustering on image patches represented by SIFT descriptors. Unfortunately, standard visual words tend to be dominated by low-level features, like edges and corners (Figure 3.3a), not the larger visual structures we are hoping to find. While we can try clustering using larger image patches (with a higher-dimensional feature descriptor, such as HOG [34]), k-means behaves poorly in very high dimensions because the distance metric becomes less meaningful, producing visually inhomogeneous clusters (Figure 3.3b). We also experimented with other clustering approaches, such as Locality-Sensitive Hashing [67], with similar results.

An alternative approach is to use the geographic information as part of the clustering, extracting elements that are both repeated and discriminative at the same time. We have experimented with such discriminative clustering methods [62, 142, 185], but found they did not provide the right behavior for our data: they either produce inhomogeneous clusters or focus too much on the most common visual features. We believe this is because such approaches include at least one step that partitions the *entire* feature space. This tends to lose the needles in our haystack: the rare discriminative elements get mixed with, and overwhelmed by, less interesting patches, making it unlikely that a distinctive element could ever emerge as its own cluster.

In this chapter, we propose an approach that avoids partitioning the entire feature space

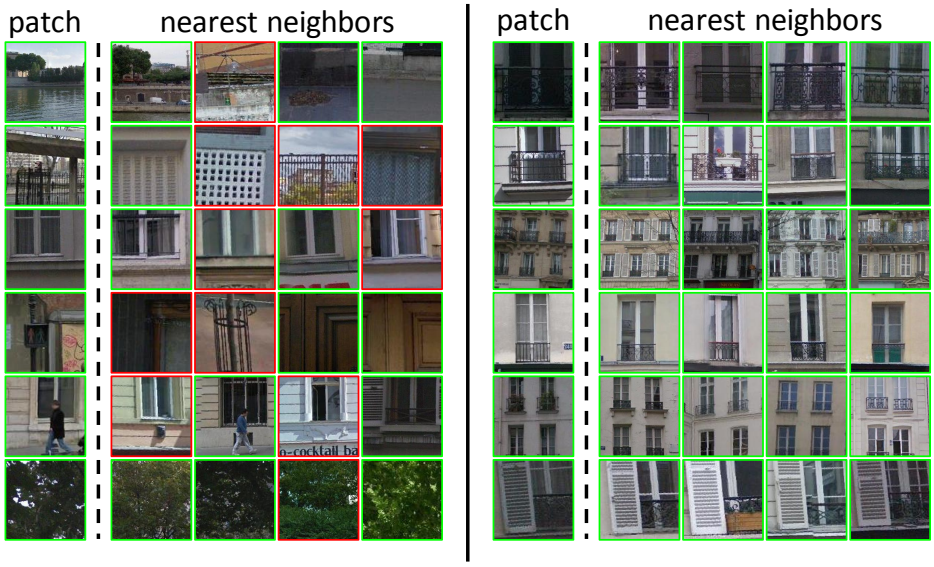


Figure 3.4: Left: randomly-sampled candidate patches and their nearest neighbors according to a standard distance metric. Right: after sorting the candidates by the number of retrieved neighbors that come from Paris, coherent Parisian elements have risen to the top.

into clusters. Instead, we start with a large number of randomly sampled candidate patches, and then give each candidate a chance to see if it can converge to a cluster that is both frequent and discriminative. We first compute the nearest neighbors of each candidate, and reject candidates with too many neighbors in the negative set. Then we gradually build clusters by applying iterative discriminative learning to each surviving candidate. The following section presents the details of this algorithm.

3.4.1 Our Approach

From the tens of millions of patches in our full positive set, we randomly sample a subset of 25,000 high-contrast patches to serve as candidates for seeding the clusters. Throughout the algorithm, we represent such patches using a HOG+color descriptor. First, the initial geo-informativeness of each patch is estimated by finding the top 20 nearest neighbor (NN) patches in the full dataset (both positive and negative), measured by normalized correlation, and counting how many of them come from Paris. Figure 3.4 shows nearest neighbors for a few randomly-selected patches and for the patches whose neighbors all come from Paris. Note that the latter patches are not only more Parisian, but also considerably more coherent. This is because generating a coherent cluster is a prerequisite to retrieving matches exclusively from Paris: any patch whose matches are incoherent will likely draw those matches randomly from inside and outside Paris. We keep the candidate patches that have the highest proportion of their nearest neighbors in the positive set, while also rejecting near-duplicate patches (measured by spatial overlap of more than 30% between any 5 of their top 50 nearest neighbors). This reduces the number of candidates to about 1000.

Figure 3.5: Top: using the naïve distance metric for this patch retrieves some good matches and some poor matches, because the patch contains both a street sign and a vertical bar on the right. Bottom: our algorithm reweights the dimensions of our patch descriptor to separate Paris from non-Paris. The algorithm learns that focusing on the street sign achieves maximum separation from the non-Paris walls.

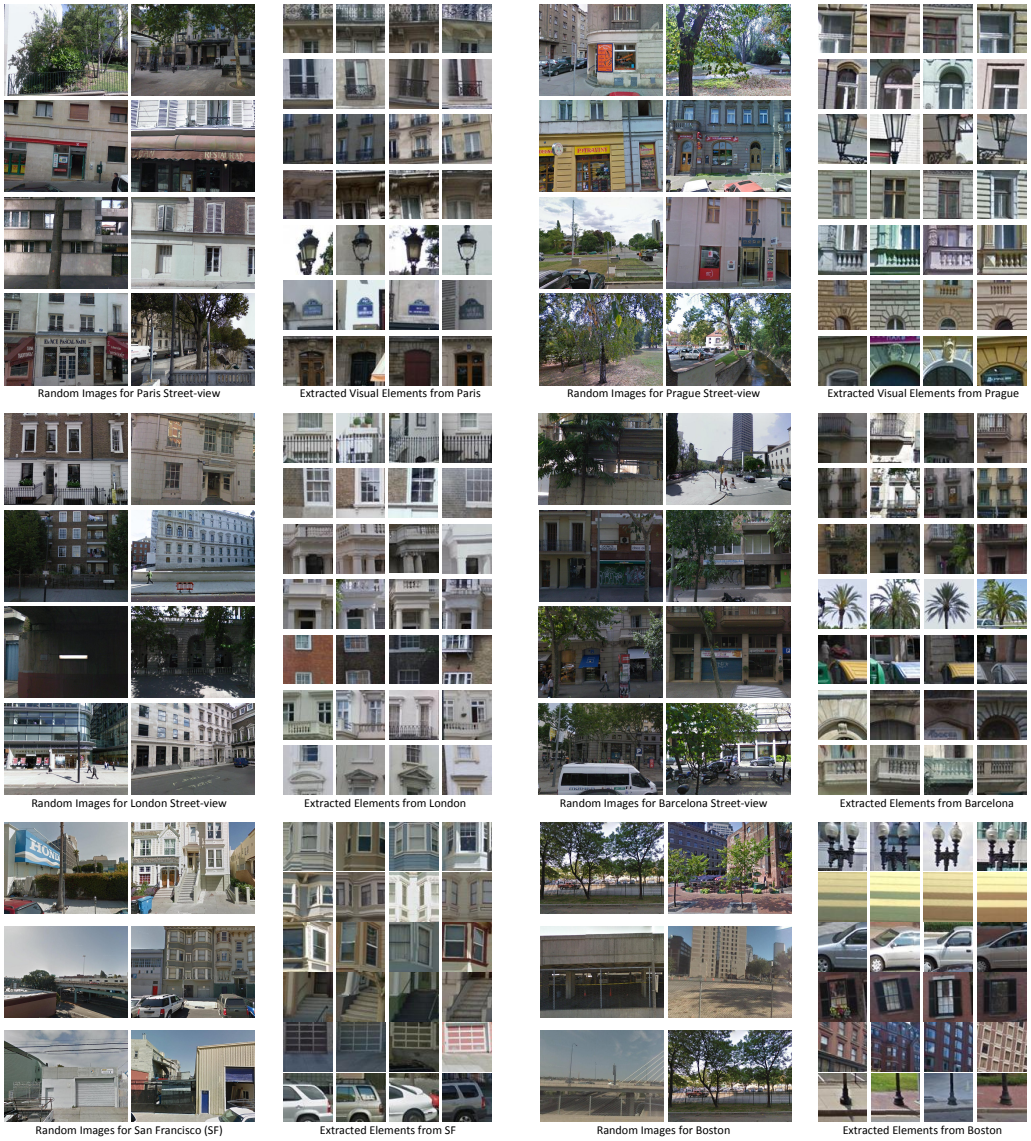


Figure 3.6: Google Street View vs. geo-informative elements for six cities. Arguably, the geo-informative elements (right) are able to provide better stylistic representation of a city than randomly sampled Google Street View images (left).

Some good elements, however, get matched incorrectly during the nearest-neighbors phase. Figure 3.5 shows a patch that contains both a street sign and a vertical bar on the right (the end of the facade). The naïve distance metric doesn't know what's important, and so it tries to match both. Yet too few such patches exist in the dataset; for the remainder, the algorithm matches the vertical bar simply because it's more frequent. To fix this problem, we aim to learn a distance metric that gives higher weight to the features that make the patch geo-discriminative.

In many cases, one can improve visual retrieval by adapting the distance metric to the given query using discriminative learning [186]. We adopt similar machinery, training a linear SVM detector for each visual element in an iterative manner as in [190]. Unlike these previous works, however, we emphasize that the weak labels are the workhorse of the distance learning. In the case of Figure 3.5, for example, we know that the street sign is more important because it occurs only in Paris, whereas the vertical bar occurs everywhere. We train an SVM detector for each visual element, using the top k nearest neigbors from the positive set as positive examples, and all negative-set patches as negative examples. While this produces a small improvement (Figure 3.2, row 2), it is not enough, since the top k matches might not have been very good to begin with. So, we iterate the SVM learning, using the top k detections from previous round as positives (we set $k = 5$ for all experiments). The idea is that with each round, the top detections will become better and better, resulting in a continuously improving detector. However, doing this directly would not produce much improvement because the SVM tends to overfit to the initial positive examples [190], and will prefer them in each next round over new (and better) ones. Therefore, we apply cross-validation by dividing both the positive and the negative parts of the dataset into l equally-sized subsets (we set $l = 3$ for all experiments). At each iteration of the training, we apply the detectors trained on the previous round to a new, *unseen* subset of data to select the top k detections for retraining. In our experiments, we used three iterations, as most good clusters didn't need more to converge (i.e. stop changing). After the final iteration, we rank the resulting detectors based on their accuracy: percentage of top 50 firings that are in the positive dataset (i.e. in Paris). We return the top few hundred detectors as our geo-informative visual elements.

Figure 3.2 illustrates the progression of these iterations. For example, in the left column, the initial nearest neighbors contain only a few windows with railings. However, windows with railings differ more from the negative set than the windows without railings; thus the detector quickly becomes more sensitive to them as the algorithm progresses. The right-most example does not appear to improve, either in visual similarity or in geo-discriminativeness. This is because the original candidate patch was intrinsically not very geo-informative and would not make a good visual element. Such patches have a low final accuracy and are discarded.

Implementation Details: Our current implementation considers only square patches (although it would not be difficult to add other aspect ratios), and takes patches at scales ranging from 80-by-80 pixels all the way to height-of-image size. Patches are represented with standard HOG [34] (8x8x31 cells), plus a 8x8 color image in L*a*b colorspace (a and b only). Thus the resulting feature has 8x8x33 = 2112 dimentions. During iterative learning, we use a soft-margin SVM with C fixed to 0.1. The full mining computation is quite expensive; a single city requires approximately $1,800$ CPU-hours. But since the algorithm is highly parallelizable, it can be done overnight on a cluster.

Figure 3.7: Examples of geographic patterns in Paris (shown as red dots on the maps) for three discovered visual elements (shown below each map). Balconies with cast-iron railings are concentrated on the main boulevards (left). Windows with railings mostly occur on smaller streets (middle). Arch supporting columns are concentrated on Place des Vosges and the St. Germain market (right).

3.4.2 Results and Validation

Figure 3.6 shows the results of running our algorithm on several well-known cities. For each city, the left column shows randomly chosen images from Google Street View, while the right column shows some of the top-ranked visual element clusters that were automatically discovered (due to space limitations, a subset of elements was selected manually to show variety; see the project webpage for the full list). Note that for each city, our visual elements convey a better stylistic feel of the city than do the random images. For example, in Paris, the top-scoring elements zero-in on some of the main features that make Paris look like Paris: doors, balconies, windows with railings, street signs and special Parisian lampposts. It is also interesting to note that, on the whole, the algorithm had more trouble with American cities: it was able to discover only a few geo-informative elements, and some of them turned out to be different brands of cars, road tunnels, etc. This might be explained by the relative lack of stylistic coherence and uniqueness in American cities (with its melting pot of styles and influences), as well as the supreme reign of the automobile on American streets.

In addition to the qualitative results, we would also like to provide a more quantitative evaluation of our algorithm. While validating data-mining approaches is difficult in general, there are a few questions about our method that we can measure: 1) do the discovered visual elements correspond to an expert opinion of what visually characterizes a particular city? 2) are they indeed objectively geo-informative? 3) do users find them *subjectively* geo-informative in a visual discrimination task? and 4) can the elements be potentially useful for some practical task? To answer the first question, we consulted a respected volume on 19th century Paris architecture [131]. We found that a number of stylistic visual elements mentioned in the book correspond quite well to those discovered by our algorithm, as illustrated on Figure 3.8.

To evaluate how geo-informative our visual elements are, we ran the top 100 Paris element detectors over an unseen dataset which was 50% from Paris and 50% from elsewhere. For each element, we found its geo-informativeness by computing the percentage of the time it fired in Paris out of the top 100 firings. The average accuracy of our top detectors was 83% (where chance is 50%). We repeated this for our top 100 Prague detectors, and

Figure 3.8: Books on Paris architecture are expressly written to give the reader a sample of the architectural elements that are specifically Parisian. We consulted one such volume [Loyer, 1988] and found that a number of their illustrative examples (left) were automatically discovered by our method (right).

found the average accuracy on an unseen dataset of Prague to be 92%. Next, we repeated the above experiment with people rather than computers. To avoid subject fatigue, we reduced the dataset to 100 visual elements, 50 from Paris and 50 from Prague. 50% of the elements were the top-ranked ones returned by our algorithm for Paris and Prague. The other 50% were randomly sampled patches of Paris and Prague (but biased to be high-contrast, as before, to avoid empty sky patches, etc). In a web-based study, subjects (who have all been to Paris but not necessarily Prague) were asked to label each of the 100 patches as belonging to either Paris or Prague (forced choice). The results of our study (22 naive subjects) are as follows: average classification performance for the algorithm-selected patches was 78.5% ($std = 11.8$), while for random patches it was 58.1% ($std = 6.1$); the p-value for a paired-samples t-test was $< 10^{-8}$. While on random patches subjects did not do much better than chance, performance on our geo-informative elements was roughly comparable to the much simpler full-image classification task reported in the beginning of the chapter (although since here we only used Prague, the setups are not quite the same).

Finally, to get a sense of whether our elements might serve as "reference art," we asked an artist to sketch a photograph of Paris, allowing only 10 minutes so that some details had to be omitted. Several days later, she made another 10-minute sketch of the same photograph, this time aided by a display of the top 10 geo-informative elements our algorithm detected in the image. In an informal, randomized survey, 10 out of our 11 naive subjects (who had all been to Paris) found the second sketch to be more Paris-like. The two sketches are shown in Figure 3.9.

3.5 Applications

Now that we have a tool for discovering geographically-informative visual elements for a given locale, we can use them to explore ways of building stylistic narratives for cities and of making visual connections between them. Here we discuss just a few such directions.

Figure 3.9: Geo-informative visual elements can provide subtle cues to help artists better capture the visual style of a place. We asked an artist to make a sketch from a photo of Paris (left), and then sketch it again after showing her the top discovered visual elements for this image (right). Note, for example, that the street sign and window railings are missing in the left sketch. In our informal survey, most people found the right sketch to be more Paris-like.

3.5.1 Mapping Patterns of Visual Elements

So far, we have shown the discovered visual elements for a given city as an ordered list of patch clusters (Figure 3.6). Given that we know the GPS coordinates of each patch, however, we could easily display them on a map, and then search for interesting geo-spatial patterns in the occurrences of a given visual element. Figure 3.7 shows the geographical locations for the top-scoring detections for each of 3 different visual elements (a sampling of detections are shown below each map), revealing interestingly non-uniform distributions. For example, it seems that balconies with cast-iron railings (left) occur predominantly on the large thoroughfares (bd Saint-Michel, bd Saint-Germain, rue de Rivoli), whereas windows with cast-iron railings (middle) appear mostly on smaller streets. The arch-supporting column (right) is a distinguishing feature of the famous Place des Vosges, yet it also appears in other parts of Paris, particularly as part of more recent Marché Saint-Germain (this is a possible example of so-called "architectural citation"). Automatically discovering such architectural patterns may be useful to both architects and urban historians.

3.5.2 Exploring Different Geo-spatial Scales

So far we have focused on extracting the visual elements which summarize appearance on one particular scale, that of a city. But what about visual patterns across larger regions, such as a continent, or a more specific region, such as a neighborhood? Here we demonstrate visual discovery at different geo-spatial scales.

We applied our algorithm to recover interesting patterns shared by the cities on the European subcontinent. Specifically, we used Street View images from five European cities (Barcelona, London, Milan, Paris and Prague) as the positive set, and the remaining 7 non-European cities as the negative set. Figure 3.10 shows some interesting discriminative fea-

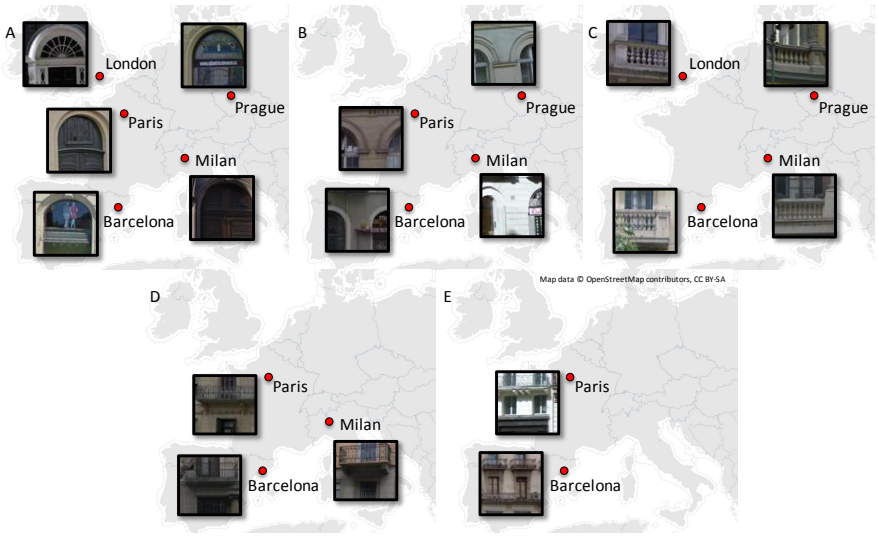


Figure 3.10: Architectural patterns across Europe. While arches (A) are common across all Europe, double arches (B) seem rare in London. Similarly, while Paris, Barcelona and Milan all share cast-iron railings on their balconies (D), the grid-like balcony arrangement (E) of Paris and Barcelona is missing in Milan.

tures and patterns in terms of their membership across the 5 European cities. For example, while arches are common in cities across Europe, double-arches seem rare in London. Similarly, while balcony railings in Paris, Barcelona and Milan are all made of cast iron, they tend to be made of stone in London and Prague.

We also analyzed visual patterns at the scale of a city neighborhood. Specifically, we considered three well-defined districts of Paris: Louvre/Opera (1e, 2e), Le Marais (4e), and Latin Quarter/Luxembourg (5e, 6e). Figure 3.11 shows examples of geographically informative elements for each of the three districts (while taking the other districts and Paris suburbs as the negative set). Predictably, Louvre/Opera is differentiated from the rest of Paris by the presence of big palatial facades. Le Marais is distinguished by its more cozy palaces, very close-up views due to narrow streets, and a specific shape of lampposts. Interestingly, one of the defining features of the Latin Quarter/Luxembourg is the high frequency of windows with closed shutters as compared to other districts in Paris. One possible explanation is that this neighborhood has become very prestigious and a lot of its real-estate has been bought up by people who don't actually live there most of the time.

Given the detectors for visual elements at different geo-spatial scales, it becomes possible to analyze a scene in terms of the regions from which it draws its architectural influences. Figure 3.12 shows images from the 5th arrondissement of Paris, pointing out which elements are specific to that arrondissement, which are Paris-specific, and which are pan-European. For example, the stone balcony railings and arches are pan-European, windows with collapsible shutters and balconies with iron railings are Parisian, and the grooves around the windows are typical of the 5th arrondissement.

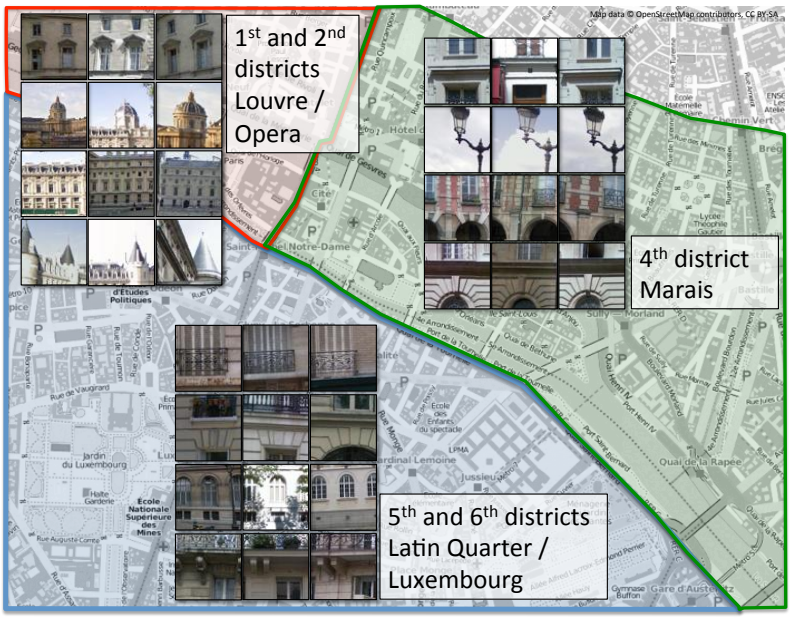


Figure 3.11: Geographically-informative visual elements at the scale of city neighborhoods. Here we show a few discovered elements particular to three of the central districts of Paris: Louvre/Opera, the Marais, and the Latin Quarter/Luxembourg.

3.5.3 Visual Correspondences Across Cities

Given a set of architectural elements (windows, balconies, etc.) discovered for a particular city, it is natural to ask what these same elements might look like in other cities. As it turns out, a minor modification to our algorithm can often accomplish this task. We have observed that a detector for a location-specific architectural element will often fire on functionally similar elements in other cities, just with a much lower score. That is, a Paris balcony detector will return mostly London balconies if it is forced to run only on London images. Naturally these results will be noisy, but we can clean them up using an iterative learning approach similar to the one in Section 3.4.1. The only difference is that we require the positive patches from each iteration of training to be taken not just from the source city, but from all the cities where we wish to find correspondences. For example, to find correspondences between Paris, Prague, and London, we initialize with visual elements discovered in Paris and then, at each round of "clean-up" training, we use 9 top positive matches to train each element SVM, 3 from each of the three cities. Figure 3.13 illustrates the result of this procedure. Note how capturing the correspondence between similar visual elements across cities can often highlight certain stylistic differences, such as the material for the balconies, the style of the street-lamps, or the presence and position of ledges on the facades.

Another interesting observation is that some discovered visual elements, despite having a limited spatial extent, can often encode a much larger architectural context. This becomes particularly apparent when looking at the same visual element detector applied in different cities. Figure 3.14 shows object-centric averages (in the style of [206]) for the detector in

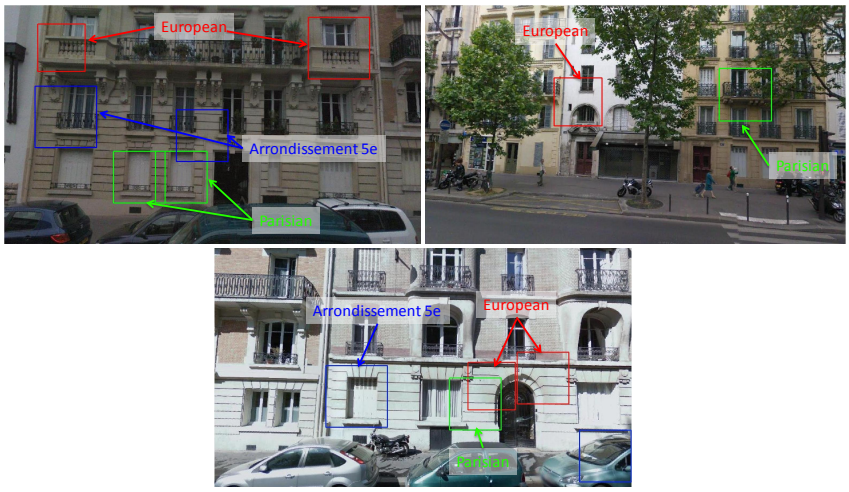


Figure 3.12: Detecting architectural influences. Each image shows confident detections for architectural styles at different geographic scales.

Figure 3.13: Visual Correspondence. Each row shows corresponding detections of a single visual element detector across three different cities.

the top row of Figure 3.13 for Paris and London. That is, for each city, the images with the top 100 detections of the element are first centered on that element and then averaged together in image space. Note that not only do the average detections (red squares) look quite different between the two cities, but the average contexts reveal quite a lot about the differences in the structure and style of facades. In Paris, one can clearly see four equal-height floors, with a balcony row on the third floor. In London, though, floor heights are

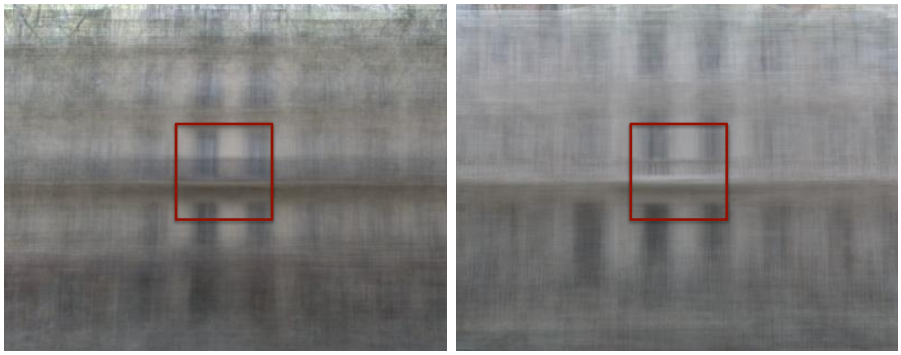


Figure 3.14: Object-centric image averages for the element detector in the top row of Figure 3.13. Note how the context captures the differences in facade styles between Paris (left) and London (right).

Figure 3.15: Geographically-informed retrieval. Given a query Prague image (left), we retrieve images in Paris (right).

uneven, with the first floor much taller and more stately.

3.5.4 Geographically-informed Image Retrieval

Once we have detectors that set up the correspondence between different cities such as Paris and Prague (Sec. 3.5.3), we can use them for geographically-informed image retrieval. Given a query image from one location, such as Prague, our task is to retrieve similar images from another location, such as Paris. For this we use the correspondence detectors from Sec. 3.5.3 while also encoding their spatial positions in the image. In particular, we construct a feature vector of the query image by building a spatial pyramid and max-pooling the SVM scores of the correspondence detectors in each spatial bin in the manner of [121]. Retrieval is then performed using the Euclidean distance between the feature vectors. Figure 3.15

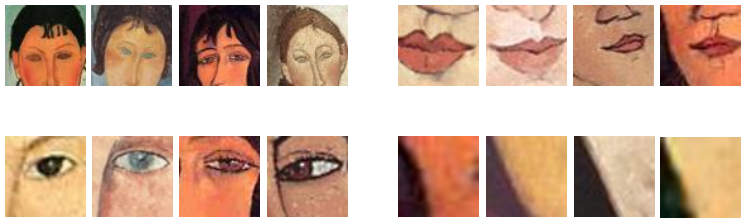


Figure 3.16: What makes Modigliani look like Modigliani?

demonstrates this approach where a query image from Prague retrieves images from Paris that contain similar balconies with cast iron railings (bottom) while honoring spatial layout of facades.

3.6 Conclusion

So, what makes Paris look like Paris? We argued that the "look and feel" of a city rests not so much on the few famous landmarks (e.g. the Eiffel Tower), but largely on a set of stylistic elements, the visual minutiae of daily urban life. We proposed a method that can automatically find a subset of such visual elements from a large dataset offered by Google Street View, and demonstrated some promising applications. This work is but a first step towards our ultimate goal of providing stylistic narratives to explore the diverse visual geographies of our world. Currently, the method is limited to discovering only local elements (image patches), so a logical next step would be trying to capture larger structures, both urban (e.g. facades), as well as natural (e.g. fields, rivers). Finally, the proposed algorithm is not limited to geographic data. For instance, we performed a preliminary experiment mining the visual style of Modigliani; the results in Figure 3.16 demonstrate that the method can isolate some of the most iconic visual themes of that master. Since publication, other authors have applied similar algorithms to understand the style of cars [119]. This suggests that our algorithm may be useful for discovering stylistic elements in a wide variety of other weakly supervised settings.

Chapter 4

Improved Patch Discovery for Scene Recognition and Visualization

4.1 Introduction

Our work in the previous chapter, as well as several approaches based on it [43, 48, 96, 102, 125, 190, 197, 222] 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 [102] or GPS coordinates [43] (but can also run unsupervised [190]), and have been recently used for tasks including image classification [102, 190, 222], object detection [48], visual data mining [43, 125], action recognition [96], and geometry estimation [59]. 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

Figure 4.1: 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 [43]; 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.

we wish to discard much of the data while making as few passes through the dataset as possible.

The well-known mean-shift algorithm [21, 29, 60] 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 [196] for a review), we did not find any that were particularly suitable for finding local maxima of density ratios. Hence, the first contribution of this chapter 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 4.2). When we perform gradient ascent separately for each element as in standard mean-shift, however, we find that the most frequently-occuring elements tend to be over-represented. Hence, section 4.4 describes a modification to our gradient ascent algorithm which uses inter-element communication to approximate common adaptive bandwidth procedures. Finally, in section 4.5 we demonstrate that our algorithms produce visual elements which are more representative and discriminative than previous methods, and in section 4.6 we show they significantly improve performance in scene classification.

4.2 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:

$$\hat{r}(x) \propto \sum_{i=1}^{n} \theta_i K(\|x - x_i\|/h)$$

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 4.1: any kernel bandwidth which is large enough to work well in the left example will be far 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 [30]: 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; [30], 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 [64], they rely on approximate nearest neighbor algorithms, which work for low-dimensional spaces (≤ 48 dimensions in [64]), 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 [21] that classical mean-shift (using a flat kernel) is equivalent to finding the local maxima of the following density estimate:

$$\frac{\sum_{i=1}^{n} \max(b - d(x_i, w), 0)}{z(b)} \tag{4.1}$$

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 [21]. 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.

$$\frac{\sum_{i=1}^{n_{pos}} \max(B(w) - d(x_i^+, w), 0)}{\sum_{i=1}^{n_{neg}} \max(B(w) - d(x_i^-, w), 0)} \tag{4.2}$$

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:

$$\sum_{i=1}^{n_{neg}} \max(b - d(x_i^-, w), 0) = \beta \tag{4.3}$$

Where β 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 (4.2) (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, [166] 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 [17], 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 (4.2) as:

$$\sum_{i=1}^{n_{pos}} \max(w^\top x_i^+ - b, 0) \text{ s.t. } \begin{array}{c} \sum_{i=1}^{n_{neg}} \max(w^\top x_i^- - b, 0) = \beta \\ \|w\|^2 = 1 \end{array} \tag{4.4}$$

Where $B(w)$ has been replaced by b as in equation (4.3), 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:

$$\sum_{i=1}^{n_{pos}} \max(w^\top x_i^+ - b, 0) - \lambda\|w\|^2 \text{ s.t. } \sum_{i=1}^{n_{neg}} \max(w^\top x_i^- - b, 0) = \beta \tag{4.5}$$

Note that (4.5) is equivalent to (4.4) for some appropriate rescaling of λ and β. It can be easily shown that multiplying λ by a constant factor does not change the relative location of local maxima, as long as we divide β 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 λ and β which makes the norm of w equal to 1. [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 [190]. However, unlike [190], 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 [133, 229].

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. 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 ϵ, akin to "hard mining" for SVMs. We perform gradient ascent after each mining phase. An example result for this algorithm is shown in in Figure 4.2, and we include further results below.

[1] 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 derease in β in (4.4). 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.

4.3 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 (4.10).

After a round of mining on a single chunk of the data (including positives and negatives), we set the α's according to the procedure described in section 4.4. We must then optimize the following:

$$\sum_{i=1}^{n_{pos}} \alpha_{i,j} \max(w_j^\top x_i^+, 0) - \lambda \sum_{j=1}^{m} \|[w_j]_{1:d}\|^2 \text{ s.t. } \sum_{i=1}^{n_{neg}} \max(w_j^\top x_i^-, 0) \leq \beta \tag{4.6}$$

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 (4.6) 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 (4.6) 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 (4.6) and then find its projection ∇ onto the current affine subspace, *i.e.*, the space defined by:

$$\nabla^\top \sum_{i=1}^{n_{neg}} x_i^- I(w_j^\top x_i^- > 0) = 0 \tag{4.7}$$

where I is the indicator function. This means that small updates in the direction ∇ 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:

$$t^* = \arg\max_t \sum_{i=1}^{n_{pos}} \alpha_{i,j} (w_j + t\nabla)^\top x_i^+ * I(w_j^\top x_i^+ \geq 0) - \lambda \|[w_j + t\nabla]_{1:d}\|^2 \tag{4.8}$$

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 (4.6). 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:

$$\left| \sum_{i=1}^{n_{neg}} \max(w_j^\top x_i^-, 0) - \beta \right| \tag{4.9}$$

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.

4.4 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 4.2 (middle). One way to deal with this is to assign smaller bandwidths to patches in dense regions of the space [30], *e.g.*, the window railing on row 1 of Figure 4.2 (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 fomulation, we maximize (again with respect to the w's and b's):

$$\sum_{i=1}^{n_{pos}} \sum_{j=1}^{m} \alpha_{i,j} \max(w_j^\top x_i^+ - b_j, 0) - \lambda \sum_{j=1}^{m} \|w_j\|^2 \text{ s.t. } \forall j \quad \sum_{i=1}^{n_{neg}} \max(w_j^\top x_i^- - b_j, 0) = \beta \tag{4.10}$$

Where each $\alpha_{i,j}$ is chosen such that any patch which is a member of multiple clusters gets a lower weight. (4.10) 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 α'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 α'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 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

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 4.3;
Set the last row of W equal to B.;
Distribute I^+ and I^- evenly into l sets, $I_1, ..., I_L$;
for $l \leftarrow 1$ **to** L **do**
  Mine for patches x in I_l for which any of $W^\top x > 0$;
  for $j \leftarrow 1$ **to** m **do**
    $X \leftarrow$ the set of x for which $w_j^\top x > 0$;
    $[w_j] \leftarrow$ `optimize`(w_j, X)
  end
end

Function optimize(w, X)

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

Function satisfyConstraints(w, X^-)

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

$$\alpha_{i,j} = \frac{\max(w_j^\top x_i^+ - b_j, 0)}{\max(w_j^\top x_i^+ - b_j, 0) + \sum_{k=1}^m I(C_k \neq C_j) \max(w_k^\top x_i^+ - b_k, 0)} \tag{4.11}$$

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"

Figure 4.2: Left: without competition, the algorithm from section 4.2 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 4.4, the window gets down-weighted and the algorithm can learn the sidewalk barrier.

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:

$$\alpha'_{i,j,p} = \frac{\max(w_j^\top x_i^+ - b_j, 0)}{\max(w_j^\top x_i^+ - b_j, 0) + \sum_{x \in Ov(p)} \sum_{k=1}^{m} I(C_k \neq C_j) \max(w_k^\top x_i^+ - b_k, 0)} \tag{4.12}$$

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 [30], 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.

4.5 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 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 [43] 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 4.3.

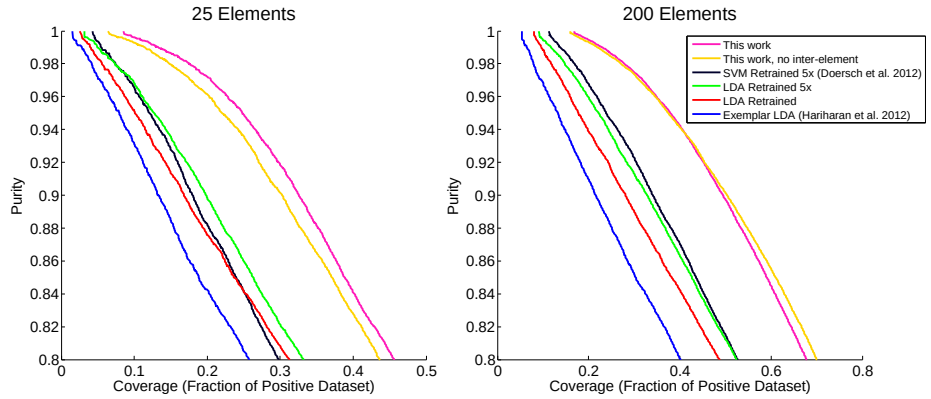


Figure 4.3: Purity-coverage graph for our algorithm and baselines. In each plot, purity measures the accuracy of the element detectors, whereas coverage captures how often they fire. Curves are computed over the top 25 (left) and 200 (right) elements. Higher is better.

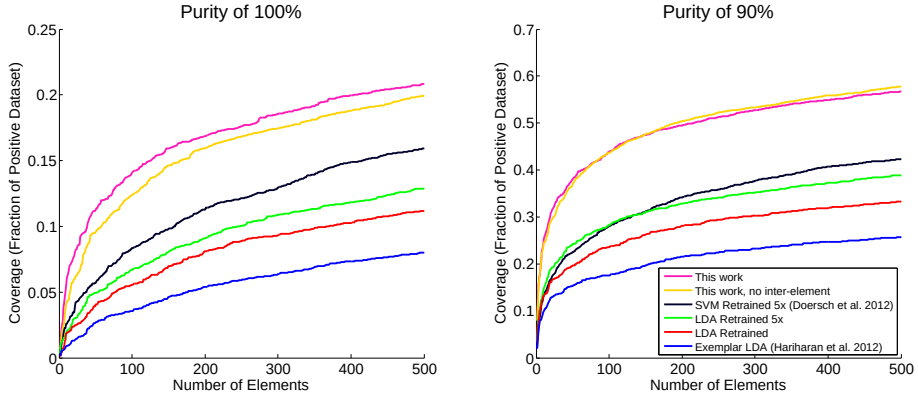


Figure 4.4: Coverage versus the number of elements used in the representation. On the left we keep only the detections with a score higher than the score of the detector's first error (*i.e.* purity 1). On the right, we lower the detection threshold until the elements are 90% pure. Note: this is the same purity and coverage measure for the same elements as Figure 4.3, just plotted differently.

To plot the curve for a given value of purity p, we rank all patches by $w^\top x - b$ 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 4.3, 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 4.4.

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 [76]. 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 [102] (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 [43], except that it uses LDA instead of an SVM. Finally, we include the algorithm of [43], which is a weakly supervised version of [190], 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 [43] and whiten them following [76]. 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.* $log2(\sqrt{width * height}) * 5$ through $log2(\sqrt{width * height}) * 5 + 4$). For experiments without inter-element communication, we simply set $\alpha_{i,j}$ to .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.

4.6 Scene Classification

Finally, we evaluate whether our visual element representation is useful for scene classification. We use the MIT Scene-67 dataset [164], where machine performance remains substantially below human performance. For indoor scenes, objects within the scene are often more useful features than global scene statistics [102]: for instance, shoe shops are similar to other stores in global layout, but they mostly contain shoes.

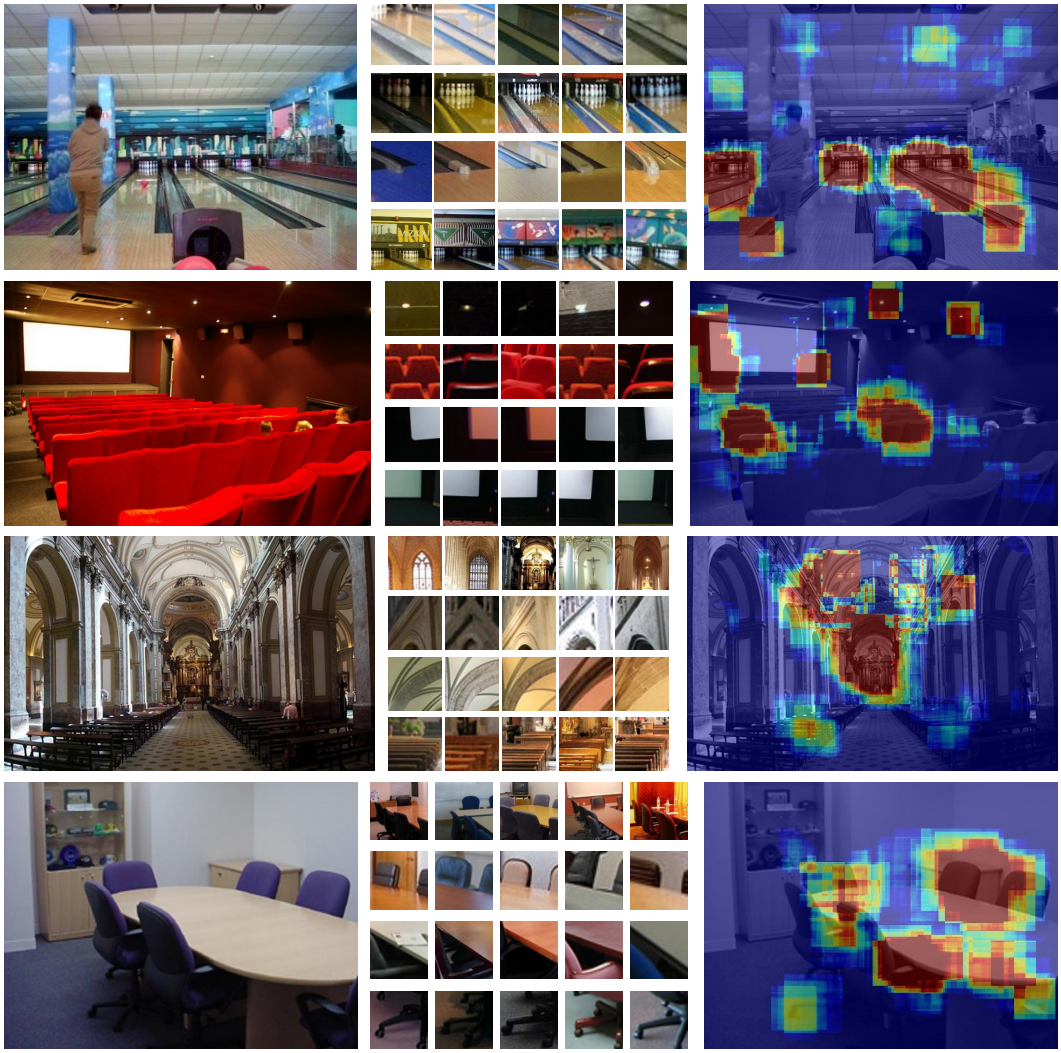


Figure 4.5: For each correctly classified image (left), we show four elements (center) and heatmap of the locations (right) that contributed most to the classification.

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 $107,200$ 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

Table 4.1: Results on MIT 67 scenes

ROI + Gist [164]	26.05	D-Patches [190]	38.10	D-Parts [197]	51.40
MM-scene [235]	28.00	LPR [180]	44.84	IFV [102]	60.77
DPM [157]	30.40	BoP [102]	46.10	BoP+IFV [102]	63.10
CENTRIST [226]	36.90	miSVM [125]	46.40	Ours (no inter-element, §4.2)	**63.36**
Object Bank [123]	37.60	D-Patches (full) [190]	49.40	Ours (§4.4)	**64.03**
RBoW [158]	37.93	MMDL [222]	50.15	Ours+IFV	**66.87**

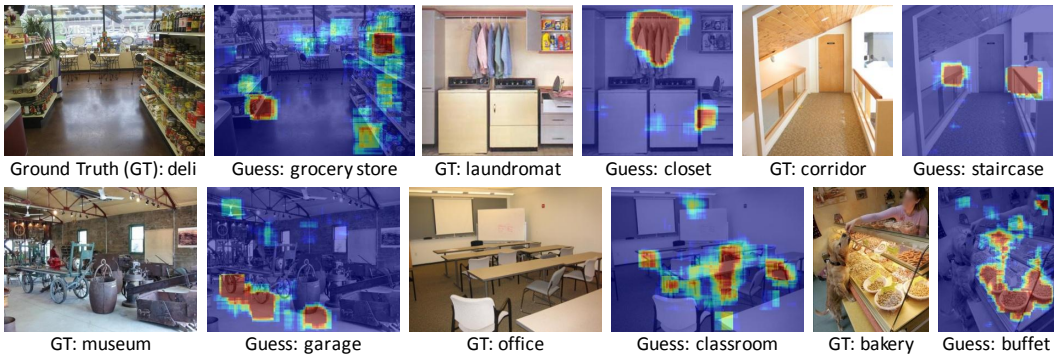


Figure 4.6: Each of these images was misclassified by the algorithm, and the heatmaps explain why. For instance, it may not be obvious why a corridor would be classified as a staircase, but we can see (top right) that the algorithm has identified the railings as a key staircase element, and has found no other staircase elements the image.

$n_{overlap}$ 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 $-.5$ (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 [190], which selects only the elements for a given class in each class-specific SVM.

Figure 4.5 shows a few qualitative results of our algorithm. Quantitative results and comparisons are shown in Table 4.1. We significantly outperform other methods based on discriminative patches, suggesting that our training method is useful. We even outperform the Improved Fisher Vector of [102], 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 performance. For instance, if we trivially zero-out low-valued features until fewer than 6% are nonzero, we still achieve 60.45% accuracy.

4.7 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 demonstrat-

ing 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. Finally, there are many more domains where discriminative patch discovery has proven effective where our mode-seeking formulation should improve results. For instance, we have already shown strong results on object detection using discriminative mode seeking [7], and we expect similar results in domains like geometry estimation and video analysis.

Chapter 5

Object Discovery by Learning to Predict Context

5.1 Introduction

Proponents of unsupervised representation learning [55,85,116,154] and unsupervised object discovery [50, 72, 106, 120, 160, 175, 178, 192] 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 [205]. Unsupervised object discovery has, however, proven extremely difficult; one state-of-the-art result [116] 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 [39]. 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 [39, 43, 48, 96, 102, 125, 190, 197, 222]. 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 chapter is the use of *context* [153] 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 5.1b, 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 5.1c 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

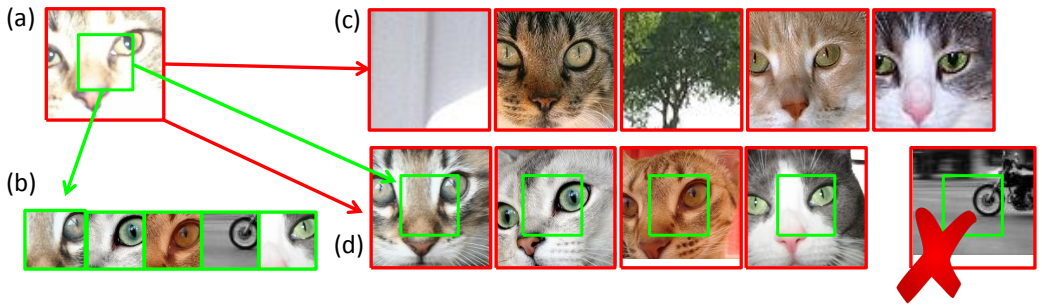


Figure 5.1: Suppose we want to create a visually meaningful cluster containing the cat eye patch in (a). (b) shows a cluster produced by simple nearest neighbors, but there is too little information in just a cat eye, so it is confused with a motorbike. Nearest neighbors on a larger patch (c) introduces 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 [223]. 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' [2], 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 [47]. The model's alignment errors are thus corrected before they drift too far.

5.2 Overview

At a high level, our pipeline is similar to algorithms for mid-level patch discovery [43, 190], especially in the early stages. Like [43], 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. ([43] uses normalized correlation as a distance metric, but we found Exemplar LDA [76], with a Gaussian learned

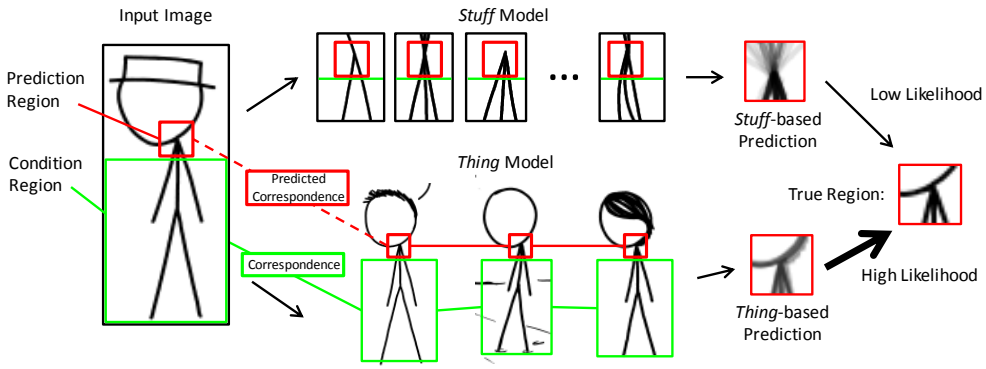


Figure 5.2: Algorithm overview. Given a "condition" region (in green), our algorithm predicts the "prediction" region (in red) twice: once using a model that assumes correspondence with some complex shape (the *thing* model, bottom), and once assuming that the region is best modeled as texture (the *stuff* 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 [145])

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 5.2 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 [51, 55, 85, 195, 217, 223], but historically these algorithms have performed poorly for object recognition problems, especially compared to the discriminative methods that have, of late, largely displaced them in the field. A core difficulty shared by generative methods is that they assume the image features are independent, conditioned on some set of latent variables. Obtaining a likelihood $P(c|p)$ requires integrating out those latent variables, which is generally intractable. Approximations (e.g. MCMC or variational methods) either do not scale well, or produce probability estimates that cannot be compared between different models. To get around this problem, our algorithm partitions the context c into small regions c_k (for example, if c is the HOG representation of the context, each c_k may be a single cell.) Next, we factorize the conditional likelihood as follows:

$$P(c|p) = \prod_{k=1}^{m} P(c_k|c_1, ..., c_{k-1}, p) \tag{5.1}$$

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 [45, 202, 203], and handwritten digits [115,147], 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.

5.3 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 5.2, $\mathcal{P}$ would be a strict subset of the region outlined in green for all but the first term in the product in Equation 5.1). 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 5.2, $\mathcal{C}$ 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. 5.1) for the *thing* model can now be written more formally as:

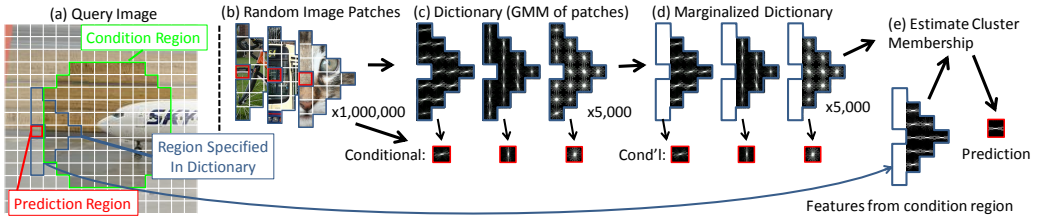


Figure 5.3: Summary of our *stuff* model. (a) Given a condition region (green), we extract cells (blue) that are near to the prediction region (red). We assume we have a dictionary reminiscent of visual words (in our case, a GMM) learned from many sampled patches (b-c). For each dictionary element, we 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.

$$P_T(H^0_{\mathcal{C}}|H^0_{\mathcal{P}}) = \prod_{t=1}^{|\mathcal{C}|} P_T(H^0_{\mathcal{C}[t]}|H^0_{\mathcal{C}[1:t-1]}, H^0_{\mathcal{P}}) \tag{5.2}$$

We will have a similar factorization for P_S of the *stuff* model. In Figure 5.2, the region outlined in red corresponds to $H^0_{\mathcal{C}[t]}$, and those outlined in green correspond to $\{H^0_{\mathcal{C}[1:t-1]}, H^0_{\mathcal{P}}\}$. We repeat this computation of $P_.(H^0_{\mathcal{C}[t]}|H^0_{\mathcal{C}[1:t-1]}, H^0_{\mathcal{P}})$ for all t; i.e. at the next iteration, the red region will get added to the green region and we'll choose a new red region.

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

5.3.1 *Stuff* Model

To construct the *stuff* prediction $P_S(H^0_{\mathcal{C}[t]}|H^0_{\mathcal{C}[1:t-1]}, H^0_{\mathcal{P}})$ (which we will abbreviate as $p^S_{\mathcal{C}[t]}$), the simplest approach is to 1) extract a subset of the condition region that is spatially close to $\mathcal{C}[t]$, 2) find nearest neighbors for that subset from a large database of images, and 3) use the context around those retrieved 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 5.3. We begin with a query image (Figure 5.3a), with a condition region (in green) and a prediction region (in red). We assume that we have available a 'dictionary' (Figure 5.3c) constructed from a large sample of image patches (Figure 5.3b), 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 5.3a). We learn 12 separate dictionaries to ensure that we always have a reasonably good match to a given local condition region. To construct these dictionaries, we first sample about a million such patches (Figure 5.3b), and learn a Gaussian Mixture Model (GMM)

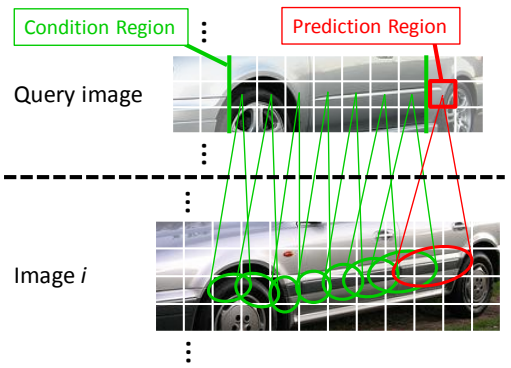


Figure 5.4: Our *thing* model predicts the prediction region (red rectangle) given the condition region (green border) in the query image. We estimate correspondence for the prediction region (red ellipse)—i.e. regions in other images likely to have similar contents—as the basis for this prediction. 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 5.3b) 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 5.3c. 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 5.3c.

To actually make a prediction, we first determine which components of the GMM should be responsible for the prediction. We soft-assign the condition region of our query image (specifically, the subset outlined in blue) to the components of our GMM. Unfortunately, there may be dimensions of our GMM components that correspond to HOG cells outside the condition region; for instance, the leftmost cells highlighted in blue in Figure 5.3a). To deal with this, we marginalize out any such cells from the GMM as shown in Figure 5.3d (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).

5.3.2 *Thing* Model

The *thing* model attempts to capture the details of a complex shape using the set of images that were retrieved when we built our initial patch cluster. Making a prediction for a particular prediction region $P_T(H^0_{\mathcal{C}[t]}|H^0_{\mathcal{C}[1:t-1]}, H^0_{\mathcal{P}})$ (which we will abbreviate as $p^T_{\mathcal{C}[t]}$) boils down to the problem of correspondence: if we can estimate which regions in the other images are likely to correspond to our current prediction region, then we can predict that the features will be the same. To avoid biasing the likelihood value, we must not to access the features $H^0_{\mathcal{C}[t]}$ while making the prediction, but there are cells in the condition region near the prediction region that we could use. Hence, we find the correspondence for each cell in

the condition region (a standard image warping problem). Once we have this correspondence, we extrapolate it to the prediction region. Note, though, that we cannot assume this correspondence will exist for every image. Besides the standard problem of occlusion, we also have to deal with the many errors in the Exemplar-LDA matching. We have found that the top 20 matches are usually reasonably pure, but for some interesting objects the lower-ranked matches may be wrong. Hence, we only use the top 20 images per prediction, meaning we must use the data extremely efficiently.

Predicting from correspondence

Formally, recall that $\mathcal{C}$ and $\mathcal{P}$ index the HOG cells in our query image, and so each index in these sets can be written as 2-dimensional points (x, y) on the grid of HOG cells. We represent the correspondence as a mapping $f^i(x, y)$ from the cell (x, y) in the query image to cells in the HOG grid for image i, where i ranges from 1 to $n - 1$ (we'll call these the 'predictor' images). We optimize our mapping such that $H^i_{f^i(x,y)}$ is as similar as possible to $H^0_{(x,y)}$ for all (x, y) *in the condition region*. Note that we are interested in correspondence for $\mathcal{C}[t] = (x_t, y_t)$, the *prediction* region, but we aren't allowed to access the HOG feature at $H^0_{\mathcal{C}[t]}$; therefore (x_t, y_t) isn't, strictly speaking, in the domain of f^i. To find correspondence for the prediction region, we find the nearest point (x^*_t, y^*_t) in the condition region and compute a simple linear extrapolation:

$$f^i(x_t, y_t) = f^i(x^*, y^*) + (x_t, y_t) - (x^*_t, y^*_t) \tag{5.3}$$

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 5.4. The condition region contains the front wheel of a car and some of the car's side panel, and we are interested in a prediction region further to the right. Ideally, the algorithm should give some probability mass to the event that the panel will continue, and some mass to the event that it will end and a wheel will start. However, if $f^i(x_t, y_t)$ returns a single point as the correspondence for the prediction region, then the algorithm will be arbitrarily confident that *either* the prediction region should contain a continuation of the panel *or* that it will end. To address this, we alter our definition of f such that its range is the space of tuples of mean vectors and covariance matrices parameterizing 2-d Gaussian distributions.

$$f^i(x, y) := \left[\mu^i_{x,y}, \Sigma^i_{x,y}\right] \tag{5.4}$$

Thus, $f^i(x, y)$ defines Gaussian distribution over the HOG grid of image i. (In Equation 5.3, the addition is only performed on μ: i.e. $f^i(x, y) + (a, b) = [\mu^i_{x,y} + (a, b), \Sigma^i_{x,y}]$). Figure 5.4 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 5.4 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 μ and Σ in equation 5.4. 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 5.4), which we parameterize by $\mu^i_{x,y}$ and $\Sigma^i_{x,y}$. We minimize with respect to Σ and μ:

$$E(\Sigma, \mu) = \sum_{i,x,y} c_{x,y} \Phi(H^0_{x,y}, \mu^i_{x,y}, \Sigma^i_{x,y}) + \lambda \sum_{i,x,y} \sum_{(x',y') \in N(x,y)} \Psi(\mu^i_{x,y}, \Sigma^i_{x,y}, \mu^i_{x',y'}, \Sigma^i_{x',y'}) \quad (5.5)$$

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

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

$$L^T_{x,y} = \prod_{(u,v) \in \{\mathcal{C}[1:t-1], \mathcal{P}\}} \left(\hat{p}^T_{u,v}\right)^{\rho([u,v]-[x,y])} \quad ; \quad L^S_{x,y} = \prod_{(u,v) \in \{\mathcal{C}[1:t-1], \mathcal{P}\}} \left(p^S_{u,v}\right)^{\rho([u,v]-[x,y])} \quad (5.6)$$

We then compute:

$$c_{x,y} = L^T_{x,y}/(L^T_{x,y} + L^S_{x,y}) \quad (5.7)$$

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

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

$$\begin{aligned} &\Phi(H^0_{x,y}, \mu^i_{x,y}, \Sigma^i_{x,y}) = \\ &\quad - \textstyle\sum_{u,v} \log \left[\mathcal{N}((u, v); \mu^i_{x,y}, \Sigma^i_{x,y}) * \mathcal{N}(H^i_{u,v}; H^0_{x,y}, \Sigma_H) + \gamma \mathcal{N}(H^i_{u,v}; \mu_H, \Sigma_H)\right] \end{aligned} \quad (5.8)$$

Here, $\mathcal{N}(\cdot; \mu, \mathbf{\Sigma})$ represents a multivariate normal PDF with mean μ and variance $\mathbf{\Sigma}$, γ is a regularization constant (set to 100 in our experiments), and μ_H, Σ_H are the empirical mean and covariance of 31-dimensional HOG cell feature vectors across the full dataset. Intuitively this is a mixture model to explain all HOG cells in image i, where as much data as

possible near $\mu^i_{x,y}$ is explained by a Gaussian in HOG space with mean $H^0_{x,y}$. The term involving μ_H provides a background distribution over HOG cells that can explain any HOG cell somewhat well, and prevents cells in image i that have no good matches in the query image from dominating the optimization.

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

$$\Psi(\mu^i_{x,y}, \Sigma^i_{x,y}, \mu^i_{x',y'}, \Sigma^i_{x',y'}) = \mathcal{D}\left(\mathcal{N}(\cdot, \mu^i_{x,y}, \Sigma^i_{x,y}), \mathcal{N}(\cdot, \mu^i_{x',y'} - [(x',y') - (x,y)], \Sigma^i_{x',y'})\right) \tag{5.9}$$

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

Improving affine invariance: pairwise potentials $\Omega_{x,y}$ The spatial priors defined in Equation 5.9 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 λ (but don't eliminate it entirely, since we don't want to allow arbitrarily large affine transformations), and define a new term which directly penalizes the departure from an affine transformation at each location in f. We define a local affine transformation $\alpha^i_{x,y}$ at each HOG cell in the query image ($\alpha^i_{x,y}$ is represented with a 2-by-2 affine transformation matrix). Then we minimize:

$$E'(\Sigma, \mu, \alpha) = E(\Sigma, \mu) + \lambda' \sum_{x,y,i} \Omega_{x,y,i}(\Sigma^i, \mu^i, \alpha^i_{x,y}) \tag{5.10}$$

In this equation:

$$\Omega_{x,y,i}(\Sigma, \mu, \alpha_{x,y}) = \sum_{[(x',y'),(x'',y'')] \in L^H(x,y)} \mathcal{D}(\mathcal{N}(\cdot; \mu^i_{x',y'}, \Sigma^i_{x',y'}), \mathcal{N}(\cdot; \mu^i_{x'',y''} - [(x'',y'') - (x',y')]\, \alpha^i_{x,y}, \Sigma^i_{x'',y''}) \tag{5.11}$$

Here, $L(x,y)$ represents the edges of a 5-by-5 lattice centered at (x,y) (where each lattice point represents a HOG cell in the query image). $L(x,y)$ defines the region in the query image over which the local affine transformation $\alpha_{x,y}$ applies. In defining $L(x,y)$, we make edges directed and include edges that are symmetries of each other; i.e. $L(3,3)$ contains both $[(1,1),(1,2)]$ and $[(1,2),(1,1)]$; $|L(x,y)|$ will thus contain 80 different edges. Note that each term in the sum over grid edges is essentially the same penalty as Ψ, except that we specify that the displacement between the Gaussians should be defined by the affine transformation. While this may seem like a large number of terms in the sum, in practice we find that the time for optimization is still dominated by Φ, so we get this approximate affine invariance essentially for free.

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 5.8. 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:

$$\zeta^i_{x,y,u,v} = \frac{\mathcal{N}((u,v); \mu^i_{x,y}, \Sigma^i_{x,y}) * \mathcal{N}(H^i_{u,v}; H^0_{u,v}, \Sigma_H)}{\mathcal{N}((u,v); \mu^i_{x,y}, \Sigma^i_{x,y}) * \mathcal{N}(H^i_{u,v}; H^0_{u,v}, \Sigma_H) + \gamma \mathcal{N}(H^i_{u,v}; \mu_H, \Sigma_H)} \tag{5.12}$$

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 $\alpha^i_{x,y}$. For each update of $\mu^i_{x,y}$, we also make an update to $\Sigma^i_{x,y}$ according to ordinary gradient desecent.

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

$$\frac{1}{2}\left(\operatorname{tr}\left(\Sigma_1^{-1}\Sigma_0\right) + (\mu_1 - \mu_0)\,\Sigma_1^{-1}(\mu_1 - \mu_0)^\top - k - \ln\left(\frac{\det \Sigma_0}{\det \Sigma_1}\right)\right). \tag{5.13}$$

Note that all of the edges summed over in Equation 5.11 are either vertical or horizontal, so we may optimize the rows of α separately, in each case focusing either on vertical or horizontal edges. The only term in the KL divergence that matters is the one involving μ. Hence, we can write the objective over the first row of α as:

$$\frac{\lambda'}{2} \sum_{[(x',y'),(x'',y''))] \in L^H(x,y)} \left(\mu^i_{x',y'} - \mu^i_{x'',y''} - \alpha^1_{x,y}\right) \left((\Sigma^i_{x',y'})^{-1} + (\Sigma^i_{x'',y''})^{-1}\right) \left(\mu^i_{x',y'} - \mu^i_{x'',y''} - \alpha^1_{x,y}\right)^\top \tag{5.14}$$

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):

$$c_{x,y} \sum_{u,v} \zeta^i_{x,y,u,v} \left(\mu^i_{x,y} - (u,v)\right) \left(\Sigma^i_{x,y}\right)^{-1} \left(\mu^i_{x,y} - (u,v)\right)^\top \tag{5.15}$$

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

$$\frac{1}{2} \sum_{x',y' \in N(x,y)} \left[\lambda \Xi(I) + \sum_{x'',y'' \in L^{-1}([(x,y),(x',y')])} \lambda' \Xi(\alpha_{x'',y''})\right] \tag{5.16}$$

Where

$$\Xi(\alpha) = \left((\mu^i_{x,y} - \mu^i_{x',y'}) - \alpha\left[(x,y) - (x',y')\right]\right) * \left((\Sigma^i_{x',y'})^{-1} + \left(\Sigma^i_{x'',y''}\right)^{-1}\right) * \left((\mu^i_{x,y} - \mu^i_{x',y'}) - \alpha\left[(x,y) - (x',y')\right]\right)^\top \tag{5.17}$$

$L^{-1}([(x,y),(x',y')])$ 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 Σ keeping all other variables constant does not result in an expression that we are aware can be solved in closed form. Hence, we compute its gradient here. The gradient for the unary term wrt. $\Sigma^i_{x,y}$ is:

$$c_{x,y}\sum_{u,v}\zeta^i_{x,y,u,v} - \Sigma^{-1}_{x,y} + \left(\Sigma^i_{x,y}\right)^{-1}(u,v)^\top(u,v)\left(\Sigma^i_{x,y}\right)^{-1} \tag{5.18}$$

We add to this the gradient with respect to the higher order terms. We'll start with the terms in the KL divergence (Equation 5.13) that don't involve μ:

$$b\sum_{(x',y')\in N(x,y)}\frac{1}{2}\left(\mathrm{tr}\left((\Sigma^i_{x,y})^{-1}\Sigma^i_{x',y'} + (\Sigma^i_{x',y'})^{-1}\Sigma^i_{x,y}\right) - 2k\right). \tag{5.19}$$

Here, b counts the number of terms where such a KL divergence occurs that includes $\Sigma^i_{x,y'}$ i.e. $b = \lambda + \lambda'|L^{-1}([(x,y),(x',y')])|$. Note that the final term from Equation 5.13 cancels when symmetrizing the KL-divergence. The gradient of this is:

$$\frac{b}{2}\sum_{(x',y')\in N(x,y)}(\Sigma^i_{x,y})^{-1}\Sigma^i_{x',y'}(\Sigma^i_{x,y})^{-1} + (\Sigma^i_{x',y'})^{-1} \tag{5.20}$$

Finally, the term involving μ in Equation 5.13 is essentially the same as Equation 5.16, 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:

$$\frac{1}{2}\sum_{x',y'\in N(x,y)}\left[\lambda\Xi'(I) + \sum_{x'',y''\in L^{-1}([(x,y),(x',y')])}\lambda'\Xi'(\alpha_{x'',y''})\right] \tag{5.21}$$

Where:

$$\Xi'(\alpha) = \left(\Sigma^i_{x,y}\right)^{-1} * \left((\mu^i_{x,y} - \mu^i_{x',y'}) - \alpha\left[(x,y) - (x',y')\right]\right)^\top * \left((\mu^i_{x,y} - \mu^i_{x',y'}) - \alpha\left[(x,y) - (x',y')\right]\right) * \left(\Sigma^i_{x,y}\right)^{-1} \tag{5.22}$$

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 $H^1 ... H^{n-1}$ that the prediction region potentially corresponds to:

$$p^T_{\mathcal{C}[t]} = \mathcal{N}(H^0_{\mathcal{C}[t]}; \mu^f_{\mathcal{C}[t]}, \Sigma^f_{\mathcal{C}[t]}) \tag{5.23}$$

where

$$\mu^f_{\mathcal{C}[t]} = \sum_{i,u,v} \eta^i_{\mathcal{C}[t],u,v} H^i_{u,v} \; ; \quad \Sigma^f_{\mathcal{C}[t]} = \sum_{i,u,v} \eta^i_{\mathcal{C}[t],u,v} (H^i_{u,v} - \mu^f_t)(H^i_{u,v} - \mu^f_t)^\top \tag{5.24}$$

Here, $\sum_{i,u,v} \eta^i_{\mathcal{C}[t],u,v} = 1$. There are two components of this weighting: $\eta^i_{\mathcal{C}[t],u,v} = w^i_{\mathcal{C}[t],u,v} \omega^i_{\mathcal{C}[t]}$. The first is based on the spatial correspondence f, and is defined as $w^i_{\mathcal{C}[t],u,v} = \mathcal{N}([u,v]; \mu^i_{x_t,y_t}, \Sigma^i_{x_t,y_t})$ for prediction region (x_t, y_t) (normalized to sum to 1 across u and v). This weight, however, is not sufficient by itself, because the correspondence from the prediction region to image i might be completely wrong (e.g. if there is nothing in image i that corresponds to the prediction region). Hence, we use $\omega^i_{\mathcal{C}[t]}$ to downweight the images where we expect the correspondence to be incorrect. Intuitively, we simply observe how useful image i was for the earlier predictions of other regions near to $\mathcal{C}[t]$.

Computing $\omega^i_{\mathcal{C}[t]}$ Intuitively, our use of $\omega^i_{\mathcal{C}[t]}$ in Equation 5.24 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-expert predictions in a manner similar to how we obtain predictions for the full model: i.e., we estimate a likelihood $P^i_T(H^0_{\mathcal{C}[\tau]} | H^0_{\mathcal{C}[1:\tau-1]}, H^0_{\mathcal{P}})$ (which we will abbreviate as $g^i_{\mathcal{C}[\tau]}$) for all $\tau < t$, using only data from image i. This single image, however, will contain too little data to do a good job estimating the covariance of each conditional gaussian. Thus, we set:

$$g^i_{\mathcal{C}[\tau]} = P^i_T(H^0_{\mathcal{C}[\tau]} | H^0_{\mathcal{C}[1:\tau-1]}, H^0_{\mathcal{P}}) = \mathcal{N}(H^0_{\mathcal{C}[\tau]}; \theta^i_{\mathcal{C}[\tau]}, \Sigma^f_{\mathcal{C}[\tau]}) \tag{5.25}$$

Where

$$\theta^i_{\mathcal{C}[\tau]} = \sum_{u,v} w^i_{\mathcal{C}[\tau],u,v} H^i_{u,v} \tag{5.26}$$

Note the similarities between the prediction made by one expert and the prediction made by the full *thing* model. $\Sigma^f_{\mathcal{C}[\tau]}$ is actually identical to Equation 5.23, i.e. it is estimated using data from all images. $\theta^i_{\mathcal{C}[\tau]}$ is identical to $\mu^f_{\mathcal{C}[\tau]}$ in Equation 5.23 except that it integrates data from only a single image i. Given these per-image predictions $g^i_{\mathcal{C}[\tau]}$, we assume that any expert which predicted well for cells near to $\mathcal{C}[t]$ will also do a good job predicting the cell $\mathcal{C}[t]$. We use Bayes rule to estimate the probability of each expert given the data: i.e. set $\omega^i_{\mathcal{C}[t]}$ equal to the posterior probability of image i versus the other images. Under a uniform prior, the posterior is simply the re-normalized likelihood of the data under the different models. To compute the likelihoods for each model at location $\mathcal{C}[t] = (x_t, y_t)$, we compute the following product of data likelihoods:

$$\omega^i_{\mathcal{C}[t]} \propto \prod_{(x,y) \in \{\mathcal{C}[1:t-1], \mathcal{P}\}} \left(g^i_{(x,y)} \right)^{\rho([x,y] - [x_t, y_t])} \tag{5.27}$$

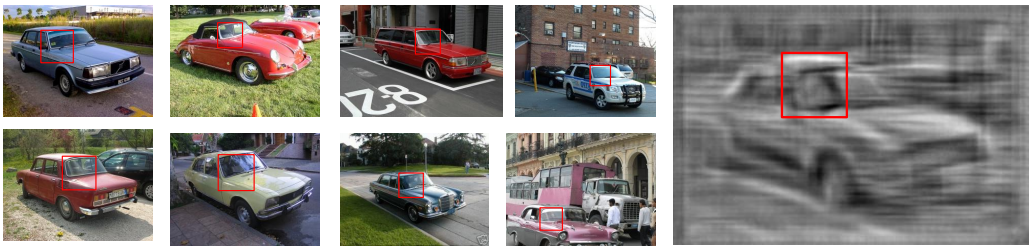


Figure 5.5: An example of our *thing* model running in "generative mode." As we run the alignment procedure, we generate each HOG cell in the prediction image as the average of the cells it corresponds to in the predictor images (we allow the algorithm to make a fresh prediction for each cell at each 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 [210].

Here, ρ is the same as in Equation 5.6. We normalize $\omega^i_{\mathcal{C}[t]}$ such that $\sum_i \omega^i_{\mathcal{C}[t]} = 1$ (since our goal is to compute a relative weighting between images), and we cap $\omega^i_{\mathcal{C}[t]}$ at $1/3$ to prevent any single image from dominating the prediction. When deciding which element to hold out and predict during our object discovery pipeline, compute the "usage score" U_i of a given patch i as:

$$U_i = \sum_t \omega^i_{\mathcal{C}[t]} * c_{\mathcal{C}[t]} \tag{5.28}$$

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 5.23) exclusively, or it 'mimics' the *stuff* model exactly (i.e. it sets its conditional distribution $P_T(h|H^0_{\mathcal{C}[1:t-1]}, H^0_{\mathcal{P}})$ equal to $P_S(h|H^0_{\mathcal{C}[1:t-1]}, H^0_{\mathcal{P}})$ for all h). Note that when the *thing* model mimics the *stuff* model, $p^T_{\mathcal{C}[t]}$ will be equal to $p^S_{\mathcal{C}[t]}$, and hence the value of $H^0_{\mathcal{C}}$ will have no effect on the likelihood ratio score. To determine when the *thing* model should mimic the *stuff* model, we use two heuristics. First, we measure how well the *thing* model has predicted the query image in regions near the current prediction region. Second, to do a better job estimating the bounds of the object, we measure whether the *thing* model believes $\mathcal{C}[t]$ corresponds to regions in other images that were predicted poorly.

Computing when to mimic the *stuff* model We have two criteria to determine when the *thing* model mimics the *stuff* model: (1) We measure how well the *thing* model has predicted

the query image in cells near the current prediction region, and (2) we measure whether the *thing* model believes $\mathcal{C}[t]$ corresponds to cells in other images that were predicted poorly. For (1), recall that $c_{x,y}$ (Equation 5.7) 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 $\pi^i_{u,v}$ be the probability of the *thing* model for image at location (u, v) in that image (i.e., $\hat{p}^T_{u,v}/(\hat{p}^T_{u,v} + \hat{p}^S_{u,v})$ computed while predicting image i, without allowing mimicry). We can integrate these values as

$$\beta_{\mathcal{C}[t]} = \sum_i \hat{\omega}^i_{\mathcal{C}[t]} \sum_{u,v} w^i_{\mathcal{C}[t],u,v} \pi^i_{u,v} \tag{5.29}$$

Here, the sum over i sums over only those images i where prediction has already been computed; $w^i_{\mathcal{C}[t],u,v}$ is defined as in equation 5.24, and $\hat{\omega}^i_{\mathcal{C}[t]} \propto \omega^i_{\mathcal{C}[t]}$ as defined in equation 5.24 but re-normalized to sum to 1 over the smaller set of images. We mimic if $\beta_{\mathcal{C}[t]} * c_{\mathcal{C}[t]} < .3$, a threshold we determined empirically. If we haven't yet performed verification on any other images, we simply set $\beta_{\mathcal{C}[t]} = .5$.

5.3.3 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 5.28), 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.

5.4 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.

5.4.1 Quantitative Results: Purity-Coverage on PASCAL VOC 2007

Following the experimental setup of [190], 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 [39], 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

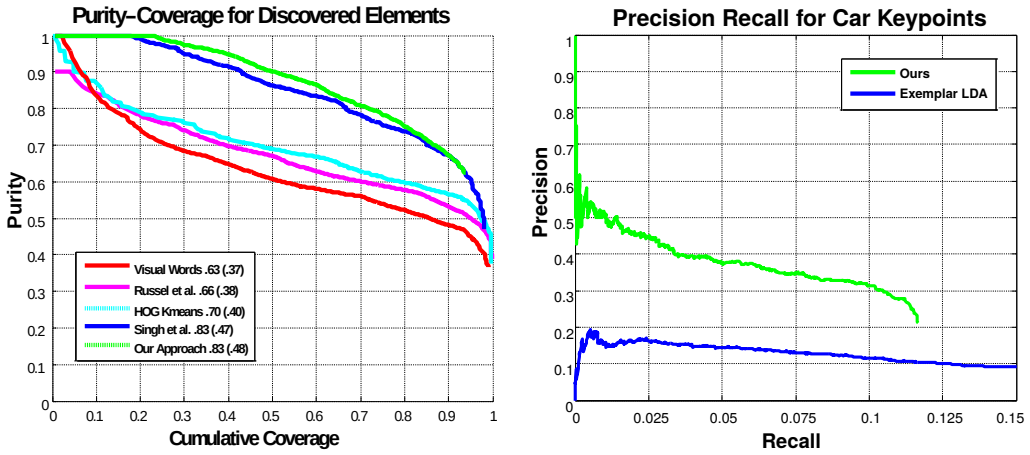


Figure 5.6: Left: purity vs coverage for objects discovered on a subset of Pascal VOC 2007. The numbers in the legend indicate area under the curve (AUC). In parentheses is the AUC up to a coverage of .5 (lower ranked clusters generally aren't objects). Right: precision-recall for car keypoints transferred via unsupervised correspondence.

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 [190], 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 5.6 left. We slightly outperform [190], especially for the high purity regime (we get nearly 10% extra coverage before we make our first mistake). However, note that [190] 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 [194] Specifically, given a selection χ of clusters, let $s^{\chi}_{i,j}$ be the log likelihood ratio for the j'th highest-scoring patch in image i out of all clusters contained in χ. We greedily select clusters to include in χ to maximize $\sum_{i,j} 2^{-j} s^{\chi}_{i,j}$, i.e. exponentially discounting the scores of patches from the same image.

Figure 5.7: 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 5.8: 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.

5.4.2 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 5.7 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 5.3.2) to their locations in the image. Black borders indicate either the edge of the image, or the display cut off. Figure 5.8 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 5.4.1 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 [43]. Our full ranking is available online at: `http://graphics.cs.cmu.edu/projects/contextPrediction/`.

5.4.3 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 [82]. The goal is to predict keypoint locations in an unlabeled image using other labeled images, in our case, without knowing it is a car. To perform transfer, we begin with the 1,000 objects discovered from PASCAL VOC 2011 above. For a given test image, we first use Exemplar-LDA to find which of the 1,000 clusters fire in this image. For each detection, we perform our context-based verification. Each verification uses the top 20 predictor images according to verification score, and we transfer keypoints from all 20 images using f. We make the assumption that each keypoint occurs only once per instance, so we score each keypoint and take, for each label, the keypoint with maximum score. We compute this score as $c_{\mathcal{C}[t]} * w^i_{\mathcal{C}[t],u,v} * s$, where the points (x, y) and (u, v) are the points that the keypoint was transferred to and from, respectively, s is the overall verification probability (likelihood of *thing* over likelihood of *thing* + likelihood of *stuff*) for the patch, and $c_{\mathcal{C}[t],x,y}$ is the per-point confidence computed in Appendix A. If multiple verifications happen for the same image (i.e. multiple patches are detected) and the regions considered to be *thing* overlap (intersection over union greater than .2), then the keypoints predicted for those verifications are merged and de-duplicated so that there is only one prediction per label.

Unfortunately, the evaluation of [82] isn't suitable in our situation, since they only mea-

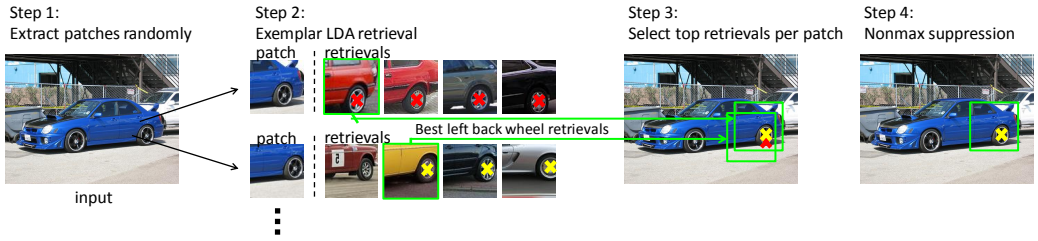


Figure 5.9: Intuition behind our baseline keypoint prediction algorithm. See text for details.

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 5.6 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 5.10 shows the raw output for one image, after thresholding the confidence of keypoint transfers.

5.5 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 5.9. 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

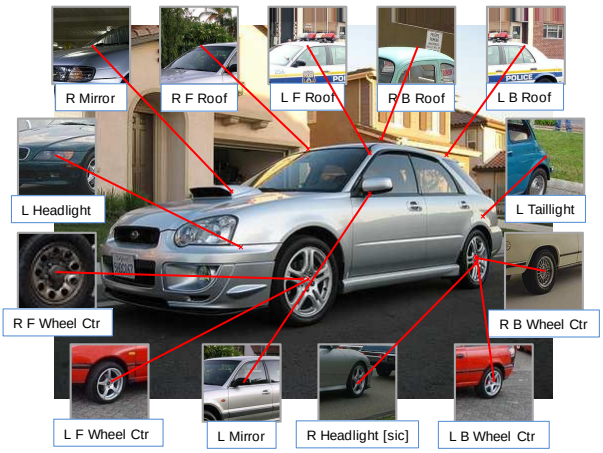


Figure 5.10: An example of keypoint transfer. For each predicted keypoint, we show a patch from the image where that keypoint was transferred from. These correspondences are discovered without any supervision. Note that multiple different labels are predicted for the wheel centers. This happens because 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 5.9, 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 5.9, 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 chapter), the baseline can achieve higher recall.

5.6 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 $P(H^0_{\mathcal{C}[t]}|H^0_{\mathcal{C}[1:t-1]}, H^0_{\mathcal{P}})$ as a more classical machine learning problem: estimating conditional distributions is, after all, a classical *discriminative* learning setup. Our algorithm did not use standard discriminative learning algorithms due to a lack of reliable training data, but in a supervised setting our *thing* model might be replaced with a far simpler regression model trained discriminatively.

Chapter 6

Deep Visual Representation Learning by Unsupervised Context Prediction

6.1 Introduction

Recently, new computer vision methods have leveraged large datasets of millions of labeled examples to learn rich, high-performance visual representations [111]. Yet efforts to scale these methods to truly Internet-scale datasets (i.e. hundreds of **b**illions 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 [5, 28, 137, 151]. 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 [137].

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 6.1 and 6.2. 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

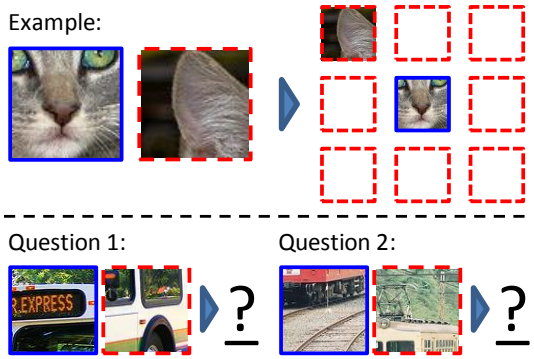


Figure 6.1: Our task for learning patch representations involves randomly sampling a patch (blue) and then one of eight possible neighbors (red). Can you guess the spatial configuration for the two pairs of patches? Note that the task is much easier once you have recognized the object!

Answer key: Q1: Bottom right Q2: Top center

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" [2]). 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 [9, 208] 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 [154], and such autoencoders may be stacked to form a deep representation [116, 118]. (however, only [116] 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" [137] 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

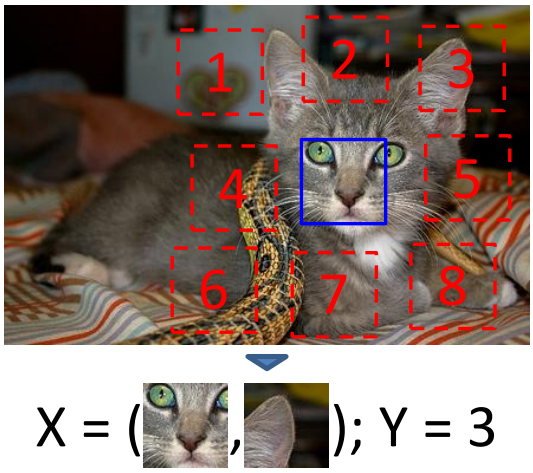


Figure 6.2: The algorithm receives two patches in one of these eight possible spatial arrangements, without any context, and must then classify which configuration was sampled.

kind of visual "fill in the blank" task, but, again, one runs into the problem of determining whether the predictions themselves are correct [40], unless one cares about predicting only very low-level features [45, 115, 202]. To address this, [134] predicts the appearance of an image region by consensus voting of the transitive nearest neighbors of its surrounding regions. Our previous work in chapter 5 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 [28, 151]. 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 6.2.

6.2 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 6.2) that might have been sampled (i.e. a softmax output).

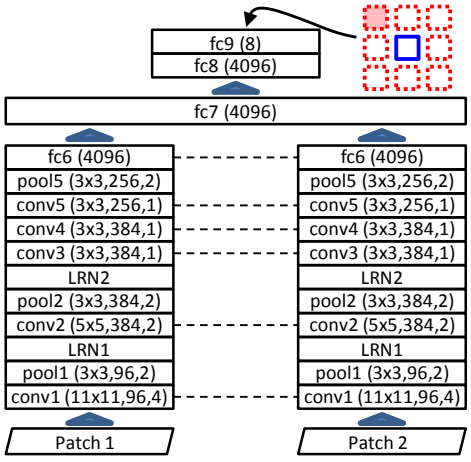


Figure 6.3: Our architecture for pair classification. Dotted lines indicate shared weights. 'conv' stands for a convolution layer, 'fc' stands for a fully-connected one, 'pool' is a max-pooling layer, and 'LRN' is a local response normalization layer. Numbers in parentheses are kernel size, number of outputs, and stride (fc layers have only a number of outputs). The LRN parameters follow [111]. 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 6.3: a pair of AlexNet-style architectures [111] 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 6.2.

6.2.1 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 could give away the correct answer. Therefore, we also randomly jitter each patch location by up to 7 pixels (see Figure 6.2).

However, even these precautions are not enough: we were surprised to find that, for

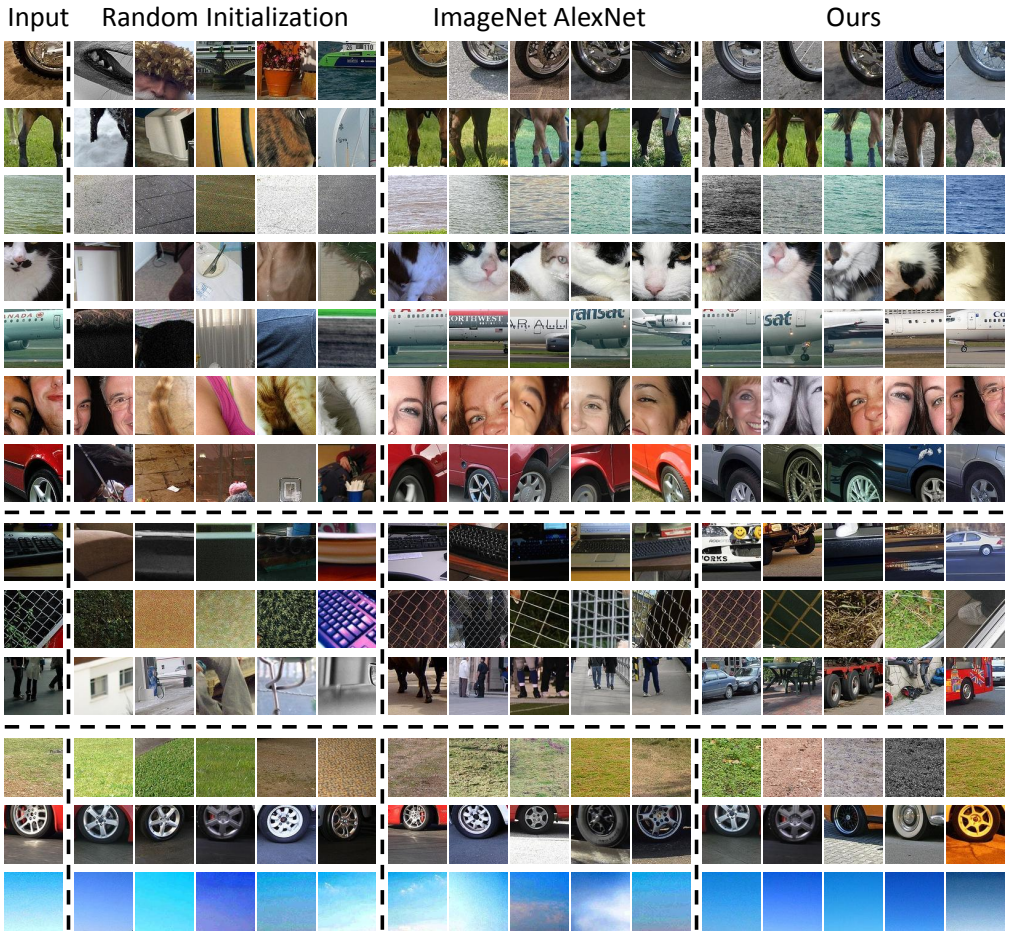


Figure 6.4: Examples of patch clusters obtained by nearest neighbors. The query patch is shown on the far left. Matches are for three different features: fc6 features from a random initialization of our architecture, AlexNet fc7 after training on labeled ImageNet, and the fc6 features learned from our 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 [14, p. 76]. A ConvNet, it turns out, can learn to localize a patch relative to the lens itself (see Section 6.3.2) 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

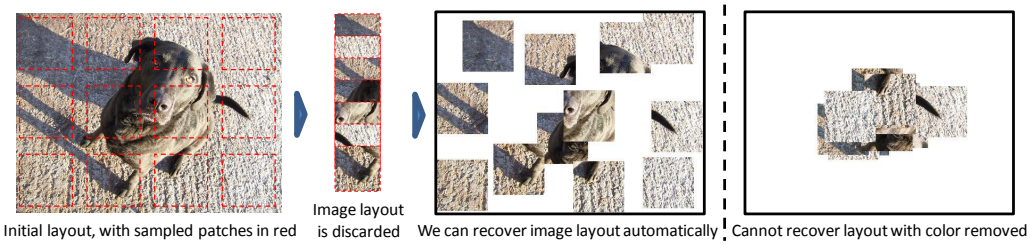


Figure 6.5: We trained a network to predict the absolute (x, y) coordinates of randomly sampled patches. Far left: input image. Center left: extracted patches. Center right: the location the trained network predicts for each patch shown on the left. Far right: the same result after our color projection 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 [98], and train on the ImageNet [35] 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 [94], 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.

6.3 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.

VOC-2007 Test	aero	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	mbike	person	plant	sheep	sofa	train	tv	mAP
DPM-v5 [53]	33.2	60.3	10.2	16.1	27.3	54.3	58.2	23.0	20.0	24.1	26.7	12.7	58.1	48.2	43.2	12.0	21.1	36.1	46.0	43.5	33.7
[26] w/o context	52.6	52.6	19.2	25.4	18.7	47.3	56.9	42.1	16.6	41.4	41.9	27.7	47.9	51.5	29.9	20.0	41.1	36.4	48.6	53.2	38.5
Regionlets [221]	54.2	52.0	20.3	24.0	20.1	55.5	68.7	42.6	19.2	44.2	49.1	26.6	57.0	54.5	43.4	16.4	36.6	37.7	59.4	52.3	41.7
Scratch-R-CNN [4]	49.9	60.6	24.7	23.7	20.3	52.5	64.8	32.9	20.4	43.5	34.2	29.9	49.0	60.4	47.5	28.0	42.3	28.6	51.2	50.0	40.7
Scratch-Ours	52.6	60.5	23.8	24.3	18.1	50.6	65.9	29.2	19.5	43.5	35.2	27.6	46.5	59.4	46.5	25.6	42.4	23.5	50.0	50.6	39.8
Ours-projection	58.4	62.8	33.5	27.7	24.4	58.5	68.5	41.2	26.3	49.5	42.6	37.3	55.7	62.5	49.4	29.0	47.5	28.4	54.7	56.8	45.7
Ours-color-dropping	60.5	66.5	29.6	28.5	26.3	56.1	70.4	44.8	24.6	45.5	45.4	35.1	52.2	60.2	50.0	28.1	46.7	42.6	54.8	58.6	46.3
Ours-Yahoo100m	56.2	63.9	29.8	27.8	23.9	57.4	69.8	35.6	23.7	47.4	43.0	29.5	52.9	62.0	48.7	28.4	45.1	33.6	49.0	55.5	44.2
ImageNet-R-CNN [66]	64.2	69.7	50	41.9	32.0	62.6	71.0	60.7	32.7	58.5	46.5	56.1	60.6	66.8	54.2	31.5	52.8	48.9	57.9	64.7	54.2
K-means-rescale [110]	55.7	60.9	27.9	30.9	12.0	59.1	63.7	47.0	21.4	45.2	55.8	40.3	67.5	61.2	48.3	21.9	32.8	46.9	61.6	51.7	45.6
Ours-rescale [110]	61.9	63.3	35.8	32.6	17.2	68.0	67.9	54.8	29.6	52.4	62.9	51.3	67.1	64.3	50.5	24.4	43.7	54.9	67.1	52.7	51.1
ImageNet-rescale [110]	64.0	69.6	53.2	44.4	24.9	65.7	69.6	69.2	28.9	63.6	62.8	63.9	73.3	64.6	55.8	25.7	50.5	55.4	69.3	56.4	56.5
VGG-K-means-rescale	59.5	58.5	25.9	33.7	11.5	57.5	65.3	46.8	20.1	44.7	35.9	38.0	60.8	59.1	47.5	21.8	34.7	41.8	61.0	50.8	43.7
VGG-Ours-rescale	71.1	72.4	54.1	48.2	29.9	75.2	78.0	71.9	38.3	60.5	62.3	68.1	74.3	74.2	64.8	32.6	56.5	66.4	74.0	60.3	61.7
VGG-ImageNet-rescale	76.6	79.6	68.5	57.4	40.8	79.9	78.4	85.4	41.7	77.0	69.3	80.1	78.6	74.6	70.1	37.5	66.0	67.5	77.4	64.9	68.6

Table 6.1: Mean Average Precision on VOC-2007.

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.

6.3.1 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 6.3, 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 6.4. 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 6.4, 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.

6.3.2 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 6.5 shows one such result. Applying the proposed "projection" scheme increases the error on the top 10% of images to .321.

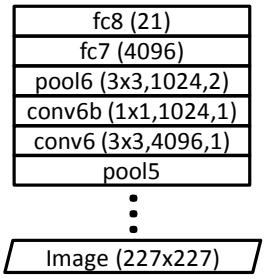


Figure 6.6: Our architecture for PASCAL VOC detection. Layers from conv1 through pool5 are copied from our patch-based network (Figure 6.3). The new 'conv6' layer is created by converting the fc6 layer into a convolution layer. Kernel sizes, output units, and stride are given in parentheses, as in Figure 6.3.

6.3.3 Object Detection

Previous work on the PASCAL VOC challenge [49] 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 [4, 66]. 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 [66]. 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 6.6. As above, we use only one stack from Figure 6.3. 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 [130]. 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 [66] (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 [66] and [4].

Table 6.1 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 [66]. 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 [205], we also performed a preliminary experiment pre-training on a randomly-selected 2M subset of the Yahoo/Flickr 100-million Dataset [204], 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 [189]

	Lower Better		Higher Better		
	Mean	Median	11.25°	22.5°	30°
Scratch	38.6	26.5	33.1	46.8	52.5
Unsup. Tracking [220]	34.2	21.9	35.7	50.6	57.0
Ours	**33.2**	21.3	36.0	51.2	57.8
ImageNet Labels	33.3	**20.8**	**36.7**	**51.7**	**58.1**

Table 6.2: Accuracy on NYUv2.

(16-layer) network, shown as "Ours-VGG" in Table 6.1.

In the above fine-tuning experiments, we removed the batch normalization layers by estimating the mean and variance of the conv- and fc- layers, and then rescaling the weights and biases such that the outputs of the conv and fc layers have mean 0 and variance 1 for each channel. Recent work [110], however, has shown empirically that the scaling of the weights prior to finetuning can have a strong impact on test-time performance, and argues that our previous method of removing batch normalization leads too poorly scaled weights. They propose a simple way to rescale the network's weights without changing the function that the network computes, such that the network behaves better during finetuning. Results using this technique are shown in Table 6.1. Their approach gives a boost to all methods, but gives less of a boost to the already-well-scaled ImageNet-category model. Note that for this comparison, we used fast-rcnn [65] to save compute time, and we discarded all pre-trained fc-layers from our model, re-initializing them with the K-means procedure of [110] (which was used to initialize all layers in the "K-means-rescale" row). Hence, the structure of the network during fine-tuning and testing was the same for all models.

Considering that we have essentially infinite data to train our model, we might expect that our algorithm should also provide a large boost to higher-capacity models such as VGG [189]. To test this, we trained a model following the 16-layer structure of [189] for the convolutional layers on each side of the network (the final fc6-fc9 layers were the same as in Figure 6.3). We again fine-tuned the representation on PASCAL VOC using fast-rcnn, by transferring only the conv layers, again following Krähenbühl et al. [110] to re-scale the transferred weights and initialize the rest. As a baseline, we performed a similar experiment with the ImageNet-pretrained 16-layer model of [189] (though we kept pre-trained fc layers rather than re-initializing them), and also by initializing the entire network with K-means [110]. Training time was considerably longer—about 8 weeks on a Titan X GPU—but the the network outperformed the AlexNet-style model by a considerable margin. Note the model initialized with K-means performed roughly on par with the analogous AlexNet model, suggesting that most of the boost came from the unsupervised pre-training.

6.3.4 Geometry Estimation

The results of Section 6.3.3 suggest that our representation is sensitive to objects, even though it was not originally trained to find them. This raises the question: Does our representation extract information that is useful for other, non-object-based tasks? To find out, we fine-tuned our network to perform the surface normal estimation on NYUv2 proposed in Fouhey et al. [59], following the finetuning procedure of Wang et al. [220] (hence, we compare directly to the unsupervised pretraining results reported there). We used the color-dropping network, restructuring the fully-connected layers as in Section 6.3.3. Surprisingly, our results are almost equivalent to those obtained using a fully-labeled ImageNet model.

Figure 6.7: Object clusters discovered by our algorithm. The number beside each cluster indicates its ranking, determined by the fraction of the top matches that geometrically verified. For all clusters, we show the raw top 7 matches that verified geometrically. The full ranking is available on our project webpage.

One possible explanation for this is that the ImageNet categorization task does relatively little to encourage a network to pay attention to geometry, since the geometry is largely irrelevant once an object is identified. Further evidence of this can be seen in seventh row of Figure 6.4: the nearest neighbors for ImageNet AlexNet are all car wheels, but they are not aligned well with the query patch.

6.3.5 Visual Data Mining

Visual data mining [43,163,169,190], or unsupervised object discovery [72,178,192], 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 coordinates [43]). For automatic data mining, our approach from section 6.3.1 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 configuration. 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 image (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 [25] 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 [25]. 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 6.7. The results are visually comparable to our previous work in chapter 5, although we discover a few objects that were not found in chapter 5, 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 5, 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 5 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 5, we often re-discover the same object multiple times with different viewpoints, which accounts for most of the gaps between ranks in Figure 6.7. The main disadvantages of this algorithm relative to chapter 5 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 [205] in ImageNet, we also applied our algorithm to 15,000 Street View images from Paris (following [43]). The results in Figure 6.8 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 [43].

Quantitative Results

As part of the qualitative evaluation, we applied our algorithm to the subset of PASCAL VOC 2007 selected in [190]: 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 5. 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 gen-

Figure 6.8: Clusters discovered and automatically ranked via our algorithm (§ 6.3.5) from the Paris Street View dataset.

erate 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 6.9, we have gained substantially in terms of coverage, suggesting increased invariance for our learned feature. However, we have also lost some highly-pure clusters compared to chapter 5—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 5 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.

6.3.6 Deep Dream

If a network can detect a semantic concept like a face, we might expect that the network would rely on a particular unit to do so. If so, then a face-like pattern should drive that unit's activation more than any other possible image. Several frameworks have used this property as the basis for visualizing neural networks trained on ImageNet labels [132, 143, 231]. For instance, "Deep Dream" [143] explicitly optimizes an image to drive specific units, and the results are often recognizable as object parts. However, will these approaches find similar semantic entities in our networks, where semantics are not necessarily represented as strongly? To find out, Alexander Mordvintsev kindly ran Deep Dream [143] on our VGG-style model, separately visualizing each channel of `conv5_3`. While many of the units depicted simple patterns or textures (especially arcs), many others were clearly semantically

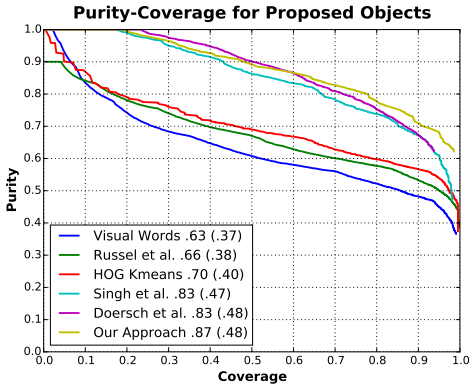


Figure 6.9: Purity vs coverage for objects discovered on a subset of PASCAL VOC 2007. The numbers in the legend indicate area under the curve (AUC). In parentheses is the AUC up to a coverage of .5.

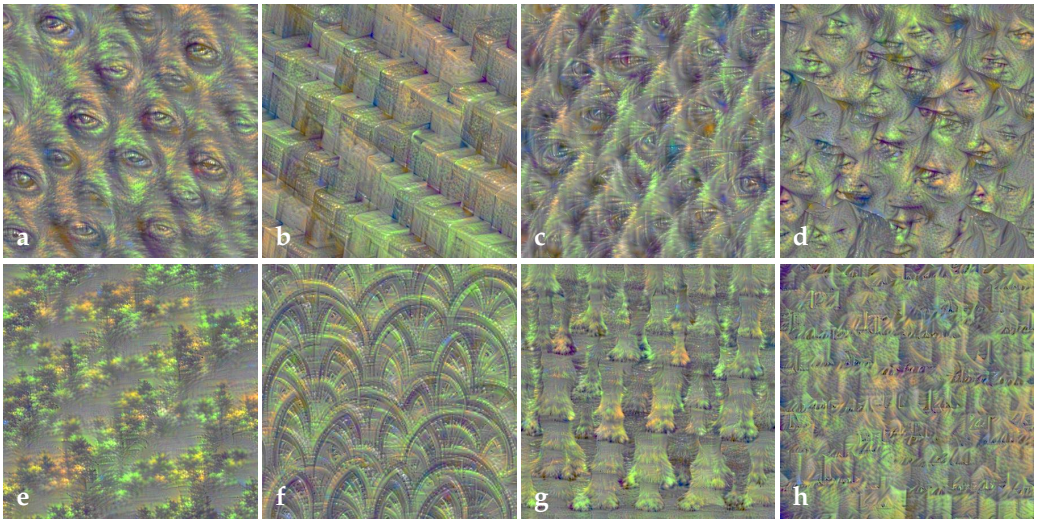


Figure 6.10: The output of Deep Dream applied separately to 8 selected filters in `conv5_3` in our VGG-style network (credit: Alexander Mordvintsev). Our interpretation: a) eyes, b) cubes/stairs, c) left side of an animal face, d) human faces, e) trees, f) circles/arcs, g) paws, h) text.

or geometrically meaningful. We show some selected visualizations in Figure 6.10. Overall, these visualizations again confirm that our relative position task results in the network learning semantics.

6.3.7 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 6.2. 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.

Chapter 7

Discussion

Our work in this thesis is just one small piece of a much larger goal: machine understanding of images. We argue that the representation which encapsulates this understanding does not need to be determined manually, but rather, can arise by itself provided that the right machine learning algorithm is trained on the right task. However, many questions remain, especially regarding how to choose the task and how to use the representation once it is learned. In this chapter, we briefly discuss the overall landscape of pretext tasks that may apply to vision, with an eye for other approaches that may learn even better representations than those shown here. We also briefly discuss potential future applications for representations learned through pretext tasks.

7.1 The Wider Landscape of Pretext Tasks

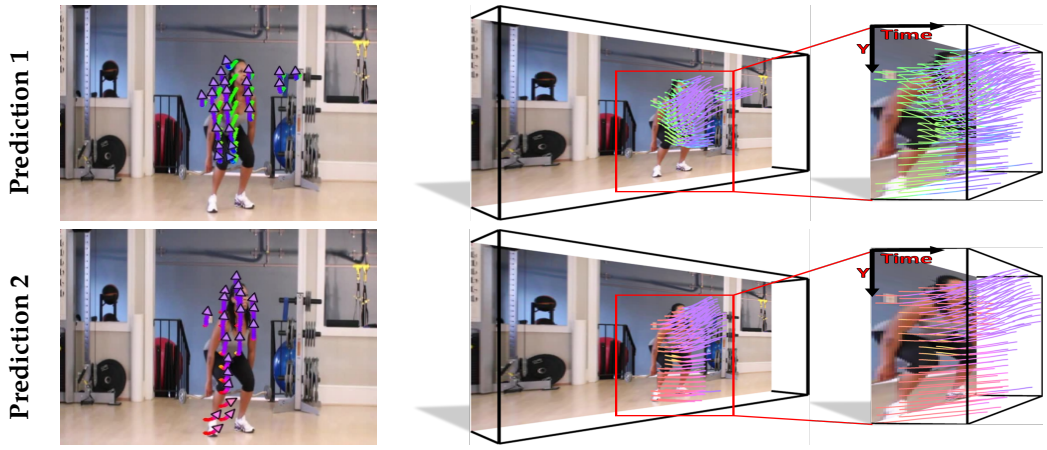
In this thesis, we have focused on two different kinds of supervision: image-level labels, like GPS and scene categories, and spatial context. However, we argue that many more pretext tasks exist that can train good representations, and furthermore, we cannot yet prove that our algorithms are the optimal way to exploit our supervisory signals. Here, we discuss a few alternative formulations that might also yield strong representations with minimal human effort.

Alternative objective functions for contextual learning In this thesis, we focused on two approaches to contextual learning: learning to predict raw low-level features, and learning to predict the relative positions of patches. The second approach currently seems more promising since we achieved better results with a far simpler algorithm. Why is this? One potential explanation is that the formulation of the relative prediction task ensures that low-level structure is unlikely to be useful. On the other hand, when predicting low-level features or pixels [155], even deep networks seem to spend most of their capacity on low-level dependencies. However, this is just conjecture: it is far from clear what other factors might be important. Unfortunately, the design space of training objectives to exploit context is very large. We have done some preliminary experiments to explore more of the space, but this job is far from complete. First, we tried expanding the space of possible labels: rather than 8 possible locations for the second patch relative to the first, we tried a patch sampling pattern where the second patch could come from one of roughly 100 spatial bins relative to

the first. Second, we tried using training examples that consisted of 3 rather than 2 patches. Finally, we tried a more complicated formulation where the network actually needed to predict the features of contextual patches. Specifically, we trained network which took in a patch and produced (1) a "template" (i.e., a fc6-sized linear operator which could be applied to the fc6 representation of every other patch in the image, thus producing a heatmap of 'firings' across the rest of the image), and (2) a prediction of where that "template" was going to fire. We scored this final model by measuring the similarity between the predicted firings and the actual firings of the template. The intuition was that if the input patch contains, for example, a car wheel, then the network can make a very specific prediction that the image should contain a second car wheel at a particular distance away. On the other hand, given a patch containing grass, no such specific prediction would be possible. Hence, this objective function should require a much more detailed understanding of object parts. However, none of these formulations produced qualitative behavior that was significantly better than the results presented in Chapter 6, so we never evaluated them in detail. Other authors have explored using 9 patches rather than 2 in each training example [150], which boosts performance while reducing training time. Predicting pixels directly instead of predicting relative locations has also been explored for representation learning [159], which has the added benefit of enabling graphics applications like hole filling.

Relation to Generative Models One of the most prominent lines of recent work in unsupervised visual learning is to directly model the distribution of natural images [70, 108, 165, 170]. This approach is not entirely unrelated to ours: both of the contextual models presented in this work are, in some sense, generative. Our approach in Chapter 5 aimed to generate the low-level features outside of a patch, and produced a full probability distribution over those features. However, we did not train that model using all the data; instead, we separated "things" from "stuff," and only modeled the "things." Likewise, an ideal generative model could produce the exact same predictions we produced in Chapter 6: we can calculate the probability of two patches appearing with a particular offest by conditioning the generative model on those two patches, and marginalizing over all other pixels. In some sense, our relative position network is modeling a tiny slice of a generative model. Current generative models can draw synthetic images that are remarkably believable for simple datasets, but their usefulness for *representing* images remains uncertain. As noted above, part of the problem may be that generative models capture the entire image formation process, including low-level image properties. We believe that the relationship between these models should be explored further. Generative models might become more sensitive to semantics if they marginalize across some irrelevent details as our formulations do. For example, it may be possible to extract features using pretext tasks like relative position, and then create a purely generative model of those features.

Other Sources of Supervision This work has described context, GPS, and scene category labels as supervisory signal for training representations. However, many other cues have been proposed for representation learning, with varying levels of success (see Chapter 2). Considering that some of these have been suggested only recently, it is unlikely that we have covered the full space of cues that will work for representation learning. We conjecture that training a network to solve any task which relies on semantics should result in a representation that captures semantics. Hence, we might expect that cues like user interaction data (how long users look at photos on photo sharing websites, or who they share photos with), or image framing (e.g. training a network to predict whether a patch is in the center



(a) Trajectories on Image (b) Trajectories in Space-Time

Figure 7.1: Given a single frame, our algorithm predicts motion over the course of one second. This woman doing squats might move either up or down, so our framework predicts multiple possible futures. The directions of the trajectories at each point in time are color-coded according to the square on the right. Left: the projection of two predicted motions on the image plane. Right: a full view of those two sets of trajectories in space time.

or near the periphery of an image) might result in useful representations. Another exciting source of supervision that the community is just beginning to explore is prediction in time [136, 212, 213]. Predicting what will happen in the future is one of the most important tasks any agent must perform; therefore, visual prediction should drive general-purpose semantics in a visual representation. In a collaboration with Jacob Walker and Abhinav Gupta, I explored visual representation learning by predicting one second into the future given a single frame in a video. In this work, we focused on predicting the dense trajectory of pixels in a scene, i.e., what will move in the scene, where it will travel, and how it will deform over the course of one second, and proposed a conditional variational autoencoder as a solution to this problem. In this framework, direct inference from the image shapes the distribution of possible trajectories, while latent variables encode any necessary information that is not available in the image. This method can successfully predict events in a wide variety of scenes, and can produce multiple different predictions when the future is ambiguous. The algorithm was trained on thousands of diverse, realistic videos and required no human labeling. In addition to non-semantic action prediction, we found that our method learned a representation that is applicable to semantic vision tasks, notably object detection. Some examples of predictions can be seen in Figure 7.1, and more information can be found in [1]. Overall, these results suggest that the future can serve as an effective source of supervision for representation learning.

Combining Methods Even though each chapter of this thesis chose a single type of supervision to train each model, our goal is a single representation that works for as many tasks as possible. This suggests that diversity in the training tasks may be useful. Also,

we have so far evaluated using only weakly supervised and unsupervised methods, but nothing we know of would prevent these from working alongside datasets with strong labels. Although it would be straightforward to train a network with multiple different loss functions, each of which corresponds to a different pretext tasks, there is currently very little published research showing that multi-task learning is beneficial in neural networks. Hence, many questions remain unanswered. For example, how should we weight different tasks in the loss function? Clearly labeled images are more informative than unlabeled ones, so perhaps they should get more weight. Another potential issue is that each task brings with it trivial shortcuts that solve the problem: for example, the chromatic aberration issue we described in Section 6.3.2. What prevents a single network from learning all these trivial shortcuts along with semantics? One approach is to assume that semantic information should be useful in all (or most) pretext tasks, whereas trivial shortcuts are useful in just one. Chromatic aberration, for example, is unlikely to be useful unless we need to analyze properties of the lens. This means that attempts to identify and suppress parts of the representation that are specific to one task might lead to a more semantically-driven representation.

7.2 Future Applications

A key advantage of the approach described in this thesis is that the representations are not tied to a specific task: we assume it is possible to learn concepts that are not specifically described by the labels. Given that computer vision has been driven largely by numeric benchmarks recently (which require well-defined labels), in this section we want to encourage researchers to step back for a moment, and remember that many of the problems vision aims to solve are actually quite hard to label. Hence, we briefly speculate on a few such application domains where we believe pretext tasks might help.

Generating Discriminative Pixels Consider an image editing program like Photoshop or an augmented reality system like HoloLens. A common task in such systems is to insert content like characters or objects into a scene. Making this process easy, and making the results convincing, are both open problems. Even when the computer knows what object to insert and has many examples of that object, it is difficult to make the inserted object match the input image. While considerable progress has been made with respect to understanding physical constraints like geometry or lighting [105,219], many of the required constraints are more semantic than physical. For example, a dog inserted into a living room should appear on the floor and not on the TV; a character inserted into an animation sequence should have the same animation style as the rest of the sequence; a virtual assistant inserted into a party should wear clothing similar to what the other attendees are wearing. For the computer system to succeed in these scenarios, it must first perceive and understand semantics and contextual relationships that appear throughout the world. In these cases, the necessary information would be quite difficult to label. However, in this thesis, we presented methods to learn about style without labeling it, and also showed how an algorithm can begin to model context without supervision. We believe that this may serve as a first step toward a more semantically-aware method of generating images.

Visualization and Visual Teaching A core component of this thesis is *visual data mining*: starting from large datasets of images and finding interesting recurring visual patterns. We

focused on visual patterns that predicted something: e.g., the city label, or something in the context. However, once an algorithm has found such a pattern, how should the computer explain it to a human? In the case of Paris, we were somewhat lucky: we could show sets of patches all depicting the same thing, and the parts of the patch that are consistent across all examples can show a person what is important about that visual element. However, clearly some visual patterns are more complicated than this. For example, how can we describe the visual style of computer products from the 1990s? With curved gray plastic and rubber oval buttons, these electronics are often distinctive and recognizable, yet the overall shape depends more on the function of a particular appliance than the style. Even if we have an algorithm—e.g., a CNN—that can distinguish what decade a product came from, how can we display that information in a way that a human would understand? One potentially better approach might to generate a style on top of an object which does not already have it. Currently, algorithms are starting to appear which can generate painterly style [63], and others which can visualize the discriminative features that a deep network detects [132,143, 231]. It is possible that future work in this direction will allow us to generate isolated stylistic elements, which would hopefully make it clear to humans what the algorithm considers discriminative.

Representations for Agents One of the overall goals of artifical intelligence is to create agents: programs that interact with an environment over time to accomplish a goal. Unlike standard computer vision algorithms, the predictions an agent needs to make are potentially very complicated, and evaluating those predictions is hard. For example, consider a robot cleaning and organizing a room. The robot needs to decide what will be the final configuration of the room (a high-dimensional, continuous state space describing the placement of all visible objects), and also needs to plan where all the objects will be during the cleaning process (no part of the floor can be vacuumed while it is occupied by an object). Part of a human's ability to do this kind of planning is visual: just like Hume can imagine Paris by closing his eyes, we can imagine the final configuration of a room when it is clean. But importantly, just as Hume cannot imagine every detail of Paris, our visual imagination only has as much detail as is required to accomplish the task. Thus we can do our planning without wasting computation on exhaustive physical calculations. It is far from clear how this kind of representation should work, so it would be difficult to train a computer by explicitly labeling what we want the computer to represent. Instead, it may be easier to design tasks which require an agent's representations to capture general-purpose information about the environment, e.g. semantics and geometry. Solving these task should also require the agent to "imagine" representations of the world in the future, i.e., predict state of the world after actions have been taken. A few relatively simple tasks can likely train the agent to imagine the short-term future, and hopefully, these simple predictions can be chained together to make more complicated, longer-term predictions as needed for complex tasks. As before, there is no fundamental reason these training tasks need to be useful ones. After all, humans and animals routinely work to attain arbitrary goals during play. Perhaps pretext tasks—chosen by the algorithm in a way humans and animals choose to play—may be sufficient for this type of learning.

Chapter 8

Conclusion

This thesis proposed several methods for learning visual representations from image collections where the labels are either weak or nonexistent. Image-level labels like GPS tags and scene categories enabled representations that cluster similar mid-level discriminative patches in ways that make sense to humans, and which perform well on real tasks like classification. Context served as a similarly informative supervisory signal for grouping patches and learning deep representations when no labels are present. Overall, we hope this work changes how the vision community thinks about the role of labels in computer vision. Machine learning, on top of the right representation, can be more than a simple function approximator, and should not be viewed as a "black box" mapping input images to output labels. Instead, we argue that learned representations will capture any additional semantics that *help* solve the task given to the computer. We urge the community to exploit this property, both to learn better general-purpose visual representations from a wealth of "free" supervisory signals, and to solve problems where human labeling is difficult or impossible.

Bibliography

[1] An uncertain future: Forecasting from static images using variational autoencoders. 2.3.2, 7.1

[2] E. H. Adelson. On seeing stuff: the perception of materials by humans and machines. In *Photonics West 2001-Electronic Imaging*, 2001. 5.1, 6.1

[3] P. Agrawal, J. Carreira, and J. Malik. Learning to see by moving. In *ICCV*, 2015. 2.3.1

[4] P. Agrawal, R. Girshick, and J. Malik. Analyzing the performance of multilayer neural networks for object recognition. In *ECCV*. 2014. 6.3.2, 6.3.3, 6.3.3

[5] R. K. Ando and T. Zhang. A framework for learning predictive structures from multiple tasks and unlabeled data. *JMLR*, 6:1817–1853, 2005. 1.2, 6.1

[6] M. Aubry, B. C. Russell, and J. Sivic. Painting-to-3d model alignment via discriminative visual elements. *ACM Transactions on Graphics (TOG)*, 33(2):14, 2014. 2.3.1

[7] A. Bansal, A. Shrivastava, C. Doersch, and A. Gupta. Mid-level elements for object detection. *arXiv preprint arXiv:1504.07284*, 2015. 4.7

[8] H. B. Barlow. Summation and inhibition in the frog's retina. *The Journal of physiology*, 119(1):69, 1953. 2.1

[9] Y. Bengio, E. Thibodeau-Laufer, G. Alain, and J. Yosinski. Deep generative stochastic networks trainable by backprop. *ICML*, 2014. 1.2, 2.3.2, 6.1

[10] T. L. Berg, D. Forsyth, et al. Animals on the web. In *CVPR*, 2006. 2.3.1

[11] T. Berg and A. Berg. Finding iconic images. In *CVPR*, 2009. 3.1

[12] L. Bossard, M. Guillaumin, and L. Van Gool. Food-101–mining discriminative components with random forests. In *ECCV*. 2014. 2.3.1

[13] L. Bourdev and J. Malik. Poselets: Body part detectors trained using 3D human pose annotations. In *ICCV*, 2009. 3.4

[14] D. Brewster and A. D. Bache. *Treatise on optics*. Blanchard and Lea, 1854. 6.2.1

[15] M. C. Burl, M. Weber, and P. Perona. A probabilistic approach to object recognition using local photometry and global geometry. In *ECCV*. 1998. 2.2

[16] J. Carreira, P. Agrawal, K. Fragkiadaki, and J. Malik. Human pose estimation with iterative error feedback. *arXiv preprint arXiv:1507.06550*, 2015. 2.3

[17] H. E. Cetingul and R. Vidal. Intrinsic mean shift for clustering on Stiefel and Grassmann manifolds. In *CVPR*, 2009. 4.2

[18] D. Chen, G. Baatz, K. Koser, S. Tsai, R. Vedantham, T. Pylvanainen, K. Roimela, X. Chen, J. Bach, M. Pollefeys, B. Girod, and R. Grzeszczuk. City-scale landmark identification on mobile devices. In *CVPR*, 2011. 3.2

[19] X. Chen, A. Shrivastava, and A. Gupta. Neil: Extracting visual knowledge from web data. In *ICCV*, 2013. 2.3.1

[20] X. Chen, A. Shrivastava, and A. Gupta. Enriching visual knowledge bases via object discovery and segmentation. In *CVPR*, 2014. 2.3.1

[21] Y. Cheng. Mean shift, mode seeking, and clustering. *PAMI*, 17(8):790–799, 1995. 4.1, 4.2, 4.2

[22] M. Cho, S. Kwak, C. Schmid, and J. Ponce. Unsupervised object discovery and localization in the wild: Part-based matching with bottom-up region proposals. *CVPR*, 2015. 2.3.1

[23] J. Choi, M. Rastegari, A. Farhadi, and L. Davis. Adding unlabeled samples to categories by learned attributes. In *CVPR*, 2013. 2.3.1

[24] O. Chum, M. Perdoch, and J. Matas. Geometric min-hashing: Finding a (thick) needle in a haystack. In *CVPR*, 2009. 2.3.2

[25] O. Chum, J. Philbin, J. Sivic, M. Isard, and A. Zisserman. Total recall: Automatic query expansion with a generative feature model for object retrieval. In *ICCV*, 2007. 6.3.5

[26] R. G. Cinbis, J. Verbeek, and C. Schmid. Segmentation driven object detection with Fisher vectors. In *ICCV*, 2013. 6.3.2

[27] R. G. Cinbis, J. Verbeek, and C. Schmid. Multi-fold MIL training for weakly supervised object localization. In *CVPR*, 2014. 2.3.1

[28] R. Collobert and J. Weston. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *ICML*, 2008. 1.2, 6.1, 6.1

[29] D. Comaniciu, V. Ramesh, and P. Meer. Real-time tracking of non-rigid objects using mean shift. In *CVPR*, 2000. 4.1

[30] D. Comaniciu, V. Ramesh, and P. Meer. The variable bandwidth mean shift and data-driven scale selection. In *ICCV*, 2001. 4.2, 4.4, 4.4

[31] D. Crandall, L. Backstrom, D. Huttenlocher, and J. Kleinberg. Mapping the world's photos. In *WWW*, 2009. 3.2, 3.3

[32] E. Crowley and A. Zisserman. The state of the art: Object retrieval in paintings using discriminative regions. In *BMVC*, 2014. 2.3.1

[33] D. Dai and L. Van Gool. Ensemble projection for semi-supervised image classification. In *ICCV*, 2013. 2.3.1

[34] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *CVPR*, 2005. 1.2, 1.1, 3.4, 3.4.1

[35] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *CVPR*, 2009. 1.1, 2.2, 6.2.1

[36] E. L. Denton, S. Chintala, R. Fergus, et al. Deep generative image models using aï£ij laplacian pyramid of adversarial networks. In *NIPS*, 2015. 2.3.2

[37] T. Deselaers, B. Alexe, and V. Ferrari. Localizing objects while learning their appearance. In *ECCV*. 2010. 2.3.1

[38] S. K. Divvala, A. Farhadi, and C. Guestrin. Learning everything about anything: Webly-supervised visual concept learning. In *CVPR*, 2014. 2.3.1

[39] C. Doersch, A. Gupta, and A. A. Efros. Mid-level visual element discovery as discriminative mode seeking. In *NIPS*, 2013. 1, 2.3.1, 5.1, 5.4.1

[40] C. Doersch, A. Gupta, and A. A. Efros. Context as supervisory signal: Discovering objects with predictable context. In *ECCV*. 2014. 1, 2.3.2, 6.1

[41] C. Doersch, A. Gupta, and A. A. Efros. Unsupervised visual representation learning by context prediction. In *ICCV*, 2015. 1, 2.3.2

[42] C. Doersch, T. Lee, G. Huang, and E. Miller. Temporal continuity learning for convolutional deep belief networks. 2010. 2.3.2

[43] C. Doersch, S. Singh, A. Gupta, J. Sivic, and A. A. Efros. What makes Paris look like Paris? *SIGGRAPH*, 2012. 1, 2.3.1, 4.1, 4.1, 4.5, 5.1, 5.2, 5.4.2, 6.3.5, 6.3.5

[44] C. Doersch, S. Singh, A. Gupta, J. Sivic, and A. A. Efros. What makes Paris look like Paris? *Communications of the ACM*, 58(12):103–110, 2015. 1

[45] J. Domke, A. Karapurkar, and Y. Aloimonos. Who killed the directed model? In *CVPR*, 2008. 5.2, 6.1

[46] S. Ebert, D. Larlus, and B. Schiele. Extracting structures in image collections for object recognition. In *ECCV*. 2010. 2.3.1

[47] A. A. Efros and T. K. Leung. Texture synthesis by non-parametric sampling. In *ICCV*, 1999. 5.1

[48] I. Endres, K. Shih, J. Jiaa, and D. Hoiem. Learning collections of part models for object recognition. In *CVPR*, 2013. 2.3.1, 4.1, 5.1

[49] M. Everingham, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman. The pascal visual object classes (voc) challenge. *IJCV*, 88(2):303–338, 2010. 2.2, 6.3.3

[50] A. Faktor and M. Irani. "clustering by composition"–unsupervised discovery of image categories. In *ECCV*. 2012. 2.3.2, 5.1

[51] L. Fei-Fei, R. Fergus, and P. Perona. Learning generative visual models from few training examples: An incremental bayesian approach tested on 101 object categories. *CVIU*, 2007. 2.2, 5.2

[52] P. F. Felzenszwalb and D. P. Huttenlocher. Pictorial structures for object recognition. *IJCV*, 61(1):55–79, 2005. 2.2

[53] P. Felzenszwalb, R. Girshick, D. McAllester, and D. Ramanan. Object detection with discriminatively trained part-based models. *PAMI*, 32(9):1627–1645, 2010. 2.2, 6.3.2

[54] R. Fergus, Y. Weiss, and A. Torralba. Semi-supervised learning in gigantic image collections. In *NIPS*, 2009. 2.3.1

[55] R. Fergus, P. Perona, and A. Zisserman. Object class recognition by unsupervised scale-invariant learning. In *CVPR*, 2003. 5.1, 5.2

[56] M. A. Fischler and R. A. Elschlager. The representation and matching of pictorial structures. *IEEE Transactions on computers*, 1973. 2.2

[57] J. Fiss, A. Agarwala, and B. Curless. Candid portrait selection from video. *SIGGRAPH Asia*, 2011. 3.1

[58] P. Földiák. Learning invariance from transformation sequences. *Neural Computation*, 3(2):194–200, 1991. 2.3.2

[59] D. F. Fouhey, A. Gupta, and M. Hebert. Data-driven 3D primitives for single image understanding. In *ICCV*, 2013. 2.3.1, 4.1, 6.3.4

[60] K. Fukunaga and L. Hostetler. The estimation of the gradient of a density function, with applications in pattern recognition. *IEEE Transactions on Information Theory*, 21(1):32–40, 1975. 1, 4.1

[61] K. Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological cybernetics*, 1980. 2.2

[62] B. Fulkerson, A. Vedaldi, and S. Soatto. Localizing objects with smart dictionaries. In *ECCV*, 2008. 3.4

[63] L. A. Gatys, A. S. Ecker, and M. Bethge. A neural algorithm of artistic style. *arXiv preprint arXiv:1508.06576*, 2015. 7.2

[64] B. Georgescu, I. Shimshoni, and P. Meer. Mean shift based clustering in high dimensions: A texture classification example. In *CVPR*, 2003. 4.2

[65] R. Girshick. Fast R-CNN. In *ICCV*, 2015. 6.3.3

[66] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *CVPR*, 2014. 1.1, 1.2, 2.3, 6.3.2, 6.3.3, 6.3.3

[67] Y. Gong and S. Lazebnik. Iterative quantization: A procrustean approach to learning binary codes. In *CVPR*, 2011. 3.4

[68] Y. Gong, L. Wang, R. Guo, and S. Lazebnik. Multi-scale orderless pooling of deep convolutional activation features. In *ECCV*. 2014. 1.1, 2.3

[69] Y. Gong, L. Wang, M. Hodosh, J. Hockenmaier, and S. Lazebnik. Improving image-sentence embeddings using large weakly annotated photo collections. In *ECCV*. 2014. 2.3.1

[70] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In *NIPS*, 2014. 2.3.2, 7.1

[71] K. Grauman and T. Darrell. The pyramid match kernel: Discriminative classification with sets of image features. In *ICCV*, 2005. 2.2

[72] K. Grauman and T. Darrell. Unsupervised learning of categories from sets of partially matching image features. In *CVPR*, 2006. 2.3.2, 5.1, 6.3.5

[73] P. Gronat, M. Havlena, J. Sivic, and T. Pajdla. Building streetview datasets for place recognition and city reconstruction. Technical Report CTU–CMP–2011–16, Czech Tech Univ., 2011. 3.3

[74] C. G. Gross, C. Rocha-Miranda, and D. Bender. Visual properties of neurons in inferotemporal cortex of the macaque. *Journal of neurophysiology*, 35(1):96–111, 1972. 2.1

[75] A. Guzman-Arenas. Computer recognition of three-dimensional objects in a visual scene. 1968. 2.2

[76] B. Hariharan, J. Malik, and D. Ramanan. Discriminative decorrelation for clustering and classification. In *ECCV*, 2012. 4.5, 5.2

[77] H. K. Hartline. The receptive fields of optic nerve fibers. *American Journal of Physiology*, 130(4):690–699, 1940. 2.1

[78] J. Hays and A. Efros. Im2gps: estimating geographic information from a single image. In *CVPR*, 2008. 3.2, 3.3

[79] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *arXiv preprint arXiv:1512.03385*, 2015. 2.3

[80] K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *ICCV*, 2015. 1

[81] K. Heath, N. Gelfand, M. Ovsjanikov, M. Aanjaneya, and L. J. Guibas. Image webs: Computing and exploiting connectivity in image collections. In *CVPR*, 2010. 2.3.2

[82] M. Hejrati and D. Ramanan. Analyzing 3d objects in cluttered images. In *NIPS*, 2012. 5.4.3

[83] G. Hinton, S. Osindero, and Y.-W. Teh. A fast learning algorithm for deep belief nets. *Neural computation*, 18(7):1527–1554, 2006. 2.3.2

[84] G. E. Hinton. Connectionist learning procedures. *Artificial Intelligence*, 14:715–770, 1989. 2.3.2

[85] G. E. Hinton, P. Dayan, B. J. Frey, and R. M. Neal. The "wake-sleep" algorithm for unsupervised neural networks. *Proceedings of the IEEE*, 1995. 2.3.2, 5.1, 5.2

[86] B. K. Horn. Shape from shading: A method for obtaining the shape of a smooth opaque object from one view. 1970. 2.2

[87] D. H. Hubel and T. N. Wiesel. Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *The Journal of physiology*, 160(1):106–154, 1962. 2.1

[88] D. H. Hubel and T. N. Wiesel. Receptive fields and functional architecture in two nonstriate visual areas (18 and 19) of the cat. *Journal of neurophysiology*, 28(2):229–289, 1965. 2.2

[89] D. H. Hubel and T. N. Wiesel. Receptive fields and functional architecture of monkey striate cortex. *The Journal of physiology*, 195(1):215–243, 1968. 2.1

[90] D. H. Hubel and T. N. Wiesel. The period of susceptibility to the physiological effects of unilateral eye closure in kittens. *The Journal of physiology*, 206(2):419, 1970. 2.2

[91] D. Hume. *A Treatise of Human Nature*. 1740. 1.1

[92] J. Hurri and A. Hyvärinen. Simple-cell-like receptive fields maximize temporal coherence in natural video. *Neural Computation*, 15(3):663–691, 2003. 2.3.2

[93] S. J. Hwang and K. Grauman. Reading between the lines: Object localization using implicit cues from image tags. *PAMI*, 34(6):1145–1158, 2012. 2.3.1

[94] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015. 6.2.1

[95] H. Izadinia, B. C. Russell, A. Farhadi, M. D. Hoffman, and A. Hertzmann. Deep classifiers from image tags in the wild. In *Proceedings of the 2015 Workshop on Community-Organized Multimodal Mining: Opportunities for Novel Solutions*, pages 13–18, 2015. 2.3.1

[96] A. Jain, A. Gupta, M. Rodriguez, and L. Davis. Representing videos using mid-level discriminative patches. In *CVPR*, 2013. 2.3.1, 4.1, 5.1

[97] D. Jayaraman and K. Grauman. Learning image representations equivariant to ego-motion. *ICCV*. 2.3.1

[98] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. In *ACM-MM*, 2014. 6.2.1

[99] A. Joulin, F. Bach, and J. Ponce. Discriminative clustering for image co-segmentation. In *CVPR*, 2010. 2.3.1

[100] A. Joulin, K. Tang, and L. Fei-Fei. Efficient image and video co-localization with frank-wolfe algorithm. In *ECCV*. 2014. 2.3.1

[101] A. Joulin, L. van der Maaten, A. Jabri, and N. Vasilache. Learning visual features from large weakly supervised data. *arXiv preprint arXiv:1511.02251*, 2015. 2.3.1

[102] M. Juneja, A. Vedaldi, C. V. Jawahar, and A. Zisserman. Blocks that shout: Distinctive parts for scene classification. In *CVPR*, 2013. 2.3.1, 4.1, 4.5, 4.6, 4.1, 4.6, 5.1

[103] E. Kalogerakis, O. Vesselova, J. Hays, A. Efros, and A. Hertzmann. Image sequence geolocation with human travel priors. In *ICCV*, 2009. 3.2

[104] L. Karlinsky, M. Dinerstein, and S. Ullman. Unsupervised feature optimization (ufo): Simultaneous selection of multiple features with their detection parameters. In *CVPR*, 2009. 2.3.2

[105] N. Kholgade, T. Simon, A. Efros, and Y. Sheikh. 3d object manipulation in a single photograph using stock 3d models. *SIGGRAPH*, 2014. 7.2

[106] G. Kim, C. Faloutsos, and M. Hebert. Unsupervised modeling of object categories using link analysis techniques. In *CVPR*, 2008. 2.3.2, 5.1

[107] G. Kim, E. P. Xing, L. Fei-Fei, and T. Kanade. Distributed cosegmentation via submodular optimization on anisotropic diffusion. In *ICCV*, 2011. 2.3.1

[108] D. P. Kingma and M. Welling. Auto-encoding variational bayes. In *ICLR*, 2014. 2.3.2, 7.1

[109] J. Knopp, J. Sivic, and T. Pajdla. Avoiding confusing features in place recognition. In *ECCV*. 2010. 2.3.1, 3.2

[110] P. Krähenbühl, C. Doersch, J. Donahue, and T. Darrell. Data-dependent initializations of convolutional neural networks. In *ICLR*, 2016. 6.3.2, 6.3.3

[111] A. Krizhevsky, I. Sutskever, and G. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, 2012. 1.1, 2.3, 6.1, 6.3, 6.2

[112] S. Lad and D. Parikh. Interactively guiding semi-supervised clustering via attribute-based explanations. In *ECCV*, 2014. 2.3.1

[113] M. Lades, J. C. Vorbrüggen, J. Buhmann, J. Lange, C. V. d Malsburg, R. Wurtz, and W. Konen. Distortion invariant object recognition in the dynamic link architecture. *IEEE Transactions on Computers*, 42(3):300–311, 1993. 2.2

[114] E. H. Land, John, and J. Mccann. Lightness and retinex theory. *Journal of the Optical Society of America*, pages 1–11, 1971. 2.2

[115] H. Larochelle and I. Murray. The neural autoregressive distribution estimator. In *AISTATS*, 2011. 2.3.2, 5.2, 6.1

[116] Q. V. Le. Building high-level features using large scale unsupervised learning. In *ICASSP*, 2013. 2.3.2, 5.1, 6.1

[117] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989. 2.2, 2.3

[118] H. Lee, A. Battle, R. Raina, and A. Y. Ng. Efficient sparse coding algorithms. In *NIPS*, 2006. 2.3.2, 6.1

[119] Y. J. Lee, A. Efros, and M. Hebert. Style-aware mid-level representation for discovering visual connections in space and time. In *ICCV*, 2013. 2.3.1, 3.6

[120] Y. J. Lee and K. Grauman. Foreground focus: Unsupervised learning from partially matching images. *IJCV*, 85(2):143–166, 2009. 2.3.2, 5.1

[121] L. Li, H. Su, E. Xing, and L. Fei-Fei. Object bank: A high-level image representation for scene classification and semantic feature sparsification. In *NIPS*, 2010. 3.5.4

[122] L.-J. Li and L. Fei-Fei. Optimol: automatic online picture collection via incremental model learning. *IJCV*, 88(2):147–168, 2010. 2.3.1

[123] L.-J. Li, H. Su, E. P. Xing, and L. Fei-Fei. Object bank: A high-level image representation for scene classification and semantic feature sparsification. In *NIPS*, 2010. 4.1

[124] N. Li and J. J. DiCarlo. Unsupervised natural experience rapidly alters invariant object representation in visual cortex. *Science*, 12:1502–1507, 2008. 2.3.2

[125] Q. Li, J. Wu, and Z. Tu. Harvesting mid-level visual concepts from large-scale internet images. In *CVPR*, 2013. 2.3.1, 4.1, 4.1, 5.1

[126] X. Li, C. Wu, C. Zach, S. Lazebnik, and J.-M. Frahm. Modeling and recognition of landmark image collections using iconic scene graphs. In *ECCV*. 2008. 2.3.1, 3.2

[127] Y. Li, D. Crandall, and D. Huttenlocher. Landmark classification in large-scale image collections. In *ICCV*, 2009. 3.2, 3.3

[128] Y. Li, M. Paluri, J. M. Rehg, and P. Dollár. Unsupervised learning of edges. *arXiv preprint arXiv:1511.04166*, 2015. 2.3.2

[129] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick. Microsoft COCO: Common objects in context. In *ECCV*. 2014. 2.2

[130] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. *CVPR*, 2015. 1.1, 2.3, 6.3.3

[131] F. Loyer. *Paris nineteenth century : architecture and urbanism*. Abbeville Press, New York, 1st american edition, 1988. 3.2, 3.4.2

[132] A. Mahendran and A. Vedaldi. Understanding deep image representations by inverting them. In *CVPR*, 2015. 6.3.6, 7.2

[133] T. Malisiewicz and A. A. Efros. Recognition by association via learning per-exemplar distances. In *CVPR*, 2008. 4.2

[134] T. Malisiewicz and A. Efros. Beyond categories: The visual memex model for reasoning about object relationships. In *NIPS*, 2009. 2.3.2, 6.1

[135] D. Marr. *Vision: A Computational Investigation into the Human Representation and Processing of Visual Information*. Henry Holt and Co., Inc., New York, NY, USA, 1982. 2.2

[136] M. Mathieu, C. Couprie, and Y. LeCun. Deep multi-scale video prediction beyond mean square error. *arXiv preprint arXiv:1511.05440*, 2015. 7.1

[137] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. In *NIPS*, 2013. 1.2, 2.3.2, 6.1

[138] I. Misra, A. Shrivastava, and M. Hebert. Watch and learn: Semi-supervised learning for object detectors from video. In *CVPR*, 2015. 2.3.1

[139] I. Misra, C. L. Zitnick, and M. Hebert. Unsupervised learning using sequential verification for action recognition. *arXiv preprint arXiv:1603.08561*, 2016. 2.3.2

[140] I. Misra, C. L. Zitnick, M. Mitchell, and R. Girshick. Seeing through the human reporting bias: Visual classifiers from noisy human-centric labels. In *CVPR*. 2.3.1

[141] H. Mobahi, R. Collobert, and J. Weston. Deep learning from temporal coherence in video. In *ICML*, Montreal, 2009. 2.3.2

[142] F. Moosmann, B. Triggs, and F. Jurie. Fast discriminative visual codebooks using randomized clustering forests. In *NIPS*, 2007. 3.4

[143] A. Mordvintsev, C. Olah, and M. Tyka. Inceptionism: going deeper into neural networks. Technical report, Google Inc., Google Research Blog, bit.ly/1BkXP09, 2015. 6.3.6, 7.2

[144] P. Mueller, P. Wonka, S. Haegler, A. Ulmer, and L. Van Gool. Procedural modeling of buildings. *SIGGRAPH*, 2006. 3.2

[145] R. Munroe. xkcd, a webcomic of romance, sarcasm, math and language. *Creative Commons Attribution-Noncommercial*, 2014. 5.2

[146] H. Murase and S. K. Nayar. Visual learning and recognition of 3-d objects from appearance. *IJCV*, 14(1):5–24, 1995. 2.2

[147] B. Murray and H. Larochelle. A deep and tractable density estimator. ICML, 2014. 5.2

[148] S. A. Nene, S. K. Nayar, H. Murase, et al. Columbia object image library (COIL-20). Technical report, CUCS-005-96, 1996. 2.2

[149] M. H. Nguyen, L. Torresani, F. de la Torre, and C. Rother. Weakly supervised discriminative localization and classification: a joint learning process. In *ICCV*, 2009. 2.3.1

[150] M. Noroozi and P. Favaro. Unsupervised learning of visual representations by solving jigsaw puzzles. *arXiv preprint arXiv:1603.09246*, 2016. 2.3.2, 7.1

[151] D. Okanohara and J. Tsujii. A discriminative language model with pseudo-negative samples. In *ACL*, 2007. 1.2, 6.1, 6.1

[152] A. Oliva and A. Torralba. Building the gist of a scene: The role of global image features in recognition. *Progress in brain research*, 155:23–36, 2006. 3.2

[153] A. Oliva and A. Torralba. The role of context in object recognition. *Trends in cognitive sciences*, 11(12):520–527, 2007. 5.1

[154] B. A. Olshausen and D. J. Field. Emergence of simple-cell receptive field properties by learning a sparse code for natural images. *Nature*, 381(6583):607–609, 1996. 2.3.2, 5.1, 6.1

[155] A. v. d. Oord, N. Kalchbrenner, and K. Kavukcuoglu. Pixel recurrent neural networks. *arXiv preprint arXiv:1601.06759*, 2016. 2.3.2, 7.1

[156] K. Paik. *The Art of Ratatouille*. Chronicle Books, 2006. 3.1

[157] M. Pandey and S. Lazebnik. Scene recognition and weakly supervised object localization with deformable part-based models. In *ICCV*, 2011. 2.3.1, 4.1

[158] S. N. Parizi, J. G. Oberlin, and P. F. Felzenszwalb. Reconfigurable models for scene recognition. In *CVPR*, 2012. 4.1

[159] D. Pathak, P. Krahenbuhl, J. Donahue, T. Darrell, and A. A. Efros. Context encoders: Feature learning by inpainting. In *CVPR*, 2016. 2.3.2, 7.1

[160] N. Payet and S. Todorovic. From a set of shapes to object discovery. In *ECCV*. 2010. 2.3.2, 5.1

[161] F. Perronnin, J. Sánchez, and T. Mensink. Improving the fisher kernel for large-scale image classification. In *ECCV*. 2010. 2.2

[162] L. Pinto and A. Gupta. Supersizing self-supervision: Learning to grasp from 50k tries and 700 robot hours. *arXiv preprint arXiv:1509.06825*, 2015. 2.3.1

[163] T. Quack, B. Leibe, and L. Van Gool. World-scale mining of objects and events from community photo collections. In *CIVR*, 2008. 2.3.1, 2.3.2, 6.3.5

[164] A. Quattoni and A. Torralba. Recognizing indoor scenes. In *CVPR*, 2009. 4.6, 4.1

[165] A. Radford, L. Metz, and S. Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015. 2.3.2, 7.1

[166] M. Radovanović, A. Nanopoulos, and M. Ivanović. Nearest neighbors in high-dimensional data: The emergence and influence of hubs. In *ICML*, 2009. 4.2

[167] A. Razavian, H. Azizpour, J. Sullivan, and S. Carlsson. Cnn features off-the-shelf: an astounding baseline for recognition. In *CVPR Workshops*, 2014. 2.3

[168] K. Rematas, B. Fernando, F. Dellaert, and T. Tuytelaars. Dataset fingerprints: Exploring image collections through data mining. In *CVPR*, 2015. 2.3.1

[169] K. Rematas, B. Fernando, F. Dellaert, and T. Tuytelaars. Dataset fingerprints: Exploring image collections through data mining. In *CVPR*, 2015. 2.3.2, 6.3.5

[170] D. J. Rezende, S. Mohamed, and D. Wierstra. Stochastic backpropagation and approximate inference in deep generative models. *ICML*, 2014. 2.3.2, 7.1

[171] L. G. Roberts. *Machine perception of three-dimensional solids*. PhD thesis, Massachusetts Institute of Technology, 1963. 2.2

[172] F. Rosenblatt. Principles of neurodynamics. 1962. 2.2

[173] H. A. Rowley, S. Baluja, and T. Kanade. Neural network-based face detection. *PAMI*, 20(1):23–38, 1998. 2.2

[174] B. C. Roy, M. C. Frank, P. DeCamp, M. Miller, and D. Roy. Predicting the birth of a spoken word. *Proceedings of the National Academy of Sciences*, 112(41):12663–12668, 2015. 1.2

[175] M. Rubinstein, A. Joulin, J. Kopf, and C. Liu. Unsupervised joint object discovery and segmentation in internet images. In *CVPR*, 2013. 2.3.1, 5.1

[176] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. Technical report, DTIC Document, 1985. 2.2

[177] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, et al. Imagenet large scale visual recognition challenge. *IJCV*, 115(3):211–252, 2015. 2.2

[178] B. C. Russell, W. T. Freeman, A. A. Efros, J. Sivic, and A. Zisserman. Using multiple segmentations to discover objects and their extent in image collections. In *CVPR*, 2006. 2.3.2, 5.1, 6.3.5

[179] B. C. Russell, A. Torralba, K. P. Murphy, and W. T. Freeman. Labelme: a database and web-based tool for image annotation. *IJCV*, 77(1-3):157–173, 2008. 2.2

[180] F. Sadeghi and M. F. Tappen. Latent pyramidal regions for recognizing scenes. In *ECCV*. 2012. 4.1

[181] R. Salakhutdinov and G. E. Hinton. Deep boltzmann machines. In *ICAIS*, 2009. 2.3.2

[182] G. Schindler, M. Brown, and R. Szeliski. City-scale location recognition. In *CVPR*, 2007. 2.3.1, 3.2

[183] F. Schroff, A. Criminisi, and A. Zisserman. Harvesting image databases from the web. *PAMI*, 33(4):754–766, 2011. 2.3.1

[184] Z. Shi, T. M. Hospedales, and T. Xiang. Bayesian joint topic modelling for weakly supervised object localisation. In *ICCV*, 2013. 2.3.1

[185] J. Shotton, M. Johnson, and R. Cipolla. Semantic texton forests for image categorization and segmentation. In *CVPR*, 2008. 3.4

[186] A. Shrivastava, T. Malisiewicz, A. Gupta, and A. A. Efros. Data-driven visual similarity for cross-domain image matching. *SIGGRAPH Asia*, 2011. 3.4.1

[187] A. Shrivastava, S. Singh, and A. Gupta. Constrained semi-supervised learning using attributes and comparative attributes. In *ECCV*. 2012. 2.3.1

[188] I. Simon, N. Snavely, and S. M. Seitz. Scene summarization for online image collections. In *ICCV*, 2007. 3.1

[189] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, 2014. 2.3, 6.3.3

[190] S. Singh, A. Gupta, and A. A. Efros. Unsupervised discovery of mid-level discriminative patches. In *ECCV*, 2012. 1.2, 2.3.1, 3.4.1, 4.1, 4.2, 4.5, 4.1, 4.6, 5.1, 5.2, 5.4.1, 6.3.5, 6.3.5

[191] P. Siva, C. Russell, and T. Xiang. In defence of negative mining for annotating weakly labelled data. In *ECCV*. 2012. 2.3.1

[192] J. Sivic, B. C. Russell, A. A. Efros, A. Zisserman, and W. T. Freeman. Discovering objects and their location in images. In *ICCV*, 2005. 2.3.2, 5.1, 6.3.5

[193] J. Sivic and A. Zisserman. Video google: A text retrieval approach to object matching in videos. In *ICCV*, 2003. 2.2, 3.4

[194] H. O. Song, R. Girshick, S. Jegelka, J. Mairal, Z. Harchaoui, and T. Darrell. On learning to localize objects with minimal supervision. In *ICML*, 2014. 5.4.1

[195] E. B. Sudderth, A. Torralba, W. T. Freeman, and A. S. Willsky. Learning hierarchical models of scenes, objects, and parts. In *ICCV*, 2005. 2.3.2, 5.2

[196] M. Sugiyama, T. Suzuki, and T. Kanamori. Density ratio estimation: A comprehensive review. *RIMS Kokyuroku*, 2010. 4.1

[197] J. Sun and J. Ponce. Learning discriminative part detectors for image classification and cosegmentation. In *ICCV*, 2013. 2.3.1, 4.1, 4.1, 5.1

[198] A. Sutcliffe. *Paris: an architectural history*. Yale University Press, 1996. 3.2

[199] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *CVPR*, 2015. 2.3

[200] K. Tang, A. Joulin, L.-J. Li, and L. Fei-Fei. Co-localization in real-world images. In *CVPR*, 2014. 2.3.1

[201] O. Teboul, L. Simon, P. Koutsourakis, and N. Paragios. Segmentation of building facades using procedural shape priors. In *CVPR*, 2010. 3.2

[202] L. Theis and M. Bethge. Generative image modeling using spatial lstms. In *NIPS*, 2015. 2.3.2, 5.2, 6.1

[203] L. Theis, R. Hosseini, M. Bethge, and C. K. Hsiao. Mixtures of conditional gaussian scale mixtures applied to multiscale image representations. *PloS one*, 7(7):e39857, 2012. 5.2

[204] B. Thomee, D. A. Shamma, G. Friedland, B. Elizalde, K. Ni, D. Poland, D. Borth, and L.-J. Li. The new data and new challenges in multimedia research. *arXiv preprint arXiv:1503.01817*, 2015. 6.3.3

[205] A. Torralba and A. A. Efros. Unbiased look at dataset bias. In *CVPR*, 2011. 5.1, 6.3.3, 6.3.5

[206] A. Torralba and A. Oliva. Statistics of natural image categories. *Network: Computation in Neural Systems*, 2003. 3.5.3

[207] M. Turk and A. Pentland. Eigenfaces for recognition. *Journal of cognitive neuroscience*, 3(1):71–86, 1991. 2.2

[208] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol. Extracting and composing robust features with denoising autoencoders. In *ICML*, 2008. 1.2, 2.3.2, 6.1

[209] H. von Helmholtz and J. P. C. Southall. *Treatise on Physiological Optics: Translated from the 3rd German Ed*. Optical Society of America, 1925. 2.1

[210] C. Vondrick, A. Khosla, T. Malisiewicz, and A. Torralba. HOG-gles: Visualizing object detection features. In *ICCV*, 2013. 5.5

[211] N. J. Wade and S. Finger. The eye as an optical instrument: from camera obscura to helmholtz's perspective. *Perception*, 30(10):1157–1177, 2001. 2.1

[212] J. Walker, A. Gupta, and M. Hebert. Patch to the future: Unsupervised visual prediction. In *CVPR*, 2014. 2.3.1, 7.1

[213] J. Walker, A. Gupta, and M. Hebert. Dense optical flow prediction from a static image. In *ICCV*, 2015. 7.1

[214] G. Wallis and H. H. Bülthoff. Effects of temporal association on recognition memory. *Proceedings of the National Academy of Sciences*, 98:4800–4804, 2001. 2.3.2

[215] G. Wallis and E. T. Rolls. Invariant face and object recognition in the visual system. *Progress in Neurobiology*, 51:167–194, 1997. 2.3.2

[216] C. Wang, W. Ren, K. Huang, and T. Tan. Weakly supervised object localization with latent category learning. In *ECCV*. 2014. 2.3.1

[217] G. Wang, Y. Zhang, and L. Fei-Fei. Using dependent regions for object categorization in a generative framework. In *CVPR*, 2006. 5.2

[218] L. Wang, Y. Qiao, and X. Tang. Motionlets: Mid-level 3d parts for human motion recognition. In *CVPR*, 2013. 2.3.1

[219] X. Wang, D. Fouhey, and A. Gupta. Designing deep networks for surface normal estimation. In *CVPR*, 2015. 1.1, 2.3, 7.2

[220] X. Wang and A. Gupta. Unsupervised learning of visual representations using videos. In *ICCV*, 2015. 2.3.2, 6.3.3, 6.3.4

[221] X. Wang, M. Yang, S. Zhu, and Y. Lin. Regionlets for generic object detection. In *ICCV*, 2013. 6.3.2

[222] X. Wang, B. Wang, X. Bai, W. Liu, and Z. Tu. Max-margin multiple-instance dictionary learning. In *ICML*, 2013. 2.3.1, 4.1, 4.1, 5.1

[223] M. Weber, M. Welling, and P. Perona. Unsupervised learning of models for recognition. In *CVPR*, 2000. 2.3.2, 5.1, 5.2

[224] J. Winn, A. Criminisi, and T. Minka. Object categorization by learned universal visual dictionary. In *ICCV*, 2005. 2.2

[225] L. Wiskott and T. J. Sejnowski. Slow feature analysis:unsupervised learning of invariances. *Neural Computation*, 14:715–770, 2002. 2.3.2

[226] J. Wu and J. M. Rehg. Centrist: A visual descriptor for scene categorization. *PAMI*, 33(8):1489–1501, 2011. 4.1

[227] J. Xiao, J. Hays, K. A. Ehinger, A. Oliva, and A. Torralba. Sun database: Large-scale scene recognition from abbey to zoo. In *CVPR*, 2010. 2.2

[228] T. Xiao, T. Xia, Y. Yang, C. Huang, and X. Wang. Learning from massive noisy labeled data for image classification. In *CVPR*, 2015. 2.3.1

[229] L. Xu, J. Neufeld, B. Larson, and D. Schuurmans. Maximum margin clustering. In *NIPS*, 2004. 4.2

[230] A. L. Yuille. Deformable templates for face recognition. *Journal of Cognitive Neuroscience*, 3(1):59–70, 1991. 2.2

[231] M. D. Zeiler and R. Fergus. Visualizing and understanding convolutional networks. In *ECCV*. 2014. 6.3.6, 7.2

[232] R. Zhao, W. Ouyang, and X. Wang. Learning mid-level filters for person re-identification. In *CVPR*, 2014. 2.3.1

[233] Y.-T. Zheng, M. Zhao, Y. Song, H. Adam, U. Buddemeier, A. Bissacco, F. Brucher, T.-S. Chua, and H. Neven. Tour the world: building a web-scale landmark recognition engine. In *CVPR*, 2009. 3.2

[234] G. Zhu, S. Yan, and Y. Ma. Image tag refinement towards low-rank, content-tag prior and error sparsity. In *ACM-MM*, 2010. 2.3.1

[235] J. Zhu, L.-J. Li, L. Fei-Fei, and E. P. Xing. Large margin learning of upstream scene understanding models. *NIPS*, 2010. 4.1

[236] W. Zou, S. Zhu, K. Yu, and A. Y. Ng. Deep learning of invariant features via simulated fixations in video. In *NIPS*, 2012. 2.3.2