THE UNIVERSITY
of EDINBURGH

ADVANCES IN DETECTING OBJECT CLASSES AND THEIR SEMANTIC PARTS

Davide Modolo

Doctor of Philosophy

Institute of Perception, Action and Behaviour

School of Informatics

University of Edinburgh

2017

Abstract

Object classes are central to computer vision and have been the focus of substantial research in the last fifteen years. This thesis addresses the tasks of localizing entire objects in images (*object class detection*) and localizing their semantic parts (*part detection*). We present four contributions, two for each task. The first two improve existing object class detection techniques by using context and calibration. The other two contributions explore semantic part detection in weakly-supervised settings.

First, the thesis presents a technique for predicting properties of objects in an image based on its global appearance only. We demonstrate the method by predicting three properties: aspect of appearance, location in the image and class membership. Overall, the technique makes multi-component object detectors faster and improves their performance.

The second contribution is a method for calibrating the popular Ensemble of Exemplar-SVM object detector. Unlike the standard approach, which calibrates each Exemplar-SVM independently, our technique optimizes their joint performance as an ensemble. We devise an efficient optimization algorithm to find the global optimal solution of the calibration problem. This leads to better object detection performance compared to using independent calibration.

The third innovation is a technique to train part-based model of object classes using data sourced from the web. We learn rich models incrementally. Our models encompass the appearance of parts and their spatial arrangement on the object, specific to each viewpoint. Importantly, it does not require any part location annotation, which is one of the main limits to training many part detectors.

Finally, the last contribution is a study on whether semantic object parts emerge in Convolutional Neural Networks trained for higher-level tasks, such as image classification. While previous efforts studied this matter by visual inspection only, we perform an extensive quantitative analysis based on ground-truth part location annotations. This provides a more conclusive answer to the question.

Acknowledgements

First and foremost I extend my gratitude to my supervisor Vittorio Ferrari. Vitto shaped my perspective as a researcher in so many fundamental ways by guiding me through difficult projects and constantly challenging my ideas. Your passion and determination is truly admirable.

Next, I would like to thank all my research mentors over the years: Joost van de Weijer, Cees G.M. Snoek, Albert Salah and Theo Gevers during my undergraduate years and Vittorio Ferrari, Chris Williams, Subramanian Ramamoorthy and Alexander Vezhnevets during my PhD years. They have provided invaluable advice and helped me grow as a researcher. A special thanks goes to Joost, for introducing me to computer vision.

A big thanks goes to all my co-authors whose work is featured in this thesis: Vittorio Ferrari, Alexander Vezhnevets, Olga Russakovsky and Abel Gonzalez-Garcia. It has been a real pleasure working together.

The PhD period is also a great opportunity for making friends and I have been extremely lucky in this respect. Thanks to the CALVIN vision group and all its rotating cast of participants for the stimulating discussions and the fun nights out: (sorted by joining date) Vittorio Ferrari, Anestis Papazoglou, Alexander Vezhnevets, Vicky Kalogeiton, Abel Gonzalez-Garcia, Dim P. Papadopoulos, Luca Del Pero, Jasper Uijlings, Holger Caesar, Michele Volpi, Paul Henderson, Miaojing Shi and Buyu Liu.

I would also like to thank my defence committee members Maurice Fallon and Mario Fritz, for taking time to examine my work and for your valuable comments. Moreover, thanks to the School of Informatics at the University of Edinburgh and to the ERC Starting Grant VisCul for supporting this research.

Finally, I would like to thank my parents for teaching me the value of hard work and for the sacrifices they made to ensure than I had an excellent education. Moreover, this thesis is, if indirectly, the fruit of my wife, Anu Modolo, who has helped me pursue my dreams and gave me confidence. A PhD can be challenging and frustrating, but Anu has always been a source of optimism and positive energy.

Declaration

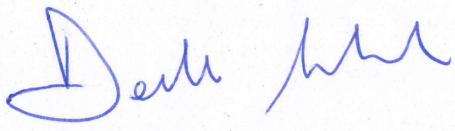
I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

Some of the material presented in this thesis is described in the following papers:

- Modolo Davide, Vezhnevets Alexander, and Ferrari Vittorio. *Context forest for object class detection*. In proceedings of the British Machine Vision Conference (BMVC), 2015 (oral presentaion).
- Modolo Davide, Vezhnevets Alexander, Russakovsky Olga and Ferrari Vittorio. *Joint calibration of Ensemble of Exemplar SVMs*. In proceedings of the Computer Vision and Pattern Recognition (CVPR) conference, 2015 (poster).
- Modolo Davide and Ferrari Vittorio. *Learning semantic part-based models from Google Images*. arXiv:1609.03140 preprint, 2016. Submitted to the IEEE Transactions on Pattern Analysis and Machine Intelligence journal (TPAMI).
- Gonzalez-Garcia Abel, Modolo Davide and Ferrari Vittorio. *Do semantic parts emerge in Convolutional Neural Networks?*. arXiv:1607.03738 preprint, 2016. Submitted to the International Journal of Computer Vision (IJCV).

Authorship of the last contribution is shared with fellow PhD student Abel Gonzalez-Garcia. Abel and I worked full-time on this project and contributed substantially to it (60-40). In the thesis we provide details of how the workload was divided.

(*Davide Modolo*)



To my beautiful wife.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

With the rapid adoption of digital cameras and mobile phone cameras, recognition and detection of objects, actions and scenes in images and videos have become important research topics.Visual recognition is one of the most simple and impressive abilities that humans possess. With little effort, humans are able to tell the identity or the category of an object despite its appearance variation. Furthermore, when observing a set of objects, humans can easily generalize and recognize objects that they have never seen before. For example, kids are able to generalize the concept of "cup" or "ball" after seeing just a few examples. However, while it may be obvious that humans are capable of recognizing objects and scenes under many variations in conditions, it is challenging to transfer the same range of capabilities to a vision system. In computer vision we define recognition as the process of mapping images (or sub-regions of images) to predefined semantic object classes. This thesis addresses two aspects of recognition: localizing whole objects in images (*object class detection*) and localizing their semantic parts (*part detection*).

The ability to detect objects and their semantic parts will be extremely useful in a variety of everyday applications that deal with images and videos. For example, object class detection will enable self-driving cars and reduce the number of accidents on the road, thanks to the reliable detection of pedestrians, other vehicles and traffic signs. Moreover, it will make visual surveillance systems more accurate by recognizing dangerous situations through the detection of objects like knifes and other weapons. Better detection will also improve image search engines on the web and it will enable organizing TV program repositories, automatic movie indexing, retrieval and summarization. Finally, it will help autonomous robots to gain accurate information about the environment, to work without the need for human assistance and to avoid situations that

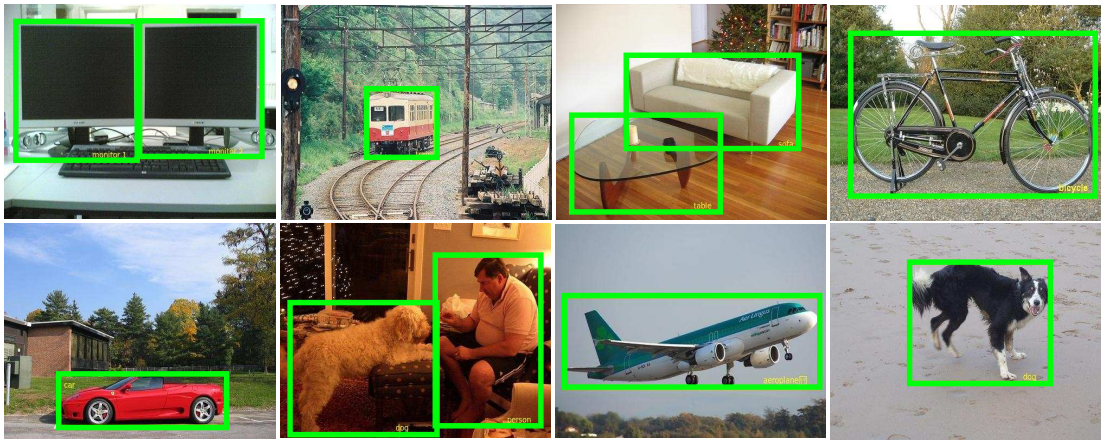


Figure 1.1: *Given an image, an object detector should predict a bounding-box around each instance of an object class. The figure presents predictions of object instances of the classes: (top) monitor, train, sofa and table, bicycle; (bottom) car, person and dog, airplane, dog.*

are harmful to people. Complementary to object class detection, semantic part detection will enable more fine-grained recognition that will improve all these applications even further. For example, in self-driving cars it will help understanding if the person detected in front of the car is crossing the street (hence slowing down the car is sufficient) or if the person is lying on the street (bringing the car to a full stop is necessary). Moreover, by predicting the different body parts of a person, it will enhance surveillance systems, search engines, etc. with the ability to answer fine-grained questions: *"is the arm of a person moving for punching or for hugging somebody"*, *"is the person running or walking"*? Finally, instead of moving around a detected table, a robot will be able to detect the table's legs and decide if the table is high enough for the it to pass underneath it, potentially achieving its goal faster.

This thesis presents two contributions on object class detection and two on semantic part detection. We define these tasks in sec. 1.1 and discuss some related work in sec. 1.2. We show how existing methods have been addressing them and we look at what challenges one has to face when designing a detection algorithm. Finally, sec. 1.3 summarises our contributions and sec. 1.4 closes the introduction by providing an overview of the content of the following chapters.

1.1 Problem definition

Object classes are central to computer vision and have been the focus of substantial research in the last fifteen years. This thesis addresses the tasks of localizing entire

(a) (b) (c) (d) (e)

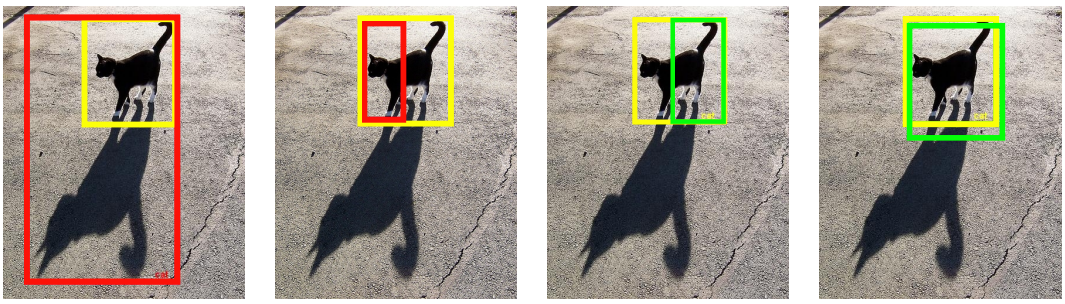
Figure 1.2: *Given an image, a part detector can predict the location of parts in terms of (a) keypoints (Wah et al., 2011), (b) ellipses (Fergus et al., 2003a), (c) segmentation masks (Chen et al., 2014) or (d-e) bounding-boxes (Felzenszwalb et al., 2010b; Zhang et al., 2014a).*

objects in images (object class detection) and localizing their parts (part detection).

Object class detection is the task of naming and localizing object instances in static images. Given an image, a detection algorithm should predict an axis-aligned rectangle (called *bounding-box*) around each instance of an object class. Fig. 1.1 presents some examples. The task is defined by an "image-class" pair: given the object class name, a detector has to find all the instances of this specific class.

Analogously to object detection, *part detection* is the task of naming and localizing parts of objects. Part detection can be solved at different levels of detail (fig. 1.2). In this thesis we address the task of localizing parts in terms of bounding-boxes (fig. 1.2d-e), as for object class detection. In this context, some works consider parts as arbitrary patches that are discriminative for an object class (fig. 1.2d), while others consider semantic parts (fig. 1.2e), which are object regions interpretable by humans (e.g., head, torso). In this thesis we focus on the latter.

Evaluation. Perfectly enclosing objects and parts instances in bounding-boxes is inherently difficult, even for humans. For evaluation purposes, in computer vision this requirement has been relaxed to the prediction of a "good enough" bounding-box around an object/part instance. Everingham et al. (2007) proposed to standardize this by using a metric called *intersection-over-union* (IoU). According to this metric, a predicted bounding-box (the detection) is considered "correct" if its intersection with a ground truth bounding-box (the real bounding-box) divided by their union is greater than a threshold (usually set to 0.5). This enabled comparison across different detectors and it is still considered as the reference metric. Fig. 1.3 presents some visual examples. The bounding-boxes in fig. 1.3a-b have IoU with the ground truth smaller than 0.5 and they are considered wrong, even if they actually contain the cat. On the other hand,



(a) IoU of 0.32 *(b) IoU of 0.45* (c) *IoU of 0.51* (d) *IoU of 0.85*

Figure 1.3: *A predicted bounding-box is considered correct if its IoU with the ground truth bounding-box (shown in yellow) is greater than 0.5. (a) and (b) present two detections not satisfying this condition, while (c) and (d) present two that do.*

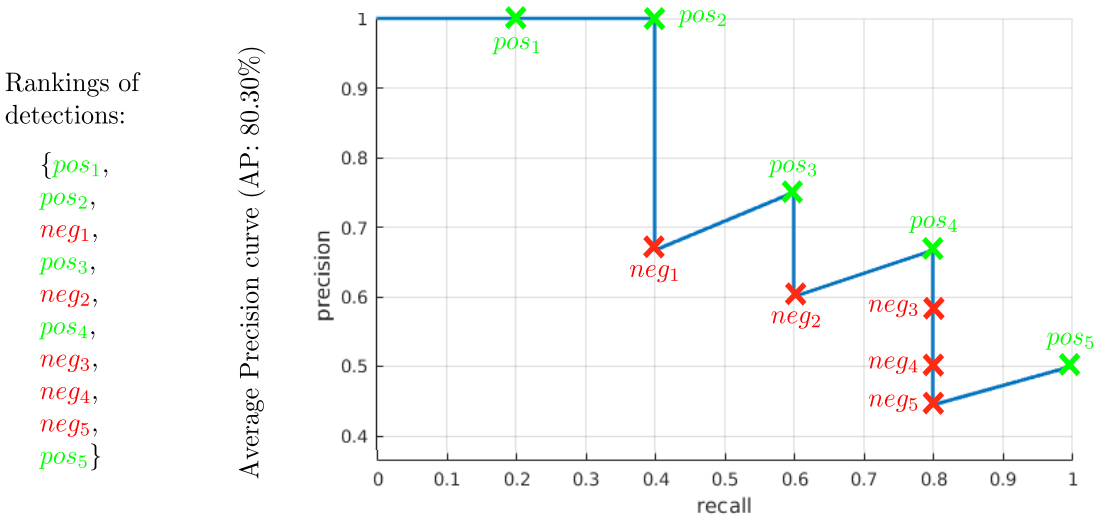


Figure 1.4: *Example of precision/recall curve and average precision. These are computed from a ranked list of datapoints and they are used in object class detection to evaluate the performance of different models.*

fig. 1.3c-d has IoU greater than 0.5 and are considered correct.

Finally, a detection algorithm is usually applied to a set of previously unseen images (the *test set*). The detector scores all possible locations (windows) in all images, based on how likely these are to contain the object the detector is trained to find. During evaluation, these windows are ranked according to their scores and the performance of the detector is assessed in terms of a quantitative measure called *average precision* (AP). This is computed in terms of a precision/recall curve (fig. 1.4). Given a list of ranked detections, the curve is computed at every point of the rankings. At each point, *Precision* is the proportion of correct detections. For example, in fig. 1.4, precision at pos_3 is 0.75, as 3 of the current 4 detections are correct. *Recall* instead is the pro-

portion of all positive object instances ranked above the given point. For example, in fig. 1.4, recall at both pos_3 and neg_2 is 0.6, as 3 of the 5 positive samples are in the current rankings. The AP value summarises the shape of the curve and it is defined as the mean precision at every recall point of the curve. In practice, for object class detection this is approximated and computed only at a set of eleven equally spaced recall levels (Everingham et al., 2007). Finally, average precision evaluates the performance on a single object class. On datasets containing several classes, *mean Average Precision* (mAP) is computed. This is the mean of the AP values of the different object classes.

1.2 State of object and semantic part detection

1.2.1 Performance evolution of object class detectors

Object class detection has been one of the central problems in computer vision for the last fifteen years (Agarwal et al., 2004; Leibe et al., 2004; Berg et al., 2005; Dalal and Triggs, 2005a; Sivic et al., 2005; Opelt et al., 2006; Gall and Lempitsky, 2009; Maji and Malik, 2009; Harzallah et al., 2009; Felzenszwalb et al., 2010b; Ferrari et al., 2010; Girshick et al., 2011; Malisiewicz et al., 2011; Van de Sande et al., 2011; Divvala et al., 2012; Hariharan et al., 2012; Chen et al., 2013a; Cinbis et al., 2013; Girshick et al., 2014; Sermanet et al., 2014; He et al., 2014; Szegedy et al., 2015; Redmon et al., 2016; Gidaris and Komodakis, 2015; Shrivastava et al., 2016). We review some of the most important object class detectors in chapter 2. Here instead we look at the evolution of object class detection performance.

For years, the PASCAL VOC dataset (Everingham et al., 2007) has been the standard way to train and evaluate object class detectors. The dataset, starting from 2007, contains 20 object classes, with an average of 630 training object instances per class. PASCAL VOC 2007 also offers a test set of 4952 images and thanks to this, it is possible to evaluate how detection performance (AP) has improved over the last few years (fig. 1.5). The figure reports mAP, which is the mean of the AP scores of the 20 object classes in the dataset. These results are obtained by running different methods on exactly the same (unseen) test set. From 2007 to 2013 the best performing algorithm achieved an mAP of: 17.1 in 2007 (Everingham et al., 2007), 22.9 in 2008 (Everingham et al., 2008), 29.0 in 2009 (Felzenszwalb et al., 2010b), 29.6 in 2010 (Zhu et al., 2010), 34.3 in 2011 (Zhang et al., 2011), 35.4 in 2012 (Girshick et al., 2012) and finally 41.7 in 2013 (Wang et al., 2013b). The real jump in performance, however, happened

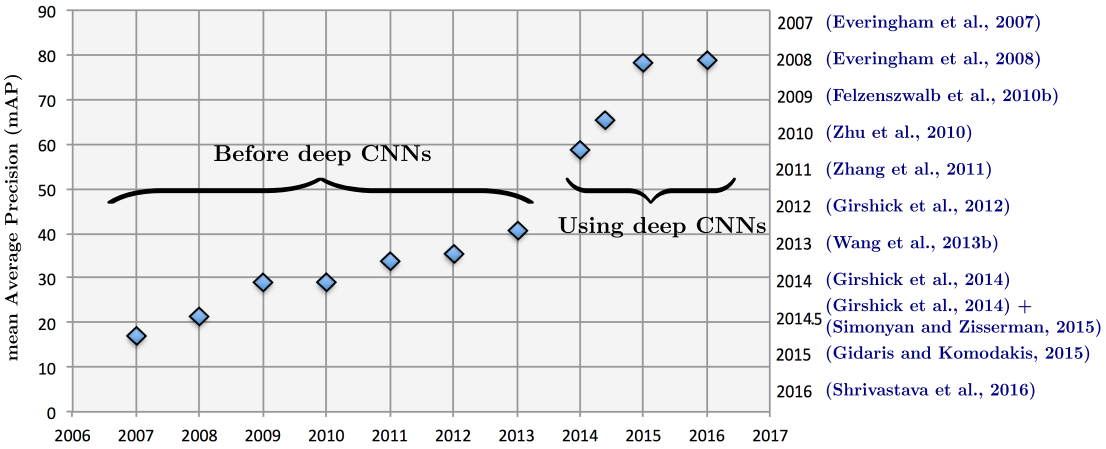


Figure 1.5: *Ten years of object detection performance on PASCAL 2007. The reported mAP is the mean Average Precision over the 20 object classes in the dataset. For each year we report the best performing object class detector.*

in the last three years. This was made possible by the successful application of deep Convolutional Neural Networks (CNN). CNN were heavily used in the 1990s (LeCun et al., 1998), but they fell out of common use in computer vision with the rise of support vector machines. In 2012, however, Krizhevsky et al. (2012) presented a way to train a CNN to achieve impressive image classification accuracy, substantially higher than any previous method. Their success resulted from training a deep CNN on 1.2 million labelled images (Russakovsky et al., 2015a). This rekindled interest in CNNs, which resulted in an increase in interest in CNN-based representations, leading to higher object detection performance. For example, Girshick et al. (2014) achieved an mAP of 58.5 in 2014 and of 65.4 few months later (using the VGG-Net of Simonyan and Zisserman (2015)). Moreover, Gidaris and Komodakis (2015) achieved an mAP of 78.2 in 2015 and, finally, Shrivastava et al. (2016) achieved an mAP of 78.9 in 2016. Similar to (Krizhevsky et al., 2012), these CNN-based methods require lots of data for training. As PASCAL VOC 2007 is too small, they all use additional external data for pre-training the network (Girshick et al., 2014) and sometimes even for training the detectors (Gidaris and Komodakis, 2015; Shrivastava et al., 2016). The approach of (Wang et al., 2013b) remains one of the best performing methods when training purely on the training set of PASCAL VOC 2007.

This remarkable improvement seems to suggest that object class detection is on the verge of being solved. This, unfortunately, is not yet the case and the arguments against are twofold. Firstly, PASCAL VOC 2007 is a very small dataset, both in terms of images and object classes. Object detectors evaluated on a much larger dataset, like

Figure 1.6: *Example of intra-class variation, where three instances of the same object class "car" have very different appearance and color. In the example, we have a red Formula 1 Ferrari, a black limousine and a blue neoclassical automobile.*

the recent 200-classes ImageNet Large Scale Visual Recognition Challenge (ILSVRC) dataset (Russakovsky et al., 2015a)[1], tend to achieve lower performance. For example, the model of Girshick et al. (2014) that achieves an mAP of 58.5 on PASCAL VOC 2007 (fig. 1.5, 2014), achieves a much lower mAP of 31.4 on ILSVRC 2013. Secondly, object detectors are evaluated using a very generous IoU threshold of 0.5. For example, the prediction of fig. 1.3c is considered correct according to this threshold, even though it fails to detect important features like the head and the front legs of the cat. Clearly, this is not acceptable for many real-world applications that rely on accurate detection of objects. Object detectors evaluated using a more conservative threshold tend to achieve lower performance. For example, Zhang et al. (2015) show that the mAP of 65.4 achieved by Girshick et al. (2014) with VGG-Net on PASCAL 2007 (fig. 1.5, 2014.5) drops to 35.2 with a stricter IoU of 0.7. This is a reduction in performance by almost half.

Some of the reasons why object detectors have not yet reached a desirable level of performance are because detectors are still severely limited by the presence of clutter, occlusion, lighting changes and a variety of other factors in the images. In the next section we present some of the major challenges of object class detection and what has been proposed so far to overcome them. Other reasons include dataset biases and training from incomplete, inconsistent or wrong annotations.

Finally, we note that newer versions of PASCAL VOC 2007 were released in 2010 (Everingham et al., 2010) and 2012 (Everingham et al., 2012), each time increasing the number of training images available. The 2012 version has an average of 1370 training instances per object class. Object class detection results on its test set are similar to the ones on PASCAL 2007 (Everingham et al., 2013; Girshick et al., 2014;

[1]The dataset has an average of 2770 training object instances per class and a test set of 40k images.

Figure 1.7: *Examples of illumination changes. The same coast scene affected by different illumination conditions.*

Shrivastava et al., 2016). Moreover, PASCAL VOC 2010 was recently augmented by Chen et al. (2014) (PASCAL-Part) with pixelwise semantic part annotations, making it one of the reference datasets for semantic part detection.

1.2.2 Challenges in object class detection

Some of the major difficulties negatively affecting the performance of object detectors are intra-class variation, illumination changes, occlusion and viewpoint changes (i.e., camera motion). Furthermore, with the recent increase in both the number of images available and the complexity of object detectors, the computational time (both for training and testing) has also become a very important point of discussion. In this section, we will briefly present all these challenges.

Intra-class variation. An object detector has to predict a bounding-box around each instance of an object class. Intra-class variation is a major problem while detecting rich object classes. For example, when considering the object class "car" (fig. 1.6), the detector has to be able to recognize any possible kind of car, from red to black, from a Formula 1 Ferrari to an older neoclassical automobile, and so on.

An object detector has to *learn* the rich variability of an object class from a training set of images containing the object. Local and global features are extracted from bounding-boxes and from negative windows (those not containing the object) (Dalal and Triggs, 2005a; Mikolajczyk et al., 2005; Felzenszwalb et al., 2010b; Malisiewicz, 2011; Girshick et al., 2014). An object model is then *trained* to retain only the most discriminative features for recognition so as to maximize detection performance.

Illumination changes. By changing the light, the color of parts of an object changes as well (fig. 1.7). There are two main types of illumination changes. (i) intensity change,

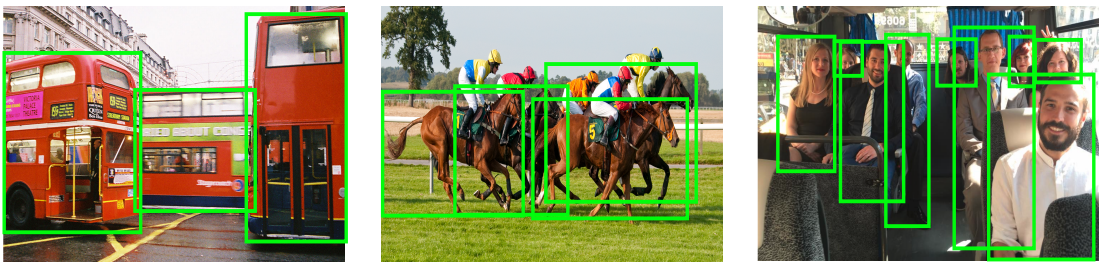


Figure 1.8: *Example of object instances occluded or truncated.*

where all the pixels of an image increase/decrease their values homogeneously. For example, if we raise the light intensity without changing anything else. (ii) direction change, where pixels change their values differently. For example, if the light source is fixed and we move the camera then some parts of an object will be more illuminated than before, and others less. This is one of the classic challenges of object detection. It is mostly alleviated by using color and illumination invariant features like properly normalized gradients (Lowe, 2004; Dalal and Triggs, 2005a; Bay et al., 2008) and contour segments (Shotton et al., 2005; Ferrari et al., 2008). Furthermore, some authors also proposed to use shape representations (Belongie et al., 2001; Felzenszwalb and Schwartz, 2007; Ferrari et al., 2010).

Occlusion. Occlusion is one of the major obstacles to robust object detection and only a few works have been proposed so far. An object's part can be occluded by other objects or an object can be truncated and partially lies outside the image area (fig. 1.8). The detector can therefore only partially observe the object. So far, sensitivity to partial occlusion has mostly been considered a lack in robustness and it has essentially been treated as noise rather than signal (Pepik et al., 2013). As a result, modelling has typically focused on preventing noisy (occluded) image evidence from affecting detection performance in a negative way. Among the most successful works, we can find integrated models of detection and segmentation using structured prediction and branch-and-bound (Gao et al., 2011), latent occlusion variables in max-margin framework (Vedaldi and Zisserman, 2009) and boosting (Wang et al., 2009b). Occlusion remains nowadays a very relevant problem and much work has to be done. One of the first few works that tried to treat occlusion as a signal rather than noise was presented by Vedaldi and Zisserman (2009). They explicitly accounted for truncation by developing a structured output model formulated as large margin learning with latent variables and slack rescaling. Furthermore, Pepik et al. (2013) recently proposed a

Figure 1.9: *Example of different viewpoints.*

method to include the occluder itself into the modelling, by mining distinctive, recurring occlusion patterns from the training data. By doing so they are able to detect highly occluded objects (e.g., lots of cars parked in a row, one in front of each other). Finally, Wang et al. (2013a) view occlusion as special contextual information and use a structured Hough voting method for detecting objects with heavy occlusion in indoor environments.

Viewpoint changes. Objects can be captured from different points of view. By moving a camera around an object, new parts of the object appear, as shown in fig. 1.9. Often these parts are very different in terms of visual appearance. For example, by looking at only the first image in fig. 1.9, one is not able to conclude that that object is a toy truck, or by looking only at the fourth image one is not able to infer that the back of the truck is flat. Viewpoint changes create a new axis of variability. If the change in viewpoint is minimal, local features partially invariant to translation and/or rotation (Lowe, 2004; Dalal and Triggs, 2005a; Bay et al., 2008; Krizhevsky et al., 2012) can overcome this issue. On the other hand, if the change is considerable, lots of images are needed to learn rich object representations. By having large training sets, one can train a single component model and hope that the classifier is good enough to learn the different modalities of an object class (Vedaldi et al., 2009; Girshick et al., 2014) or train a multi-component detector, where each component explicitly learns a different viewpoint of the object class (Felzenszwalb et al., 2010b; Malisiewicz, 2011). In the spirit of multi-component detectors, the EE-SVM object model (Malisiewicz, 2011) is an extreme case that trains one component per available training object instance. In chapter 4 of this thesis we present a technique to calibrate all the EE-SVM components jointly that improves the EE-SVM detection performance (Modolo et al., 2015b). Finally, we note that if the 3D models of an object class is available, one could augment a training set with synthetic viewpoints (Rematas et al., 2014). These novel views are generated automatically in an image-based rendering problem.

1.2.3 Runtime complexity of object class detectors

Object detection algorithms often rely on the sliding window paradigm (Viola and Jones, 2001, 2004; Dalal and Triggs, 2005a; Vedaldi et al., 2009; Felzenszwalb et al., 2010b; Malisiewicz, 2011; Cinbis et al., 2013; Girshick et al., 2014; Shrivastava et al., 2016), which scores a large number of windows at all positions and scales on an image using a classifier. This, however, leads to a complexity of $O(LC)$, where L is the number of locations (windows) that need to be evaluated and C is the number of classifiers evaluated on each window.

Reducing the number of classifier evaluations L. Some approaches try to reduce the number of times a complex classifier is evaluated. Lampert et al. (2008) propose a branch-and-bound scheme called *efficient subwindow search*. The algorithm finds the highest scored window while evaluating the classifier as few times as possible. However, it is restricted to classifiers for which a good upper bound on a set of windows exists. Alexe et al. (2012) propose a generic *objectness* measure that quantifies how likely it is for an image window to contain an object of any class. They use this measure to evaluate the classifier only on a small number of windows likely to cover objects rather than backgrounds. Inspired by this work, (Uijlings et al., 2013; Manen et al., 2013; Cheng et al., 2014; Zitnick and Dollár, 2014) present techniques that, given an image, propose a small subset of object window candidates. Then, they evaluate the classifier only on these windows. Among these, one of the most successful is the *selective search* method of Uijlings et al. (2013), which hierarchically and greedily groups similar segmented regions together and defines window proposals accordingly.

Reducing the cost of evaluating the classifier. Complementary approaches try to reduce the cost of evaluating a complex classifier on a window. Viola and Jones (2001) propose a method for combining classifiers in a *cascade*. They employ many classifiers of different complexity, which are run in sequence (low to high). Each classifier only evaluates the subset of windows selected by the previous classifier. In this way, complex non-linear kernels only evaluate few highly scored windows. Recently, this idea has been successfully applied to star-structured models (Felzenszwalb et al., 2010a,b; Pedersoli et al., 2011). Moreover, in the context of complex non-linear kernels, Maji et al. (2008) and Vedaldi and Zisserman (2010) present ways to approximate the expensive histogram intersection and χ^2-kernels into compact linear representations, leading

to faster evaluation. Finally, instead of proposing a fixed cascade of features/classifiers in a pre-defined ordering (Viola and Jones, 2001), (Karayev et al., 2012, 2014) proposes to learn what subsets of features are relevant at each step of the cascade. Decisions are made at test time and are based on the observed data and on the results of the previous levels of the cascade.

Reducing the complexity in the number of classes C. (Song et al., 2012; Dean et al., 2013) propose to reduce object detection complexity by targeting its dependence on the number of object classes. The framework of Song et al. (2012) reduces object detection complexity using sparse prototype representations that exploit the intrinsic redundancy among model filters. Their speed-up increases with the number of object classes considered, but it's performance decreases with it. The framework of (Dean et al., 2013) instead reduces object detection complexity from $O(LC)$ to $O(L)$ by exploiting locality-sensitive hashing (Indyk and Motwani, 1998) to approximate the dot product in the kernel operator of the convolution with a multi-band hash-table lookup. This enables them to determine which object classifier has the highest response in nearly constant time, independent of the number of object classes. However, as the number of objects increase, the prediction accuracy of their method also decreases.

In chapter 3 of this thesis we also present a technique to reduce runtime complexity of object class detectors (Modolo et al., 2015a). Our technique reduces complexity in two ways. First, it reduces the cost of evaluating multi-component object class detectors on an image by selecting a small subset of relevant components. And second, it reduces the complexity in the number of classes for *any* object class detector by predicting what object classes are likely to be present in an image. Finally, by selecting relevant detectors the technique also improves object class detection performance.

1.2.4 Semantic part detection

Semantic part detection is a relatively new task and has gained significant attention in the last few years. The key advantages of exploiting parts are that they have lower intra-class variability than whole objects, they deal better with pose variation and their configuration provides useful information about the aspect of the object. Differently from the standalone task of object class detection, semantic part detection has mostly been addressed in the context of other vision tasks.

The most notable example is the fine-grained recognition task, which heavily relies on accurate detection of semantic parts, like body parts of birds (Farrell et al., 2011; Wah et al., 2011; Zhang et al., 2012, 2013; Gavves et al., 2013; Liu and Belhumeur, 2013; Goering et al., 2014; Liu et al., 2014; Zhang et al., 2014a; Simon et al., 2014; Lin et al., 2015; Shih et al., 2015; Akata et al., 2016) and pets (Parkhi et al., 2011; Liu et al., 2012; Parkhi et al., 2012). In these works, an object is treated as a collection of parts which models its shape and appearance. Semantic parts help capturing subtle object appearance differences that could not be captured by a monolithic object model. These differences are crucial to discriminate between animal breeds.

Other applications where semantic part detection have been used are object class detection (Chen et al., 2014), object segmentation (Wang and Yuille, 2015), articulated human and animal pose estimation (Sun and Savarese, 2011; Ukita, 2012; Liu et al., 2014) and attribute prediction (Zhang et al., 2013; Vedaldi et al., 2014; Gkioxari et al., 2015). In class detection and segmentation, object parts help deal with deformable, occluded and low resolution objects. In articulated pose estimation, parts help identifying objects in special configurations (e.g., jumping and sitting) as opposed to canonical ones. Finally, in attribute prediction, attributes are best predicted by the part containing direct evidence about them.

Some of the above mentioned methods detect semantic parts in terms of simple keypoints by indicating the centre of a part (Wah et al., 2011; Liu et al., 2012; Gavves et al., 2013; Liu and Belhumeur, 2013; Goering et al., 2014; Liu et al., 2014). Others, instead, predict the whole extent of a part, either in terms of bounding-boxes (Farrell et al., 2011; Sun and Savarese, 2011; Parkhi et al., 2011, 2012; Zhang et al., 2012; Ukita, 2012; Zhang et al., 2013; Chen et al., 2014; Vedaldi et al., 2014; Zhang et al., 2014a; Simon et al., 2014; Lin et al., 2015; Shih et al., 2015; Gkioxari et al., 2015; Akata et al., 2016) or segmentation masks (Wang and Yuille, 2015).

Predicting the whole extent is a challenging task and requires accurate part location annotations for training. Annotations are however very expensive to obtain. Russakovsky et al. (2015a) show that it takes on average 42 seconds to obtain a single object bounding-box by crowd-sourcing on Amazon Mechanical Turk[2]. This time increases considerably when annotating semantic parts. First, parts are smaller and more difficult to annotate than objects and second, there are several semantic parts for a single object instance. This is probably the reason why part detection in terms of bounding-boxes has been addressed in the context of other tasks and not evaluated on

[2] www.mturk.com

its own.

One solution to bypass this time consuming process is to train detectors in weakly-supervised settings. Weakly-supervised object detection is already an active area of research (Deselaers et al., 2010; Siva and Xiang, 2011; Russakovsky et al., 2012; Song et al., 2014a,b; Bilen et al., 2014, 2015; Cinbis et al., 2015; Wang et al., 2015). Given a set of training images known to contain instances of a certain object class, these methods try to predict the location of these instances in the images and train an object class detector from them. Training semantic part detectors in weakly-supervised settings is instead much harder. Some works have trained non-semantic part models, where the parts are simply arbitrary patches that are discriminative for an object class (Crandall and Huttenlocher, 2006; Felzenszwalb et al., 2010b; Arbeláez et al., 2012; Endres et al., 2013; Juneja et al., 2013; Zhang et al., 2014b; Tsai et al., 2015), but to the best of our knowledge, no work has trained semantic part detectors in weakly-supervised settings. In chapters 5 and 6 of this thesis we explore semantic part detection in this context. In chapter 5 we train semantic part-based models of object classes from the web, while in chapter 6 we study whether semantic object parts emerge in Convolutional Neural Networks trained for higher level tasks, such as image classification and object detection.

1.3 Contributions

This thesis presents four main contributions. The first two improve existing object class detection techniques by using context and calibration. The other two try to bypass the issue of expensive manual part annotations by exploring semantic part detection in weakly-supervised settings. Each of these contributions is described in a different chapter. These are:

- **Chapter 3**. A technique for predicting properties of the objects in an image based on its global appearance, called Context Forest (ConF) (Modolo et al., 2015a). Compared to standard nearest-neighbour techniques, ConF is more accurate, faster and more memory efficient. We demonstrate ConF by predicting three properties: aspects of appearance, location in the image, and class membership. In extensive experiments we show that (i) ConF can automatically select

which components of a multi-component detector to run on a given test image, obtaining a considerable speed-up for detectors trained from large sets; (ii) ConF can improve object detection performance by removing false positive detections at unlikely locations and by (iii) removing false positives produced by classes unlikely to be present in the image.

- **Chapter 4**. A method for calibrating the popular Ensemble of Exemplar-SVMs object class detector (Modolo et al., 2015b). Unlike the standard approach, which calibrates each Exemplar-SVM independently, our method optimizes their joint performance as an ensemble. We formulate joint calibration as a constrained optimization problem and devise an efficient optimization algorithm to find its global optimum. The algorithm dynamically discards parts of the solution space that cannot contain the optimum early on, making the optimization computationally feasible. To scale the method to large datasets, we show how to relax global optimality and attain a good approximate solution. In extensive experiments we show that (i) our joint calibration procedure outperforms independent calibration on the task of classifying windows as belonging to an object class or not; and (ii) this improved window classifier leads to better performance on the object detection task.

- **Chapter 5**. A technique to train semantic part-based models of object classes from Google Images (Modolo and Ferrari, 2016). Our models encompass the appearance of parts and their spatial arrangement on the object, specific to each viewpoint. We learn these rich models by collecting training instances for both parts and objects, and automatically connecting the two levels. Our framework works incrementally, by learning from easy examples first, and then gradually adapting to harder ones. A key benefit of this approach is that it requires no manual part location annotations. In extensive experiments we show that (i) the performance increases at every step of the learning, with the final models more than doubling the performance of directly training from images retrieved by querying for part names and (ii) our part models can help improving object detection performance.

- **Chapter 6**. A study on whether Convolutional Neural Networks trained for higher-level tasks, such as image classification and object detection, learn se-

mantic parts in their internal representation (Gonzalez-Garcia et al., 2016)[3]. We investigate the responses of convolutional filters and try to associate their stimuli with semantic parts. While previous efforts studied this matter by casual visual inspection, we perform an extensive quantitative analysis based on ground-truth part bounding-boxes and human judgments. We explore different layers, network depths, and supervision levels. Even after assisting the filters with several mechanisms to favor this association, we find that only about 30% of the semantic parts in PASCAL-Part dataset emerge from a network trained for object class detection. Moreover, human judgment reveal that, on average per object class, 7% of the filters correspond to semantic parts and 13% to other systematic concepts. Finally, we investigate how discriminative semantic parts and network filters are for the objects the network is trained to predict. We discover that discriminative semantic parts tend to emerge more than non-discriminative ones, and that the discriminative power of the network can be attributed to a few discriminative filters specialized to each object class.

1.4 Thesis Outline

The structure of this thesis reflects the heterogeneous nature of its contributions and is organized as follows. First, in chapter 2 we present some popular object class detectors used throughout this thesis. Then, each of the following chapters covers one of the contributions, as indicated in the previous section. Since every chapter is dedicated to a relatively different problem, it directly includes related work of the particular field, a description of the method, a series of experiments we conducted, a discussion of the results we obtained and some future directions. Chapters are rather self-contained, so that anyone interested in just one of the works will be able to understand it. Finally, we draw some general conclusions in chapter 7.

[3] Authorship of the last contribution is shared with fellow PhD student Abel Gonzalez-Garzia. Abel and I worked full-time on this project and contributed substantially to it (60-40). In chapter 6 we provide details of how we divided the workload.

Chapter 2

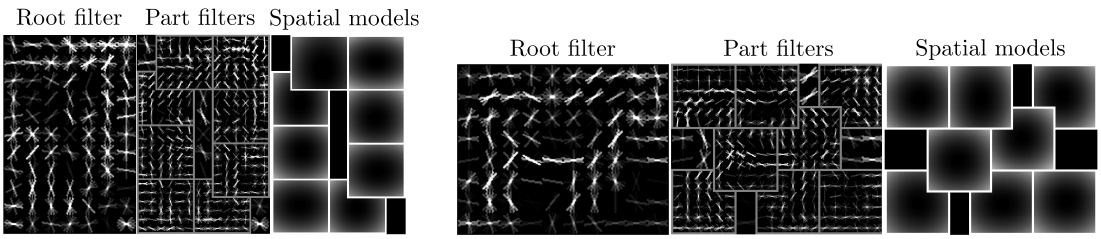
Object class detectors

In this chapter we introduce some of the most relevant object detectors. First, we describe three detectors used in the experiments presented in this thesis. These are the *Deformable part-based model* of Felzenszwalb et al. (2010b) (sec. 2.1), the *Ensemble of Exemplar-SVMs* of Malisiewicz (2011) (sec. 2.2) and the *Region-based Convolutional Neural Network* of Girshick et al. (2014) (sec. 2.3). Finally, we briefly present two detectors not used in this thesis, but historically relevant (sec. 2.4).

2.1 Deformable part-based model (DPM)

DPM was introduced by Felzenswalb et al. in (Felzenszwalb et al., 2008, 2010b). It is a fast sliding-window detector that can accommodate the rich intra-class variation of object classes and achieve good detection performance. From 2009 to 2013 it was one of the best object detectors available and nowadays it is still widely used.

DPM uses multi-scale deformable part models. It builds on the pictorial framework of Felzenszwalb and Huttenlocher (2005), which follows the structure of a star-graph, where there is one central part that covers the whole object (the *root*) and a set of parts connected to it. More specifically, DPM represents an object by a collection of parts arranged in a deformable configuration. The idea is that the parts capture local object appearance while the deformable configuration captures the relative position of each part with respect to the root (fig. 2.1). As an object class can exhibit a wide range of variation in appearance, a single deformable model is often not expressive enough. DPM instead trains a mixture of model components, each specialized to an aspect of the training data. Each component is trained on a subset of the training data with compact appearance, e.g., different viewpoints (Felzenszwalb et al., 2010b) or



(a) (b)

Figure 2.1: *Example of trained DPM model for object class horse. It is a mixture model with two components ((a) and (b)), each defined by a coarse root filter, eight higher resolution part filters, and a spatial model for the location of each part relative to the root. The filters specify weights for histogram of oriented gradients (HOG) features (Dalal and Triggs, 2005a). Their visualization shows the positive weights at different orientations. The visualization of the spatial models reflects the cost of placing the center of a part at different locations relative to the root.*

subclasses (Divvala et al., 2012).

The model is trained using a latent support vector machine classifier with hard-negative mining (Dalal and Triggs, 2005b). At each training iteration, "easy" examples are removed from the training set and hard examples are added. The definition of easy and hard examples is related to bootstrapping, and they are, respectively, examples that are correctly classified and examples that are wrongly classified, with high confidence. An example of a trained model for the object class *horse* is shown in fig. 2.1. It is a mixture model with two components, (a) and (b). To learn them, positive bounding-boxes are sorted by their aspect ratio and split into two groups. Two different root filters (linear SVMs) are then trained, one on each group. Aspect ratio is here used as an indicator of intra-class variation. Finally, for each root filter, the model learns a set of part filters (also linear SVMs) and deformation configurations. These parts are learned in weakly-supervised settings, without using any part annotation. For this reason they are recurring discriminative patches, rather than semantic parts. To learn a mixture model with m components the process is equivalent, but with the positive bounding-boxes split into m groups.

At test time, the score of one model component on a window is the weighted sum of the appearance score of the root, the appearance scores of the parts and the deformation costs. Each deformation cost penalizes a part for moving too far from its learnt ideal location relative to the root. Finally, the score of a mixture model is the maximum over the scores of its components.

A publicly available implementation of DPM is available at (Girshick et al., 2012). We use it in our experiments in chapter 3.

Improvements over the original work. Many works successfully extended the DPM object model. For example, (Felzenszwalb et al., 2010a; Pedersoli et al., 2011; Song et al., 2012; Zhu et al., 2014; Yan et al., 2014) developed new algorithms to speed up detection performance. Park et al. (2010), instead, extended DPM models to multi-resolution models that can achieve higher flexibility in terms of object representation; Ott and Everingham (2011) proposed to share object parts among multiple mixture models. This led to more compact models that allowed training examples to be shared by multiple components, ameliorating the performance on small training sets; Felzenszwalb and McAllester (2011) formulated DPMs in terms of grammars, so that they can model objects with a variable structure, differently from before where the structure of the object was fixed. Finally, Girshick and Malik (2013) decreased the training time of DPM models by using linear discriminant analysis (LDA).

Works that use DPM models. DPM is most commonly used for the detection of object classes (Felzenszwalb et al., 2010b,a; Park et al., 2010; Parkhi et al., 2011; Ott and Everingham, 2011; Felzenszwalb and McAllester, 2011; Divvala et al., 2012; Azizpour and Laptev, 2012; Chen et al., 2013a; Girshick and Malik, 2013; Chen et al., 2013b; Yan et al., 2014; Divvala et al., 2014; Zhu et al., 2014; Lin et al., 2014; Papadopoulos et al., 2014; Kalogeiton et al., 2016), pedestrians (Pedersoli et al., 2011) and humans (Drayer and Brox, 2014), Sometimes it is also used for other tasks, like fine-grained recognition (Zhang et al., 2013; Chai et al., 2013), multiple-person tracking (Shu et al., 2012), scene recognition (Pandey and Lazebnik, 2011) and viewpoint classification (Gu and Ren, 2010).

2.2 Ensemble of Exemplar-SVMs (EE-SVM)

(Malisiewicz et al., 2011) propose a simple, yet powerful, method to train a separate object classifier for every positive training window (*Exemplar*) in a training set. Each of these Exemplars (E-SVM) is a linear SVM represented by using a rigid HOG template and it is trained using the Exemplar as the only positive against millions of negative windows. Hard-negative mining is adopted during training (Dalal and Triggs, 2005b). An example of trained E-SVM models for the object category *horse* is shown

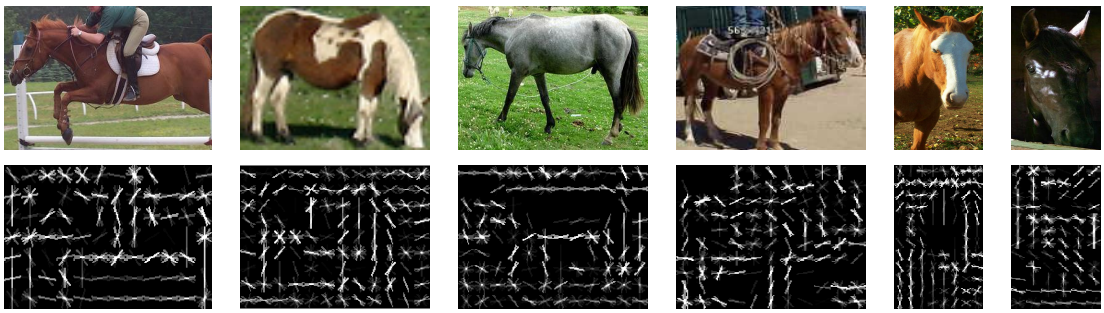


Figure 2.2: *Example of trained EE-SVM model for object category horse. The model is an ensemble of six Exemplar-SVMs represented using rigid HOG templates (bottom row) trained from six different positive training instances (top row). Note how the Exemplar-SVMs are visually similar to their corresponding positive instances. This property is the key to transferring annotations from Exemplars onto test windows.*

in fig. 2.2. The model is an ensemble of six Exemplars, each capturing one specific aspect of the object.

At test time, each window is scored by all E-SVMs, and the highest score is assigned to each window. Different Exemplars therefore compete for the same image region. Because of this max operation, it is necessary to calibrate the E-SVMs to make their scores comparable. To achieve this, EE-SVM calibrates each Exemplar independently by fitting a sigmoid (Platt, 1999) on its output. Finally, the detections are filtered by non-maximum suppression in a final stage.

A publicly available implementation of EE-SVM is available at (Malisiewicz, 2011) and we use it in our experiments in chapters 3 and 4.

Improvements over the original work. Training linear SVMs for each positive window can become very expensive. Hariharan et al. (2012) proposed to train Linear Discriminant Analysis (LDA) models instead of linear SVMs, which drastically reduces the training time, while achieving similar performance. Moreover, Aytar and Zisserman (2012) proposed a transfer learning approach to boost the E-SVMs performance using patches of parts from previously learned classifiers. Finally, Modolo et al. (2015b) (sec. 4 of this thesis) proposed a new way to calibrate all the E-SVMs jointly that achieves higher detection performance.

Works that use EE-SVM models. EE-SVM is widely used for object class detection applications (Shrivastava et al., 2011; Singh et al., 2012; Aytar and Zisserman, 2012; Endres et al., 2013; Dong et al., 2013; Tighe and Lazebnik, 2013; Juneja et al., 2013;

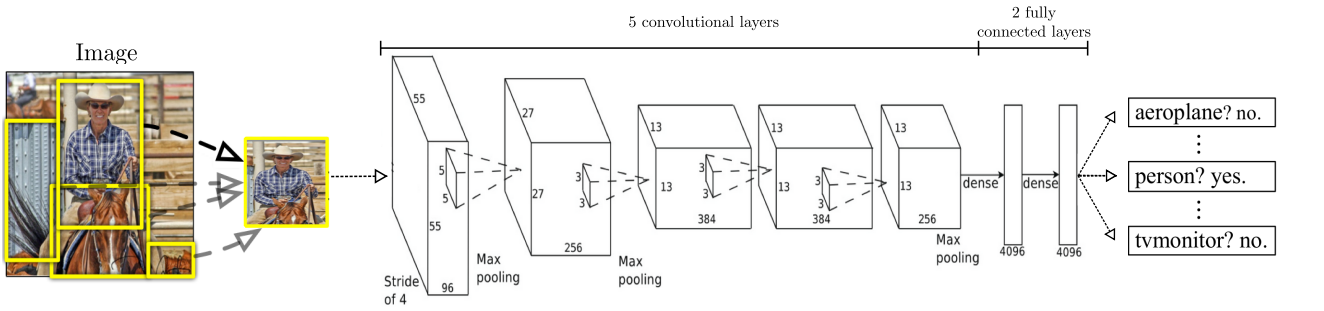
Gronat et al., 2013; Song and Xiao, 2014; Vezhnevets and Ferrari, 2014; Aubry et al., 2014a) and beyond, because it explicitly associates a training example to each object it detects in a test image. This enables transferring meta-data such as segmentation masks (Malisiewicz et al., 2011; Tighe and Lazebnik, 2013), 3D models (Malisiewicz et al., 2011), viewpoints (Aubry et al., 2014a), GPS locations (Gronat et al., 2013) and part-level regularization (Aytar and Zisserman, 2012). Furthermore, EE-SVM can also be used for discovering objects parts (Singh et al., 2012; Endres et al., 2013), scene classification (Singh et al., 2012; Juneja et al., 2013), object classification (Dong et al., 2013), image parsing (Tighe and Lazebnik, 2013), image matching (Shrivastava et al., 2011), automatic image annotation (Vezhnevets and Ferrari, 2014), 3D object detection (Song and Xiao, 2014) and as visual feature encoders (Vezhnevets and Ferrari, 2014; Zepeda and Perez, 2015).

2.3 Region-based Convolutional Neural Network (R-CNN)

A novel object detector achieving state-of-the-art results in 2014 was introduced by Girshick et al. (2014). R-CNN consists of three components (fig. 2.3). The first generates class-independent object proposals (Uijlings et al., 2013) (sec. 1.2.3) which define the set of available windows (fig. 2.3a). The second component is a convolutional neural network (CNN) that extracts a fixed-length feature vector from each of these windows (fig. 2.3b) and the third is a set of class-specific linear SVMs used to classify them (fig. 2.3c).

The novelty of this detector lies in its second component. R-CNN uses one of the most popular convolutional neural network architectures in computer vision, which is the CNN of Krizhevsky et al. (2012), winner of the ILSVRC 2012 image classification challenge (Russakovsky et al., 2015a). This has five convolutional layers and two fully connected ones (fig. 2.3b). Each convolutional layer takes the output of the previous layer as input, and produces its output by applying four operations: convolution, non-linearity, pooling, and normalization. The convolution operation slides a set of learned filters of different sizes and strides over the input. The nonlinearity of choice is the Rectified Linear Unit (ReLU) and it is applied right after the convolution. Finally, the fully connected layers take the output of the last convolutional layer and produce a fixed-length feature vector of 4096 dimensions.

At training time, R-CNN extracts features from all training windows and uses them to train class-specific linear SVMs. Clearly, it is important that the CNN learns features



a) Generate independent object proposals

b) Extract fixed-length feature vector (using a CNN)

c) Class-specific linear SVMs classifiers

Figure 2.3: *Illustration of the R-CNN object class detector. Given a test image, the model operates in three stages: (a) it generates independent window proposals, (b) it extracts a fixed-length feature vector from each window and (c) it classifies each window using class-specific linear SVMs. Note: the figure is adapted from fig. 1 of (Girshick et al., 2014) and fig. 2 of (Krizhevsky et al., 2012).*

that have sufficient representational power and generalizability. R-CNN achieves this by fine-tuning the image classification CNN of (Krizhevsky et al., 2012) for window classification. At training time, this CNN has an additional layer (after the last fully connected one) that outputs a class probability vector, which contains one score for each available object class plus one for the "background" (to indicate that the object is not present). Finally, all the layers of this CNN are trained with a loss that tries to maximize the number of training windows for which this new layer gives the highest score to the correct class.

At test time, R-CNN extracts features from every window in a test image and scores them with its linear SVM classifiers (fig. 2.3). Finally, it applies class-specific non-maximum suppression (NMS) to eliminate windows that are not locally the highest-scored for a class, reducing the final set of detections.

As described in sec. 1.2.1, the R-CNN object class detector performs considerably better than all previous detection methods. There are two reasons to its success. First, it replaces hand-engineered features like HOG (Dalal and Triggs, 2005a) or SIFT (Lowe, 2004) with high level object representations produced by a complex CNN classifier, trained for window classification. The weights of the network are learnt automatically from large collections of images, leading to very discriminative CNN features. And second, R-CNN uses object proposal techniques to extract features from only a few thousands windows. It would be impractical for R-CNN to extract features from a sliding-window grid with more than a million windows.

A publicly available implementation of R-CNN is available at (Girshick et al., 2014) and we use it in our experiments in chapters 3, 4, 5 and 6.

Improvements over the original work. Some works successfully improved the R-CNN object detector. For example, *Fast R-CNN* (Girshick, 2015) improved performance and training/testing speed by sharing computation in the CNN across different windows in an image. *Faster R-CNN* (Ren et al., 2015) improved these even further by proposing a new way to generate region proposals within the network itself. Finally Simonyan and Zisserman (2015) proposed a 16-layer network architecture that is deeper than the 7-layers of Krizhevsky et al. (2012) and leads to a substantial improvement in performance when used in R-CNN.

Works that use R-CNN or CNNs. R-CNN and its successors (Fast-RCNN, Faster R-CNN) have become a reference model for object recognition and they are widely used by the community in tasks like object detection (Hariharan et al., 2014; Gonzalez-Garcia et al., 2015; Papadopoulos et al., 2016; Shrivastava and Gupta, 2016), semantic part detection (Zhang et al., 2014a), pedestrian detection (Hosang et al., 2015), object segmentation (Pinheiro et al., 2016; Shrivastava and Gupta, 2016) and action detection (Peng and Schmid, 2016).

Moreover, for completeness, it is worth noticing that CNNs (not in the context of R-CNN) have been established as a powerful class of models for visual recognition problems. Encouraged by the results of Krizhevsky et al. (2012), a lot of methods have used CNNs and achieved impressive results. For example, on image classification (Simonyan and Zisserman, 2015; Szegedy et al., 2015), object detection (Girshick et al., 2014; Sermanet et al., 2014; He et al., 2014; Szegedy et al., 2015; Gidaris and Komodakis, 2015; Redmon et al., 2016; Shrivastava et al., 2016), semantic segmentation (Long et al., 2015; Hariharan et al., 2015; Caesar et al., 2015), fine-grained recognition (Zhang et al., 2014a; Hariharan et al., 2015; Lin et al., 2015), action recognition (Karpathy et al., 2014), pedestrian detection (Tian et al., 2015; Zhang et al., 2016), face recognition (Taigman et al., 2014), human pose estimation (Toshev and Szegedy, 2014; Wei et al., 2016), RGB-D object detection (Gupta et al., 2014), scene recognition (Zhou et al., 2014), caption generation (Vinyals et al., 2015; Xu et al., 2015) and visual question answering (Malinowski et al., 2015).

2.4 Other object class detectors

In this section we briefly present two families of work not used in the experiments presented in this thesis, but historically relevant.

2.4.1 Generalized Hough-transform

An older family of work is based on the generalized Hough-transform (Ballard, 1981). Leibe et al. (2004) were the first to propose an object detector of this family: class-specific "implicit shape models". In this model, the detections of local interest points (fig. 2.4a, top) cast probabilistic votes for possible locations of the centroid of the whole object. A Hough image is used to accumulate all these votes and its maxima corresponds to the detection hypotheses. Leibe et al. (2004) uses generative codebooks of part appearances. Many works extended on this (Leibe and Schiele, 2003; Berg et al., 2005; Ferrari et al., 2006; Opelt et al., 2006; Maji and Malik, 2009; Gall and Lempitsky, 2009; Ferrari et al., 2010). For example, an accurate detector was developed by Ferrari et al. (2010). They learned class-specific "explicitly shape models" that integrate Hough-style voting with a non-rigid point matching algorithm to localize the model in cluttered images. Another model is presented by Maji and Malik (2009). Their object detector discriminatively learns the weights of the implicit shape model in a max-margin framework that directly optimizes the classification performance. Furthermore, Gall and Lempitsky (2009) trained discriminative class-specific "Hough forests" based on random forests (Criminisi et al., 2011) that are able to cast probabilistic votes within the Hough transform framework.

2.4.2 Bag-of-words (BOW)

BOW approaches (Sivic et al., 2005; Harzallah et al., 2009; Van de Sande et al., 2011; Prest et al., 2012; Chen et al., 2013a; Cinbis et al., 2013) represent an image (or window in our case) as an orderless collection of local features (fig. 2.4). The BOW method has been originally proposed for information retrieval, where it is used to categorize documents in a text corpus, where each document is represented by its word frequency. In the visual domain, an image (or a window) is the analogue of a document and it can be represented by a bag of vector quantized features, called visual-words. These features are extracted directly from image pixel responses and they usually describe statistics such as color distributions, edge orientations, etc. (fig. 2.4b). When using BOW for

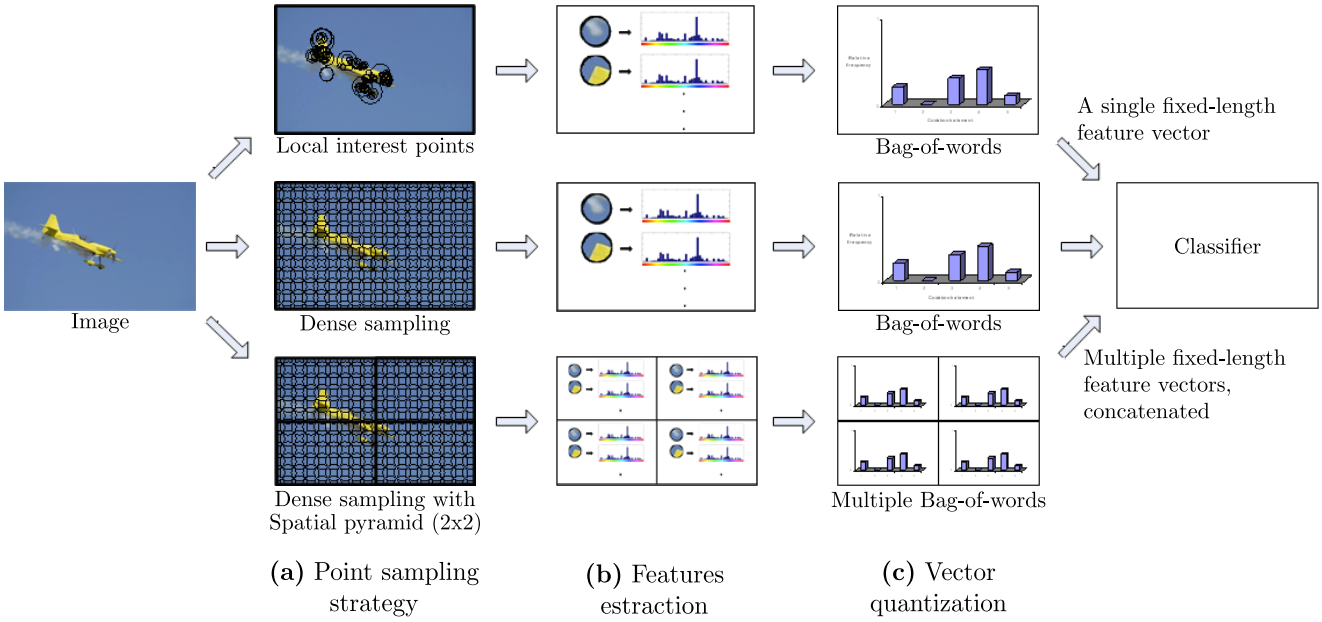


Figure 2.4: *Illustration of a bag-of-words object class detector. Points are initially sampled from the image, either sparsely or densely* ***(a)****. Features are then extracted from these points* ***(b)*** *and vector quantized* ***(c)****. This results in a fixed-length feature vector representing an image, which is fed into a classifier. Note: the figure is adapted from figs. 1 and 2 of (Van De Sande et al., 2010).*

object class detection, features are extracted on object bounding-boxes and negative windows. Some older works like Mikolajczyk and Schmid (2004) extract features on sparse interest points (fig. 2.4a, top), while more recent ones like Van De Sande et al. (2010) extract them densely (fig. 2.4a, mid). These are then used to train complex classifiers, like intersection-kernels (Maji et al., 2008) or χ^2-kernels (Zhang et al., 2007).

Spatial pyramid matching (SPM) models improve orderless bag-of-words representations by augmenting it with spatial pooling (the *pyramid*, fig. 2.4, bottom) (Lazebnik et al., 2006). Technically, bag-of-words are computed on each tile of the pyramid and than concatenated to form the final feature vector. This representation can learn spatial information about the object. For example, it can learn that for the objects "car", "motorbike" and "bus", it is more likely to find a tire on the bottom part of the image rather than on the top.

Ideally, models would use all the available negative windows to train. Unfortunately, this is computationally infeasible. Dalal and Triggs (2005b) alleviated this issue by using a technique called *hard-negative mining*. After training an initial clas-

sifier with positive bounding-boxes and a few randomly sampled negative windows, this technique searches exhaustively over the whole dataset for negative samples that are mistakenly classified as positives, with high confidence, by the current classifier. These samples are then added to the initial pool and used to train a better classifier. As these negatives were scored positively with high confidence, they have a large impact on the training of the new classifier. Prest et al. (2012) refined this idea and proposed to train the initial classifier using positive bounding-boxes and 10 window proposals (Alexe et al., 2010) from each negative training image. This ensures higher variety in the space of negative samples, while considerably reducing the training time of the classifier.

As mentioned earlier, complex non-linear kernels are usually the best choice as classifiers. These are however very expensive and often they cannot be used in a reasonable amount of time. Inspired by Viola and Jones (2001), Vedaldi et al. (2009) proposed to approximate these demanding kernels by using a cascade of increasingly powerful classifiers: first linear, then quasi-linear and only at the very end non-linear kernels. On a different direction, (Maji et al., 2008; Vedaldi and Zisserman, 2010; Maji et al., 2013) presented ways to approximate all the additive homogeneous kernels (e.g., the expensive histogram intersection and χ^2-kernels) into compact linear representations, by transforming the original feature vectors into new representations.

Finally, quantizing features into bags is a lossy process (Perronnin et al., 2010). To overcome this issue, Cinbis et al. (2013) present an SPM object detector based on the *Fisher vector* (Jaakkola and Haussler, 1999) image representation. This representation extends the BOW representation by encoding additional information about the distribution of the descriptors.

Chapter 3

Context forest for object class detection

3.1 Introduction

Global image appearance carries information about properties of objects in the image. For instance, a picture of a highway taken from a car is more likely to contain cars from the back viewpoint than from the side (fig. 3.1) and a picture taken under water is more likely to contain a fish than a car. This show how the global image appearance of images can help understanding what objects are present and what they look like. Moreover, another property that can be inferred from global image appearance is the rough location of object instances (Russell et al., 2007). For instance, an urban scene with cars parked in front of a building, shows cars in the bottom half of the image (fig. 3.2).

In this chapter we exploit this observation for object class detection. We propose Context Forest (ConF): a technique for learning the relation between the global appearance of an image and the properties of the objects it contains. Given only the global appearance of a test image, ConF retrieves a subset of training images that contain objects with similar properties. ConF is based on the Random Forest (Breiman, 2001; Criminisi et al., 2011) framework, which has the ability to learn complex, non-linear relations between global image appearance and objects properties with high computational efficiency. It is very flexible and only requires these properties to be defined through a distance function between two object instances, e.g., their appearance similarity or difference in location. We demonstrate ConF by learning to predict three properties: aspects of appearance, location in the image, and class membership.

Multi-component detectors (Dollár et al., 2008; Felzenszwalb et al., 2010b; Malisiewicz et al., 2011; Divvala et al., 2012; Gu et al., 2012; Aghazadeh et al., 2012; Zhu et al., 2012; Drayer and Brox, 2014) model an object class as a mixture of components, each trained to recognize a different aspect of appearance. For example, different viewpoints (e.g., front and back view of a car (Felzenszwalb et al., 2010b; Gu and Ren, 2010)) or articulation states (e.g., a person sitting vs standing (Sun and Savarese, 2011)). When trained on a large set, such a detector has many components (Zhu et al., 2012; Divvala et al., 2012), which all need to be evaluated on a test image, making it slow. Instead, we use ConF to select a subset of model components which is most relevant to a particular test image. We then run only those components, obtaining a speed-up (2x for DPM (Felzenszwalb et al., 2010b) and 10x for EE-SVM (Malisiewicz et al., 2011), sec. 3.4.3). Hence, ConF makes large multi-component detectors practical. This is particularly useful for EE-SVMs, as they have as many components as there are training instances. Interestingly, in some cases we even gain a small improvement in accuracy by not running some components that would produce false positive detections.

Moreover, we train a second ConF to predict at which positions and scales objects are likely to appear in a given test image, analogue to (Torralba, 2003; Russell et al., 2007; Liu et al., 2009). By incorporating this information in the detector score at test time, we reduce the false positive rate by removing detections at unlikely locations. Experiments show an mAP improvement of 2% (sec. 3.4.4).

Finally, we train a third ConF to predict what object classes are present in an image, as in (Harzallah et al., 2009; Song et al., 2011; Van De Sande et al., 2013). We use it at test time by only running detectors of classes predicted to be in the test image. In experiments on a 200-class dataset (Russakovsky et al., 2015a), this allows us to run just 10 detectors per image on average, while also improving mAP by 5%, as it removes false positives produced by detectors of classes unlikely to be present in the image (sec. 3.4.5).

All these experiments demonstrate that ConF is a general technique that can predict various kinds of object properties. We carry out an extensive comparison to standard nearest-neighbour techniques for such context-based predictions (Torralba, 2003; Russell et al., 2007; Rabinovich et al., 2007; Liu et al., 2009; Tighe and Lazebnik, 2010). This analysis shows that ConF predicts object properties from global image appearance more accurately, while being much faster and memory efficient (sec. from 3.4.2 to sec. 3.4.6).

The rest of the chapter is organized as follows. We start by reviewing related work in sec. 3.2. Sec. 3.3 explains ConF, our main contribution. In sec. 3.4 we describe our experimental settings and we present several experiments to validate the benefits of ConF on two object class detectors. Finally, we conclude in sec. 3.5.

This work has been published at BMVC (Modolo et al., 2015a) (*oral presentation*).

3.2 Related work

3.2.1 Context

The use of context for object detection is a broad research area. Some works (Rabinovich et al., 2007; Heitz and Koller, 2008; Desai et al., 2009; Choi et al., 2010; Felzenszwalb et al., 2010b) model context as the interactions between multiple object class detectors in the same image. Here, we model context as the relation between global image appearance and properties of the objects within them, as in (Murphy et al., 2003; Torralba, 2003; Russell et al., 2007; Harzallah et al., 2009; Liu et al., 2009; Tighe and Lazebnik, 2010; Song et al., 2011; Yang et al., 2014). These works have shown that global image descriptors give a valuable cue about which classes might be present in an image and where they are located. Many object detectors (Torralba, 2003; Murphy et al., 2003; Russell et al., 2007; Harzallah et al., 2009; Song et al., 2011; Van De Sande et al., 2013) employ such global context to remove out-of-context false-positive detections. A similar approach is also used for image parsing (Liu et al., 2009; Tighe and Lazebnik, 2010; Yang et al., 2014). Most of these works have a nearest neighbour core: they first retrieve a small subset of training images which are most globally similar to a test image, and then transfer the relevant statistics of the object properties in this retrieval set to the test image. In our work instead the retrieval set is estimated by ConF, which is *explicitly trained* to return images containing objects with similar properties to those in the test image. ConF has several advantages over nearest-neighbour approaches: (i) it can learn highly complex non-linear dependencies between the global descriptor and the object property. As a result, it estimates it more accurately; (ii) in large training sets, nearest neighbour becomes very slow, as its complexity is linear in their size. ConF is much faster and more memory efficient; (iii) ConF supports any objective function, which might even be evaluated on a different data representation than the input at test time. This is a crucial feature for our problem, as we want to predict properties of objects, but based on global image features.

3.2.2 Multi-component detectors

These detectors (Dollár et al., 2008; Felzenszwalb et al., 2010b; Malisiewicz et al., 2011; Gu et al., 2012; Divvala et al., 2012; Aghazadeh et al., 2012; Zhu et al., 2012; Drayer and Brox, 2014) model an object class as mixture of components. They can be slow when trained from large training sets as they need many components to reach peak performance (Zhu et al., 2012; Divvala et al., 2012). While we present experiments on DPM (Felzenszwalb et al., 2010b) and EE-SVM (Malisiewicz et al., 2011), ConF can in principle speed-up any multi-component detector. The problem of speeding-up detection, especially when evaluating many HOG templates, has also been attacked by other works (Felzenszwalb et al., 2010a; Song et al., 2012; Dean et al., 2013). However, these are specialized to the HOG/DPM detectors, as they exploit its internal structure. Moreover, (Song et al., 2012; Dean et al., 2013) are multi-class methods and achieve a speed-up only when the number of classes predicted at the same time is large. In contrast, ConF can speed-up detection of even a single class (sec. 3.4.3). Finally, we believe ConF can offer a complementary way to speed-up object detection and can potentially be combined with these works.

3.2.3 EE-SVM

The EE-SVM (Malisiewicz et al., 2011) is an extreme case of multi-component detection, where a separate component is created for each training example (sec. 2.2). Because of this, EE-SVM can benefit the most from dynamically selecting components using ConF. As discussed in sec. 2.2, EE-SVM is widely used for applications beyond object detection (Shrivastava et al., 2011; Singh et al., 2012; Aytar and Zisserman, 2012; Endres et al., 2013; Dong et al., 2013; Tighe and Lazebnik, 2013; Juneja et al., 2013; Gronat et al., 2013; Song and Xiao, 2014; Vezhnevets and Ferrari, 2014; Aubry et al., 2014a) because it explicitly associates a training example to each object it detects in a test image. This enables transferring meta-data such as segmentation masks (Malisiewicz et al., 2011; Tighe and Lazebnik, 2013), 3D models (Malisiewicz et al., 2011), viewpoints (Aubry et al., 2014a), GPS locations (Gronat et al., 2013) and part-level regularization (Aytar and Zisserman, 2012). Furthermore, EE-SVM can also be used for discovering objects parts (Singh et al., 2012; Endres et al., 2013), scene classification (Singh et al., 2012; Juneja et al., 2013), object classification (Dong et al., 2013), image parsing (Tighe and Lazebnik, 2013), image matching (Shrivastava et al., 2011), automatic image annotation (Vezhnevets and Ferrari, 2014) and 3D object de-

tection (Song and Xiao, 2014). All these applications can potentially be accelerated by ConF, extending its use beyond object detection.

3.3 Estimating object properties from context

We exploit the observation that global image appearance contains information about properties of the objects inside it. We focus on three properties: aspect, location, and class. We propose a new method (ConF) based on the Random Forest framework (Breiman, 2001; Criminisi et al., 2011), which learns the relation between global image features and the properties of the object in that image. Given only the global image appearance of a test image, ConF retrieves a subset of training images that contain objects with similar properties. The properties in this retrieval set can then be used to modify the behaviour of the object detector, e.g., by running only some components or by downgrading the score of false positive detections at unlikely locations. In the following, we first describe ConF in its general form (sec. 3.3.1), and then specialize it for each of the three properties we consider (sec. 3.3.2 to 3.3.4).

3.3.1 Context Forest (ConF)

Given a training set $\mathcal{T}$ the goal of ConF is to map the global appearance $\phi(I_t)$ of a test image I_t into a retrieval set $\mathcal{R} \subset \mathcal{T}$. We want to construct a mapping, such that properties of objects in images of $\mathcal{R}$ are similar to the properties of objects in I_t (e.g., appearance, location, class).

3.3.1.1 ConF at training time

ConF at training time learns an ensemble of decision trees (forest) that operates on global image features $\phi(I)$. We construct each tree by recursively splitting the training set $\mathcal{T}$ at each node. We want the leaves of the trees to contain images whose objects properties are compact according to some measure $c(\mathcal{T}_l)$, where $\mathcal{T}_l$ are the training images in leaf l. Each internal node n contains a binary split function $f(\phi(I), \theta_n)$, where θ_n are its parameters. Let $\mathcal{T}_n$ be the training images that reached node n, then $f(\phi(I), \theta_n)$ will split $\mathcal{T}_n$ into two subsets $\mathcal{T}_l$ and $\mathcal{T}_r$. We use axis-aligned weak learners as f (Criminisi et al., 2011). The split function $f(\phi(I), \theta_n)$ applies a threshold to one of the dimensions of the image feature vector $\phi(I)$. Following the extremely randomized forest approach (Moosman et al., 2006), for each node we randomly sample several

thousand possible splits θ and choose the one that maximizes the joint compactness:

$$\theta_n = \arg\max_{\theta} c(\mathcal{T}_l) + c(\mathcal{T}_r) \tag{3.1}$$

$$\text{s.t. } \forall I \in \mathcal{T}_l,\ f(\phi(I),\theta) = 0,\ \forall I \in \mathcal{T}_r,\ f(\phi(I),\theta) = 1$$

where compactness is defined as

$$c(\mathcal{T}) = \frac{1}{N(\mathcal{T})^2} \frac{1}{\sigma^2\sqrt{2\pi}} \sum_{w_i \in \mathcal{T}} \sum_{w_j \in \{\mathcal{T} \setminus w_i\}} e^{-\frac{1}{2}\frac{D(w_i,w_j)^2}{\sigma^2}} \tag{3.2}$$

where $N(\mathcal{T})$ is the number of ground-truth object bounding-boxes in $\mathcal{T}$ and $D(w_i, w_j)$ is a distance measure between the properties of two object bounding-boxes w_i and w_j. Note how the inner summation in (3.2) is an estimation of the density of the distribution induced by all bounding-boxes in $\{\mathcal{T} \setminus w_i\}$, evaluated at w_j. This value is high if w_j has other bounding-boxes nearby. The estimate is done with a Gaussian Kernel Density estimator (Parzen, 1962) (KDE). We estimate the standard deviation σ from the entire training set once before training the forest. This determines the scale of the problem, i.e. at which range two bounding-boxes should be considered close. For each training bounding-box w_i we compute its k-nearest neighbours in the whole training set and compute the standard deviation over them. Finally, we set σ as the median of these standard deviations over all bounding-boxes.

By employing different compactness measures c, we can use ConF to learn relations between different object properties and global image features. Later we show how to use it for selecting components relevant for a test image (sec. 3.3.2), for estimating likely object locations in it (sec. 3.3.3), and to predict which object classes it is likely to contain (sec. 3.3.4).

3.3.1.2 ConF at test time

ConF at test time operates in two phases (fig. 3.1, 3.2 and 3.3). First, the test image I_t is passed through the forest, reaching a leaf in each tree. Thereby, each tree selects the subset of training images contained in that leaf. For each training image I_i, we count how many trees have selected it, forming the score $\eta(I_i, I_t)$. We now construct the retrieval set $\mathcal{R}$ to contain the k most frequently selected training images. In our experiments $k = 20$.

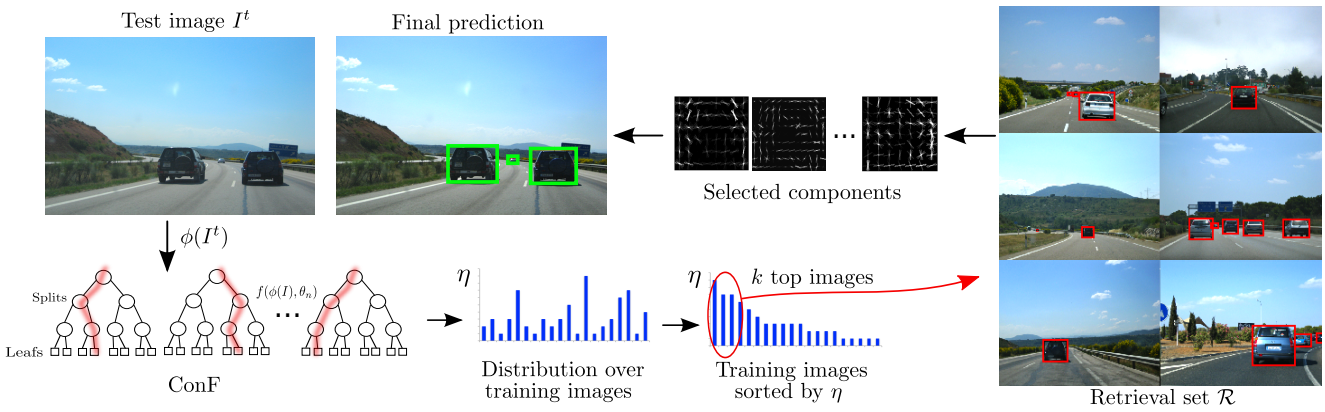


Figure 3.1: *Illustration of ConF selecting components for a test image (sec. 3.3.1.)*

Note how the split function f and the compactness measure c operate in structurally different spaces. While f operates on the global image features $\phi(I)$, c is measuring the similarity of properties of *objects inside the images*. In this way ConF learns the relation between the two. Importantly, c is neither convex nor differentiable in $\phi(I)$ and we are only able to learn this inter-space relation thanks to the unique advantages of Random Forests.

3.3.2 ConF for component selection

Here we assume that each component of an object detector has been previously trained from a visually compact set of object instances, and that we know the component id ξ_j of each training instance j. In EE-SVMs each training instance leads to a different component. In DPM the component id of a training instance can be inferred by the output of the training procedure (Girshick et al., 2012). Based on this information, we train a ConF to select a small subset of components to run on a given test image I_t.

Training. To train ConF for component selection we define the distance $D(w_i, w_j)$ in (3.2) as the L2 distance between the HOG descriptors of bounding-boxes w_i and w_j.

Test. We pass the test image I_t through ConF obtaining a retrieval set $\mathcal{R}$. We then estimate a posterior distribution $p(\xi_j|I_t)$ over components ξ_j given I_t. As a training image I might contain multiple instances from different components, each training image is "labelled" by a distribution over components $p(\xi_j|I)$. We estimate the component distribution for the test image I_t as the average over the training images in the retrieval set

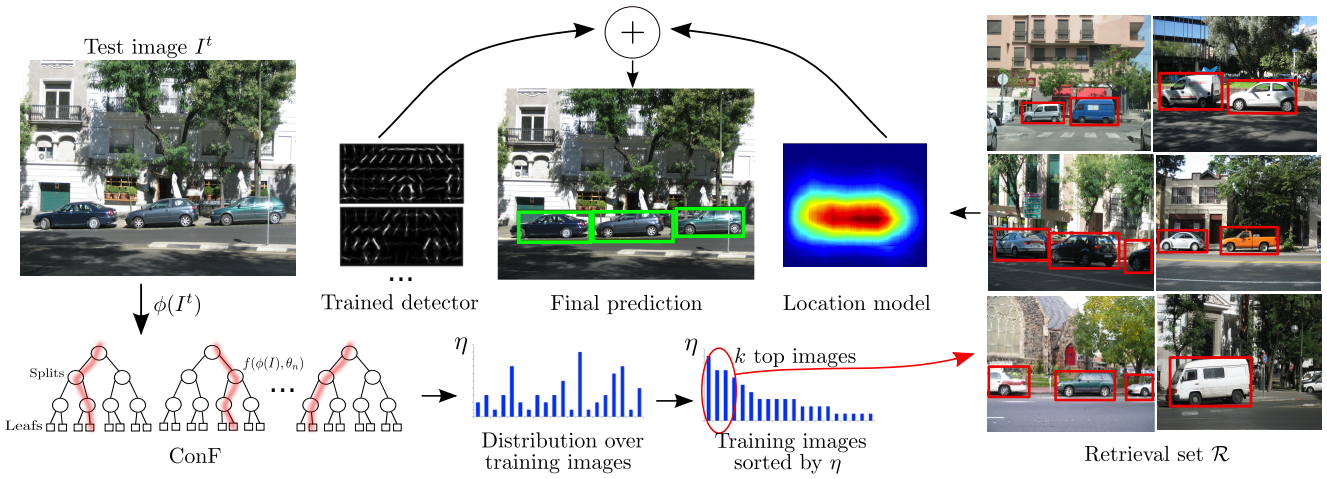


Figure 3.2: *Illustration of ConF predicting object locations (sec. 3.3.3).*

$\mathcal{R}$

$$p(\xi_j|I_t) = \frac{1}{|\mathcal{R}|}\sum_{I\in\mathcal{R}} p(\xi_j|I) \tag{3.3}$$

Based on this distribution, we can now select which components to run on I_t. We rank components by their probability and iteratively pick them until their combined probability mass exceeds a threshold γ. This threshold controls a trade-off between running few components and getting high detection performance. An interesting aspect of our formulation is that the number of selected components changes depending on the test image. A test image with a characteristic appearance matching training images with a systematic recurrence of a few components will lead to a peaky $p(\xi_j|I_t)$. In this case it is safe to run only a few components and we obtain a substantial speed-up. On the other hand, if the ConF is uncertain about the contents of the test image, then the entropy of $p(\xi_j|I_t)$ will be high, and many components will be selected. In the extreme case, for a very difficult test image, our procedure naturally degenerates to the default case of running all components.

3.3.3 ConF for object location

At test time, a typical detector scores windows in the test image I_t, based on their appearance only. We propose here to augment the detector's scores by adding knowledge about likely positions and scales of the object class, derived purely from the global appearance of I_t.

Training. We train two ConFs to predict object positions and scales, respectively. To

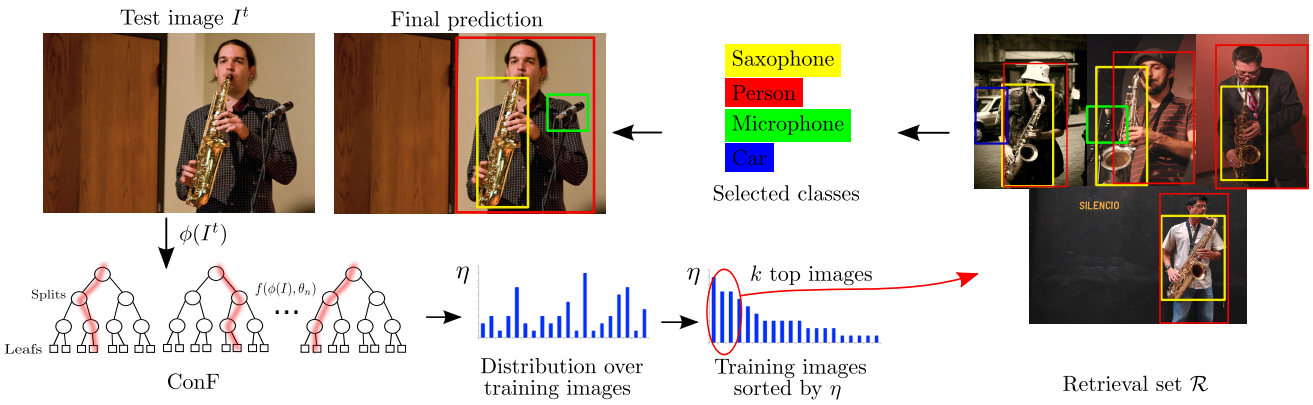


Figure 3.3: *Illustration of ConF predicting class membership (sec. 3.3.4).*

do so, we employ a different measure of compactness, substituting the distance between two bounding-boxes in (3.2) with D_{POS} (or D_{SCALE}). We define $D_{\text{POS}}(w_i, w_j)$ as the L2 distance between w_i and w_j. We define $D_{\text{SCALE}}(w_i, w_j) = \max(\frac{H_i}{H_j}, \frac{H_j}{H_i}) \cdot \max(\frac{W_i}{W_j}, \frac{W_j}{W_i})$ as the difference in their scale (W and H refer to width and height).

Test. We first pass the test image I_t through ConF obtaining a retrieval set $\mathcal{R}$, and then compute the following score for each window w in the test image:

$$\frac{1}{N(\mathcal{R})} \sum_{w_i \in \mathcal{R}} \frac{1}{\sigma^2 \sqrt{2\pi}} e^{-\frac{1}{2}\frac{D(w,w_i)^2}{\sigma^2}} \tag{3.4}$$

where $N(\mathcal{R})$ is the number of object instances in $\mathcal{R}$, and D is either D_{POS} or D_{SCALE} . We learn σ from the entire training set as in sec. 3.3.1. This score captures how likely a window is to cover an object based on its position/scale. Finally, we linearly combine the position/scale scores with the detector's score of w. The usual non-maxima suppression stage follows.

3.3.4 ConF for class selection

In multi-class problems, a typical system would run detectors for all classes on all test images (Russakovsky et al., 2015a). Instead, here we use ConF to predict what classes are present in each image, and run only the corresponding detectors. This greatly reduces the number of detectors run, and removes some false-positives.

Training. To train ConF to predict what classes are present in an image, we use the following distance between two bounding-boxes in the compactness function (3.2):

$$D_{\text{CLASS}}(w_i, w_j) = \begin{cases} 0 & \text{if } w_i \text{ and } w_j \text{ are objects of the same class} \\ \infty & \text{otherwise} \end{cases} \tag{3.5}$$

This definition simplifies (3.2) considerably, as $e^{-\frac{1}{2}\frac{D(w_i,w_j)^2}{\sigma^2}}$ can only be 0 or 1. Note how the inner summation in (3.2) now simply counts the number of objects $w_j \in \{\mathcal{T} \backslash w_i\}$ of the same class as w_i. We can rewrite (3.2) equivalently as:

$$c(\mathcal{T}) = \frac{1}{N(\mathcal{T})^2} \frac{1}{\sigma^2\sqrt{2\pi}} \sum_{c \in C} N(\mathcal{T}, c) \cdot (N(\mathcal{T}, c)) - 1) \tag{3.6}$$

where $N(\mathcal{T}, c)$ counts how many objects in $\mathcal{T}$ belong to class c. Note how (3.6) allows an important speed up over (3.2), as it avoids computing the KDE. Evaluating (3.6) takes time $O(|\mathcal{T}|)$, compared to $O(|\mathcal{T}|^2)$ for (3.2).

Test. We pass the test image I_t through ConF obtaining a retrieval set $\mathcal{R}$. We then estimate a posterior distribution $p(c_j|I_t)$ over object classes c_j given I_t, analog to what done for components of one class in sec. 3.3.2. Based on this distribution, we can now select which detectors to run on I_t. In our experiments we run all detectors with $p(c_j|I_t) > 0$.

3.4 Experiments

We present a series of experiments on two datasets (sec. 3.4.1). In sec. 3.4.2 we first evaluate how good the retrieval sets generated by ConF are. Then, we show how ConF for component selection (sec. 3.4.3), object location (sec. 3.4.4) and class selection (sec. 3.4.5) can be used to improve object detection. Finally, we show how ConF is more computation and memory efficient then a nearest neighbour approach (sec. 3.4.6).

3.4.1 Datasets

We present experiments on two datasets: a 2-class dataset we call BigCH, and the 200-class `val` subset of ILSVRC2014 (Russakovsky et al., 2015a). The first one has few classes, but many training instances per class (on average $\sim 21k$). We use it to evaluate ConF for selecting model components (sec. 3.4.3) and predicting object location (sec. 3.4.4). The second dataset has many classes, but much fewer training instances

Class / Dataset	Car			Horse		
	# pos imgs	# pos objs	# neg imgs	# pos imgs	# pos objs	# neg imgs
PASCAL VOC 2012	1161	2017	1161	482	710	482
ImageNet	6383	7120	6383	4550	6631	4550
SUN2012	828	1779	828	-	-	-
Labelme	6566	16743	6566	-	-	-
UIUC	828	889	500	-	-	-
PASCAL-10x	-	-	-	4065	6454	4065
Total	15766	28548	15438	10107	13071	10107
BigCH training set	14125	25774	13830	9097	12407	9097
BigCH test set	1641	2774	1608	1010	1388	1010

Table 3.1: *Statistics of BigCH, which we assembled by combining images from existing source datasets. The column "# pos imgs" reports the number of images containing the class in that dataset; "# pos objs" is the total number of instances of the class in all positive images; "# neg imgs" is the number of negative images we sampled from that dataset.*

per class (on average ~ 530). We use it to evaluate ConF for class selection (sec. 3.4.5).

BigCH is a 2-class dataset (*Car* and *Horse*, table 3.1, figure 3.4) that we assembled. Below we discuss the car dataset in detail. The horse dataset was designed analogously.

- **Source datasets.** It combines 6 existing datasets: PASCAL VOC 2012 (Everingham et al., 2012), ImageNet (Deng et al., 2009), LabelMe (Russel and Torralba, 2008), SUN 2012 (Xiao et al., 2010), UIUC (Agarwal et al., 2004) and PASCAL-10x (Zhu et al., 2012). PASCAL VOC 2012, PASCAL 10x, and ImageNet contain a wide variety of image types, with both difficult, cluttered images and easier images with big centered cars. UIUC has low resolution, gray-scale images of side-view cars. LabelMe and Sun 2012 contain mainly wide open street scenes with small cars. All these datasets have images not containing cars, that we use as negatives.

- **Positive images and ground-truth annotations.** We collected all the images with bounding-box annotations on cars. We took several steps to ensure assembling a high quality, clean dataset. First of all, we run a near-duplicate detector to remove duplicate images (which were a few hundreds). Next, we removed incorrect bounding-boxes not covering cars or covering the same car multiple times. Finally, as some images have unannotated cars, we annotated all missing in-

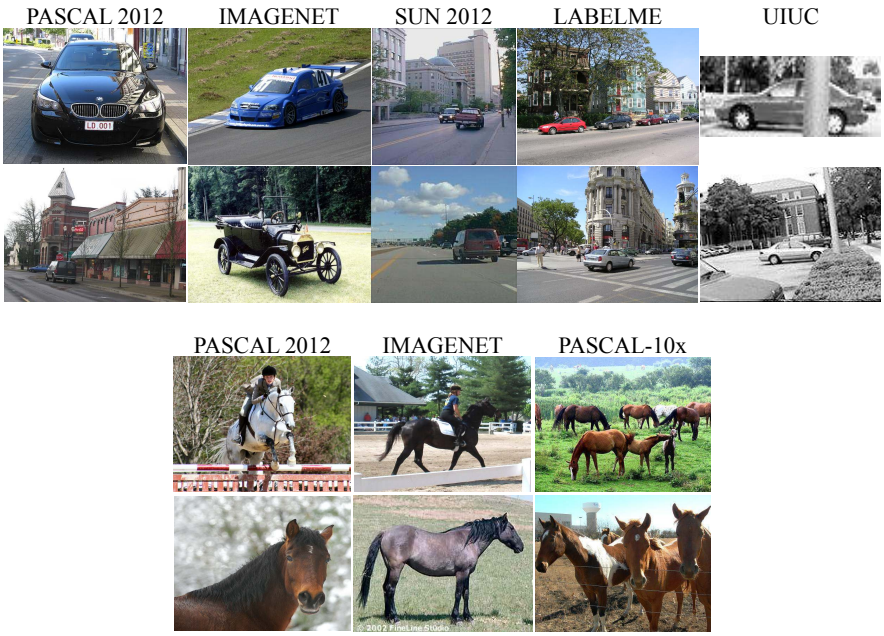


Figure 3.4: *Example images from the source datasets we use to assemble BigCH.*

stances with bounding-boxes for our entire test set (e.g., images from LabelMe). This enables reliable performance measurements.

- **Negative images.** We collect negative images from each dataset, so that it contributes an equal number of positive and negative images (except UIUC, which does not have enough negatives). To ensure variation, we randomly sampled the negative images.

- **Train/test splits.** We split the dataset into training (90%) and testing (10%) sets. Experiments in (Torralba and Efros, 2011) suggest the existence of various type of bias in our source datasets. Because of this, a detector can perform very differently on the same test set, when trained from different datasets, even if they have similar size. This is due to the different kind of images present in different datasets. Hence, splitting has to be done carefully. A naive split, e.g., using a single source dataset for testing and all others for training could be harmful. We split each dataset *in the same proportion* to avoid dataset bias (Torralba and Efros, 2011). These proportions correspond to the percentage of images that a particular source dataset contributes to the entire dataset (e.g., 8% of all car images are from PASCAL VOC 2012, and 40% from ImageNet).

ILSVRC2014 (Russakovsky et al., 2015a) is a 200-class dataset annotated by bounding-

boxes. Following (Girshick et al., 2014), we perform experiments on the `val` set and consider two disjoint subsets: $\texttt{val}_1$ (10k images) and $\texttt{val}_2$ (10k images). We use $\texttt{val}_1$ for training and test on $\texttt{val}_2$. In total, these sets contain >53k object instances.

3.4.2 Quality of retrieval sets

We quantify how similar ground truth object bounding-boxes from a test image I_t are to those in the retrieval set $\mathcal{R}$ returned by the method using the average density of $\mathcal{R}$ evaluated at the object properties in I_t

$$\frac{1}{Z\sigma^2\sqrt{2\pi}} \sum_{w_i \in I_t} \sum_{w_j \in \mathcal{R}(I_t)} e^{-\frac{1}{2}\frac{D(w_i,w_j)^2}{\sigma^2}} \tag{3.7}$$

where Z is the number of pairs of bounding-boxes in I_t and $\mathcal{R}$. The distance D and standard deviation σ vary depending on the property, as defined in sec. 3.3.2-3.3.4. We extract different features as global image descriptors $\phi(I)$, depending on the property. For the appearance and location properties, we extract SURF (Bay et al., 2008), LAB and SIFT (Lowe, 2004) features on a dense grid at multiple scales. For each feature type we train a class-specific codebook of 1000 visual words and construct a 2-level spatial pyramid (Lazebnik et al., 2006). Additionally, we also extract a GIST (Oliva and Torralba, 2001) descriptor for the image. Overall, we train ConF on a 16000 dimensional feature space. For the class membership property, we extract state-of-the-art convolutional neural network (CNN) descriptors of 4096 dimensions (Jia, 2013). These are the output of a CNN pre-trained specifically for image classification (Krizhevsky et al., 2012) and so they are very suited to the task. Finally, we train 750 trees for each property. We found this to be a good tread-off between achieving good performance and fast prediction.

Table 3.2 show results averaged over the test set. The higher the values are, the more the retrieval sets contain relevant information of the properties of the objects in the test image. Hence, higher values are better. For appearance and location, we also average results over the two classes (car and horse). Importantly, the ground-truth bounding-boxes from the test images are used for evaluation only, they were *not* used to produce the retrieval sets.

As a baseline, we return the whole training set as the retrieval set. This leads to a generic prior on image properties, independent of the test image. Moreover, we compare to the traditional way of building retrieval sets by k-nearest neighbours (kNN) (Torralba, 2003; Russell et al., 2007; Rabinovich et al., 2007; Liu et al., 2009;

Obj property	All train data	NN		ConF	
		1	10	1	10
Appearance	0.02	0.07	0.03	**0.09**	0.07
Position	0.36	0.90	0.70	**1.20**	1.00
Scale	0.04	0.07	0.06	**0.09**	0.08
Class	0.05	0.36	0.30	**0.39**	0.32

Table 3.2: *Evaluation of the quality of retrieval sets to predict object properties. Each entry represents the average density of the retrieval set $\mathcal{R}$ evaluated at the objects properties in the test images (eq. 3.7). We consider two sizes for $\mathcal{R}$: 1 and 10.*

Tighe and Lazebnik, 2010), defined on the same features as ConF. Both kNN and ConF greatly outperform the baseline, proving they return meaningful retrieval sets. This confirms the observation that the global image appearance conveys information about object properties inside the images. Moreover, ConF returns better retrieval sets than kNN across all object properties and retrieval set sizes evaluated. In the following sections we show how more accurate retrieval sets result in better detection performance and greater speed ups.

3.4.3 ConF for automatic component selection

We now use ConF to select object detector components relevant for a given test image on the BigCH dataset. We present experiments on two detectors: DPM (Felzenszwalb et al., 2010b), presented in sec. 2.1 and EE-SVM (Malisiewicz et al., 2011), presented in sec. 2.2.

DPM trains a mixture of components, each on a subset of the data with compact HOG appearance (Felzenszwalb et al., 2010b; Divvala et al., 2012). We use the publicly available implementation (Girshick et al., 2012) and train 16 components for the class *car* and 10 components for *horse* (these number of components lead to best performance on the large BigCH dataset)[1]. The EE-SVM model is composed of a separate

[1]Zhu et al. (2012) make the surprising observation that the performance of DPM decreases as the amount of training data increases. We were able to reproduce this undesirable behaviour on our large dataset. This is caused by the default DPM implementation not being able to use enough negative samples. We fixed this issue by making two changes to the DPM implementation: (1) we increased the maximal number of negative images used when building root filters. The initial value was capped to 200. This is insufficient when training from thousands of positive images as it leads to imbalanced training sets. (2) we doubled the cache size used to hold the feature vectors during hard negative mining (from 3GBs to 6GBs). Unfortunately the procedure to harvest hard negatives stops when the cache memory is full. On large training sets this results in ignoring many valuable negative images.

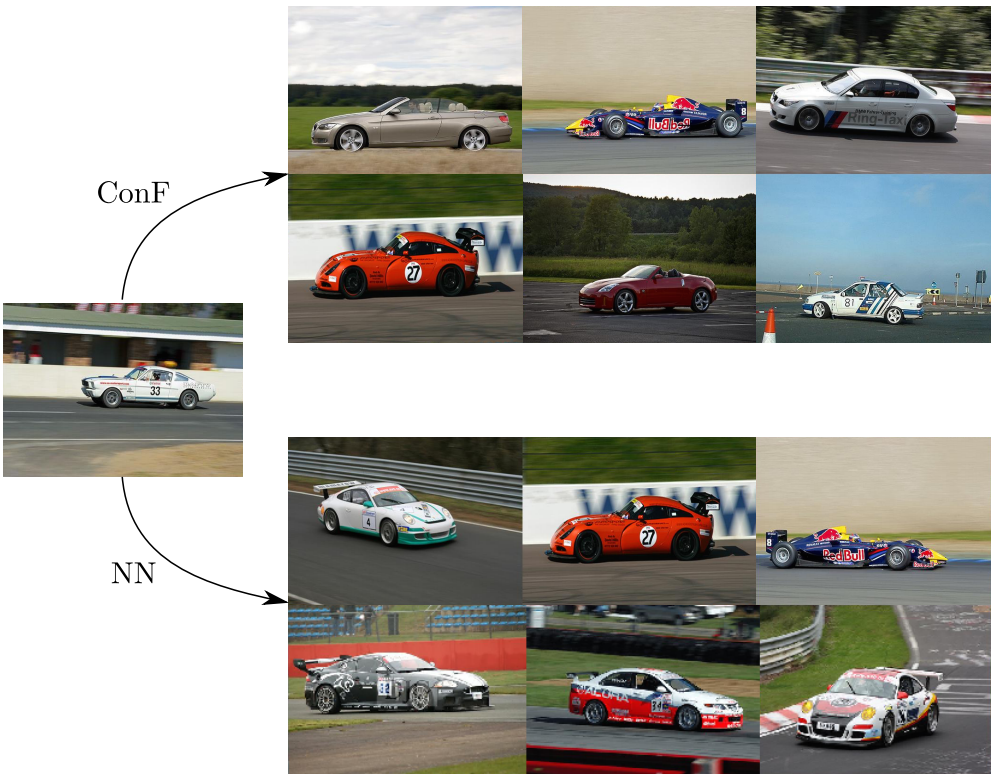
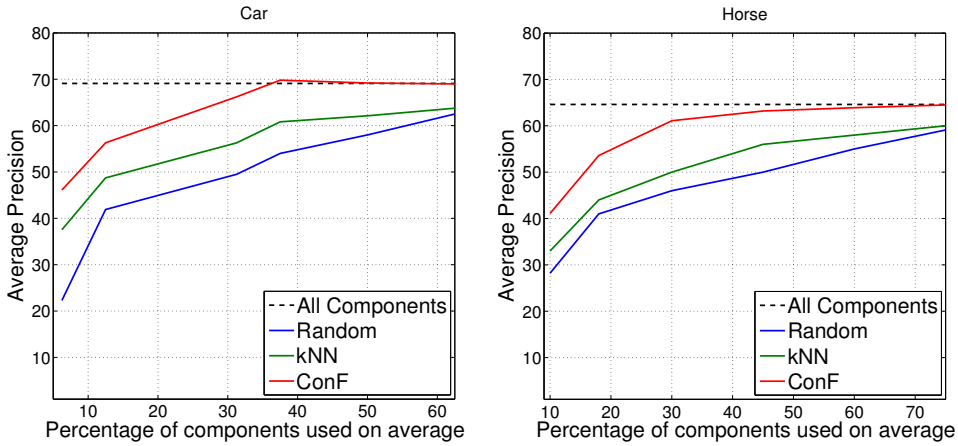


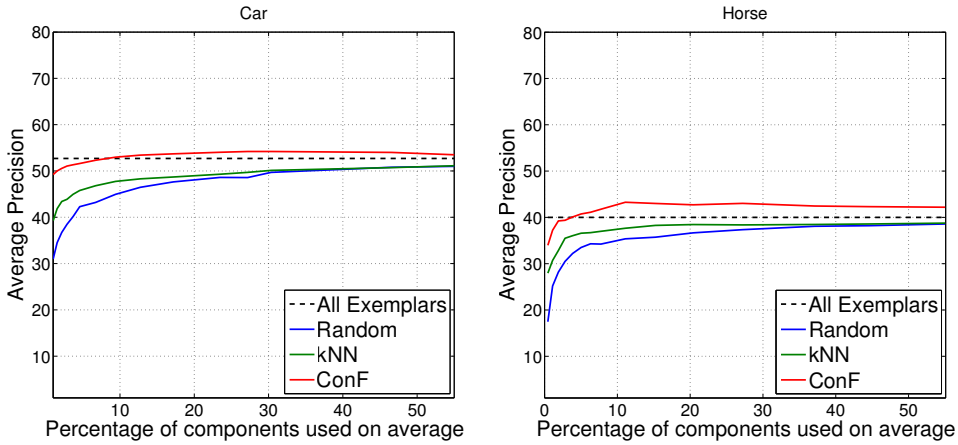
Figure 3.5: *Example of sets retrieved by ConF (top) and NN (bottom) for component selection. While both techniques return retrieval sets with racing cars, only ConF returns cars consistently facing left. This is because ConF was trained to retrieve visually compact sets of object instances, leading to accurate selection of the right model component.*

linear SVM for every training instance (exemplar), trained using the exemplar as the only positive against all negatives in the training set (here on HOG features). We refer to a single exemplar SVM as a component of the EE-SVM model, by analogy with DPM components. We use the publicly available implementation (Malisiewicz, 2011).

Fig. 3.6 shows the evolution of AP while increasing the percentage of components used (higher is better). We compare to building retrieval sets by kNN, and to a baseline which randomly selects components without looking at the test image. ConF outperforms the baseline and kNN for both object classes, for both detectors, and over the whole range of the plots. By employing ConF, we closely match the performance of the full DPM model by running roughly half of the components. We match the performance of a full EE-SVM when running less than 10% of the components. Even in the extreme case of running just *one* EE-SVM component, the AP is about 90% of that of the full model. Interestingly, for EE-SVM on the horse class, ConF *improves AP by 3%* over the full ensemble using all components, when running *10× fewer components*. There is also a minor improvement in AP for EE-SVM on the car class. This



(a) DPM



(b) EESVM

Figure 3.6: *Results of applying ConF automatic component selection. The points on the plot correspond to different choices for the threshold γ (sec. 3.3.2). The horizontal axis is the average amount of components used. The vertical axis is the AP of the detector using components selected by ConF.*

comes from dropping some components that lead to false positives.

These experiments demonstrate the ability of ConF to select relevant components given just global image appearance. This makes EE-SVMs practical even when trained from large sets with tens of thousands of exemplars (the average runtime for a test image decreases from 10 minutes to 1 minute on our machine with 4 i5-core 2.00GHz processors and 16 GB memory). Fig. 3.7 shows some example results.

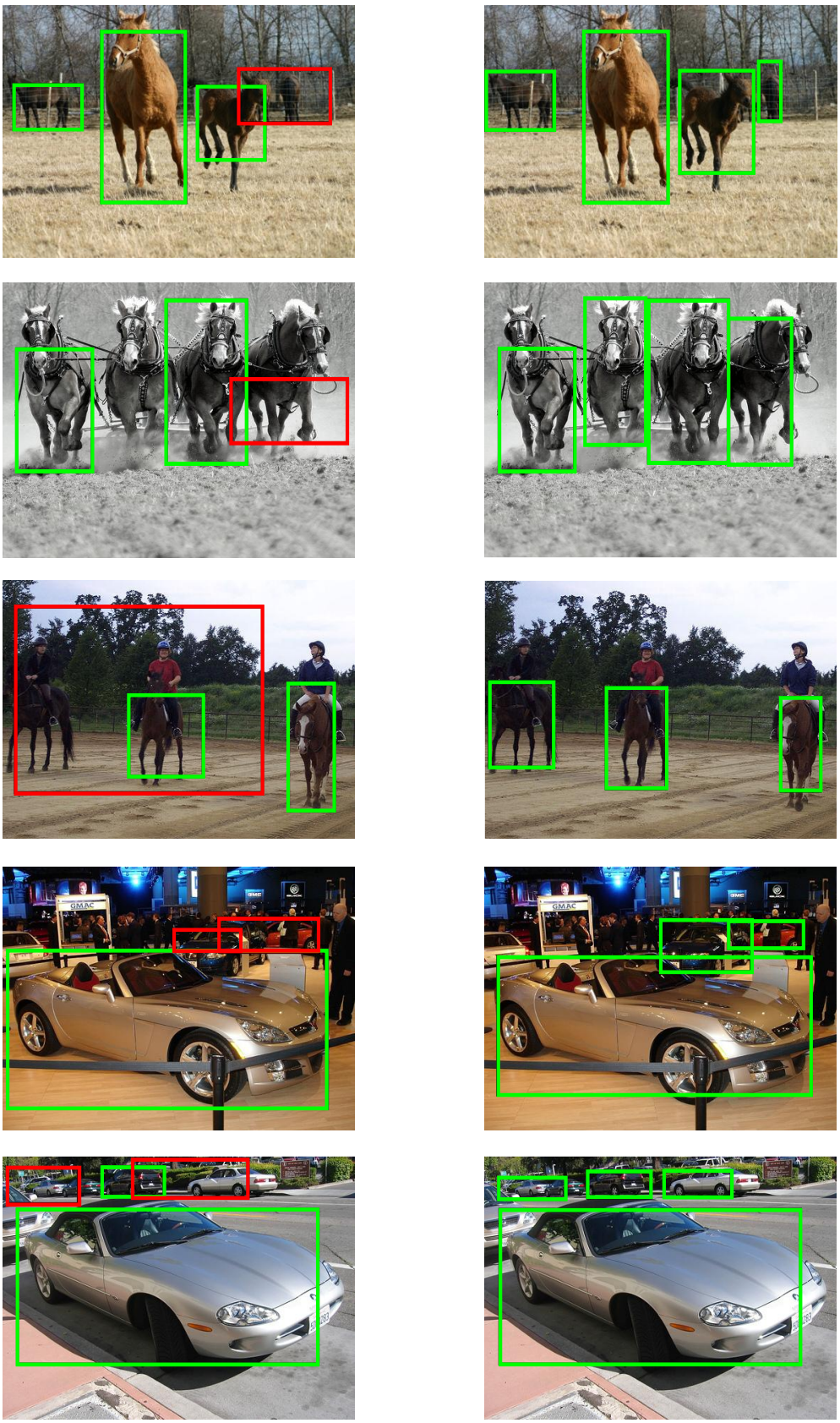


Figure 3.7: *Detections obtained before (left) and after (right) applying ConF for component selection. Green bounding-boxes highlight correct detections, while red ones show false positives.*

DPM - *Car*			EE-SVM - *Car*		
None	NN	ConF	None	NN	ConF
69.1	0	+2.1	52.7	0	+2.3

DPM - *Horse*			EE-SVM - *Horse*		
None	NN	ConF	None	NN	ConF
64.6	0	+0.7	40	0	+1.1

Table 3.3: *Results of augmenting the detector score with the location model derived by ConF (sec. 3.3.3) and NN compared to not using location model at all.*

3.4.4 ConF for object locations

Here we demonstrate how ConF trained to estimate the location of objects from global image features can improve detection performance by downgrading the score of false positives at unlikely locations. We experiment with DPM and EE-SVM on BigCH, as in sec. 3.4.3. As tab. 3.3 shows, ConF improves AP for both classes and both detectors (+2% for cars and +1% for horses). Instead, kNN does not bring any improvement. Fig. 3.8 shows example results.

3.4.5 ConF for class selection

Here we experiment on the ILSVRC2014 dataset, which has a large number of classes (200). In this scenario, we can use ConF to predict which classes are present in an image, and then run only detectors for those classes (sec. 3.3.4). We use the EE-SVM detector, but this time based on object proposals (Uijlings et al., 2013) and state-of-the-art R-CNN features (Girshick et al., 2014). These are produced by a CNN pre-trained for image classification (Krizhevsky et al., 2012; Jia, 2013) and then fine-tuned for object detection (Girshick et al., 2014). To train an E-SVM, we set $C = 10^{-4}$ and we mine hard negatives from 2000 random training images (using more did not bring any improvement). Both fine-tuning and E-SVM training are done on the $\texttt{val}_1$ set. We measure test performance on $\texttt{val}_2$, as AP averaged over the 200 classes (mAP).

Without any context, EE-SVM achieves an mAP of 16.3%. Selecting classes based on kNN retrieval sets improves performance by +3.3% (mAP 19.6%), while ConF delivers a larger improvement of +4.8% (mAP 21.1%). The improvements are due to removing false positives produced by detectors of classes unlikely to be present in the image. Interestingly, ConF selects less than 10 classes per image on average, and

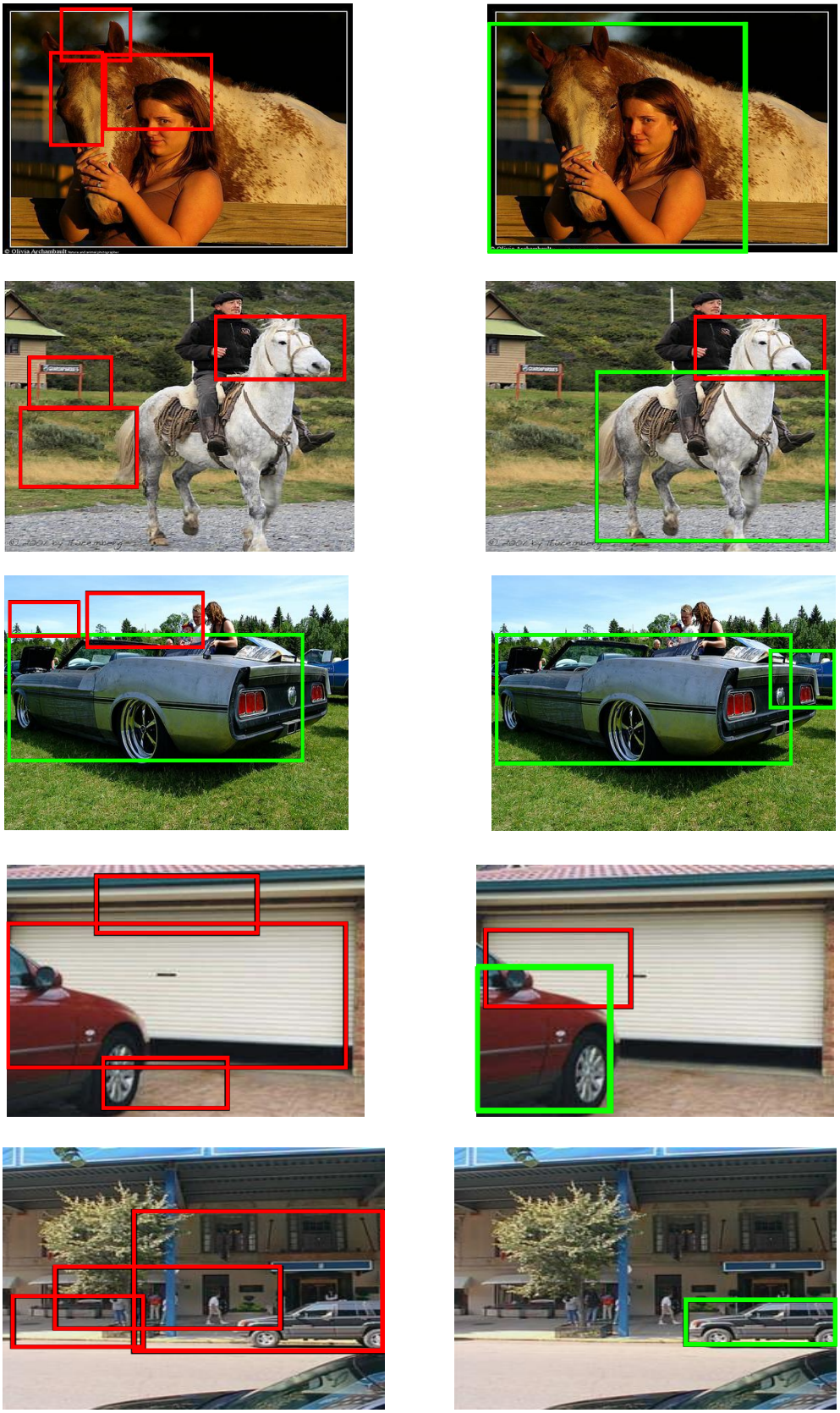


Figure 3.8: *Detections obtained before (left) and after (right) applying ConF as location model. Green bounding-boxes highlight correct detections, while red ones show false positives.*

therefore runs 20× fewer E-SVM detectors than the context-free baseline.

3.4.6 Computational and memory efficiency

ConF does not only offer better performance than kNN, but is also more memory and computationally efficient. kNN requires a number of distances computations linear in the number of training images, whereas ConF requires only a logarithmic number of thresholding operations. In practice this makes a big difference in runtime, e.g., ConF for component selection takes on average 0.5s per image vs 9.9s for kNN (on a 4-cores Intel Core i5 2.0GHz).

In terms of memory, kNN stores all feature vectors of all images in the training set. For cars in BigCH, this amounts to 1.7 GB. For each internal node ConF stores a threshold, a feature id and the ids of its children, amounting to 16 bytes. The leaves store the indices of the training images they contain, for a total of exactly the number of training images $\times 2$ bytes overall (per tree). For cars in BigCH, there are < 900 internal nodes on average per tree. As we store 750 trees per class, the grand total is only 27 MB (60× less than kNN).

3.5 Conclusions and outlook

In this chapter we presented Context Forest (ConF) (Modolo et al., 2015a), a technique to predict properties of objects in an image based on its global appearance. We trained ConF to predict three properties: aspects of appearance, location in the image, and class membership. We presented an extensive comparison to standard nearest-neighbour techniques for such context-based predictions. Results showed that ConF predicts object's properties from global image appearance more accurately than kNN, while being more memory efficient and much faster.

We also used ConF to speed-up and improve object class detection (with DPM and EE-SVM). We first showed how ConF can be employed to dynamically select a small subset of model components which is most relevant for a test image. Running only the selected components led to a speed-up (*2*× for DPM and *10*× for EE-SVM). Interestingly, in some cases we gained an improvement in accuracy as well, by not running the components that led to false detections. Then, we trained a second ConF to predict at which positions and scales objects are likely to appear in a given test image. By incorporating this location information in the detector score, we reduced the

false positive rate by removing detections in unlikely locations and improved detection performance (AP +1-2%). Finally, we trained a third ConF to predict which classes are present in each image and ran only the corresponding detectors. This greatly reduced the number of detectors run ($20\times$), and removed some false-positives, achieving an improvement of $+4.8\%$.

The technique presented in this chapter could be extended in several ways:

Combining all properties. In this chapter we trained ConF to predict three object properties and showed how object class detection can benefit from each of them. So far, we experimented with each property independently, but in the future we would like to explore them jointly. Each of the predicted properties captures unique information, complementary to the other two. For this reason, we believe that their combined usage is likely to lead to a higher improvement in detection performance.

Predicting more properties. ConF is a generic framework to predict properties of object classes and its applicability goes beyond the three properties predicted in this chapter. In the future it would be interesting to explore ConF for more properties. For example, an idea would be to train a new ConF to predict the number of object instances in a given image. This knowledge could be then used to suppress false-positive predictions, potentially leading to an improvement in object class detection performance. Moreover, by accurately designing the distance measure D of eq. (3.2), one can even predict more complex information like segmentation masks and 3D models of objects, if these are available for training images.

Predicting properties of objects out of context. One limitation of ConF is that it relies on objects appearing in a coherent context. If however an object is out of context (e.g., "a car inside a swimming pool" or "a car flying in the sky" (Choi et al., 2012)), ConF cannot correctly predict its properties, unless similar instances are present in the training set. Detecting properties of out-of-context objects is challenging for any context-based method because of contextual violation. While there is no easy way to extend ConF to learn about these objects, two ideas could be exploited. First, one could synthesize artificial images with out-of-context objects, and re-train ConF using these instances. In this way, ConF has the potential to learn to predict new image as belonging to this out-of-context category, and act accordingly. Second, one could model contextual relationship between predicted object instances, as a post-processing. If this

results to be incoherent, ConF's predictions could be discarded: all the components of all the object models are run and no location information is used as support.

Training object class detectors from retrieval sets. In this chapter we showed how to use ConF to select already trained object detector components relevant for a given image. A new idea would be to train object class detectors on the fly on the object instances in the retrieval set of ConF. As these are visually compact and very few, the training of the detector will likely be very fast. While this idea loses the speed-up brought by ConF to multi-component detectors, it enables the training of *any* object class detector. Moreover, as these detectors are optimal, according to ConF, to the given image, they have the potential to perform better than any pre-trained model.

ConF to predict properties of semantic parts. In this chapter we used ConF to predict properties of objects. It would be interesting to extend ConF to semantic parts of objects. For example, given an object bounding-box, ConF could predict the aspect of appearance of its semantic parts, where they are located and what parts are present. As object bounding-boxes provide stable coordinate frames common to all parts (chapter 5), predicting properties of semantic parts from them is probably easier than predicting properties of objects from whole images. For this reason, ConF has the potential to bring a substantial improvement to semantic part detection.

Chapter 4

Joint calibration of Ensemble of Exemplar SVMs

4.1 Introduction

The Ensemble of Exemplar SVMs (Malisiewicz, 2011) (EE-SVM, sec. 2.2) is a powerful non-parametric approach to object detection. It is widely used because it explicitly associates a training example to each object it detects in a test image, which enables transferring meta-data information. Furthermore, EE-SVM can also be used for other tasks, like discovering objects parts, scene classification, object classification, image parsing, image matching, automatic image annotation and 3D object detection.

An EE-SVM is a large collection of linear SVM classifiers, each trained from one positive example and many negative ones (an E-SVM). At test time each window is scored by all E-SVMs, and the highest score is assigned to the window. Because of this max operation, it is necessary to calibrate the E-SVMs to make their scores comparable. A common procedure is to calibrate each SVM independently, by fitting a logistic sigmoid to its output on a validation set (Malisiewicz, 2011). Such independent calibration, however, does not take into account that the final score is the max over many E-SVMs. Moreover, calibrating one E-SVM in isolation requires choosing which positive training samples it should score high and which ones it can afford to score low (fig. 4.1). Such a prior association of positive training samples to E-SVMs is arbitrary, as there is no predefined notion of how much and in which way a particular E-SVM should generalize. What truly matters is the interplay between all E-SVMs through the max operation.

In this chapter we present a *joint* calibration procedure that takes into account the

Figure 4.1: *Objects within a semantic category, such as train, have a diverse set of appearances due to variations in shape, pose, viewpoint, texture, and lighting. Given a collection of object examples, EE-SVM must determine which ones should be covered by each E-SVM.*

max operation. We calibrate all E-SVMs at the same time by optimizing their joint performance *after* the max. Our method finds a threshold for each E-SVM, so that (i) all positive windows are scored positively by at least one E-SVM, and (ii) the number of negative windows scored positively by any E-SVM is minimized. The first criterion ensures that there are no positive windows scored negatively after the max, while the second criterion minimizes the number of false positives.

We formalize these two criteria in a well-defined constrained optimization problem. The first requirement is formalised in its constraints, while the second comes in as a loss function to be minimized. Each threshold defines which training samples the respective E-SVM is scoring positively. By lowering a threshold we cover more positives and thereby satisfy more constraints, but we also include more negatives and therefore suffer a greater loss. Any positive sample can be potentially covered by any E-SVM, but at a different loss. This combinatorial nature of the problem makes it difficult to find the global optimum. We propose an efficient, globally optimal optimization technique. By exploiting the structure of the problem we are able to identify areas of the solution space that cannot contain the optimal solution and discard them early on. Our globally optimal algorithm is able to calibrate a few hundred E-SVMs quickly. In order to solve larger problems with thousands of E-SVMs, we present a simple modification of our exact algorithm to deliver high quality approximate solutions.

The rest of the chapter is organized as follows. We start by reviewing related work in sec. 4.2. Sec. 4.3 introduces the formulation of our optimization problem, while sec. 4.4 presents our algorithm for efficiently finding the global optimal solution as well as its approximation. We train EE-SVM on CNN descriptors (Girshick et al., 2014) and present experiments on 10 classes of the ILSVRC 2014 dataset (Russakovsky et al., 2015a) and on all 20 classes of PASCAL VOC 2007 (Everingham et al., 2010) in sec. 4.5. These experiments show that (i) our joint calibration procedure out-

performs standard independent sigmoid calibration (Malisiewicz, 2011) on the task of classifying windows as belonging to an object class or not; and (ii) this translates to better object detection performance. Finally, we conclude in sec. 4.6.

This work has been published at CVPR (Modolo et al., 2015b) and its code has been released at calvin.inf.ed.ac.uk/software/jointcalibration.

4.2 Related Work

In the machine learning literature, classifier calibration has been considered in the context of deriving probabilistic output for binary classifiers (Platt, 1999; Zadrozny and Elkan, 2001a,b; Niculescu-Mizil and Caruana, 2005) or multi-class classification (Zadrozny and Elkan, 2002; Gronat et al., 2013). Multi-class problems are often cast as a series of binary problems (e.g. 1-vs-all) and (Zadrozny and Elkan, 2002; Malisiewicz, 2011; Aubry et al., 2014b) showed that calibrating these binary classifier often leads to improved prediction.

The two most popular methods for calibrating binary classifiers are Platt scaling (Platt, 1999) and isotonic regression (Zadrozny and Elkan, 2001a). They both fit a monotonic function of the classifier score to the empirical label probability, obtaining an estimate of the conditional probability of a class label given the score. Platt scaling (Platt, 1999) uses a simple sigmoid function, while (Zadrozny and Elkan, 2001a) employs a more flexible isotonic regression. In computer vision, Platt scaling is the most popular calibration tool (Malisiewicz, 2011; Hoiem et al., 2008; Endres et al., 2013). We compare our approach to both methods in sec. 4.5.3.

All these works (Platt, 1999; Zadrozny and Elkan, 2001a,b; Niculescu-Mizil and Caruana, 2005) assume that the set of positive training samples for each classifier is fixed and given beforehand, even if small. In contrast, in the EE-SVM model, any positive sample can potentially be associated with any E-SVM. In the original EE-SVM paper (Malisiewicz, 2011) this was resolved in a greedy fashion, where each E-SVM was calibrated independently. The association of a positive sample to an E-SVM was resolved by comparing its uncalibrated E-SVM score to a fixed threshold. Instead, we calibrate E-SVMs *and* associate positive samples with them jointly over all positives and all E-SVMs. Our joint formulation (sec. 4.3) ensures that every positive is associated with at least one E-SVM, while the total number of false positives is minimized. As an additional benefit, this enables removing up to 25% of redundant E-SVMs that are not associated with any positives after the global optimum is found.

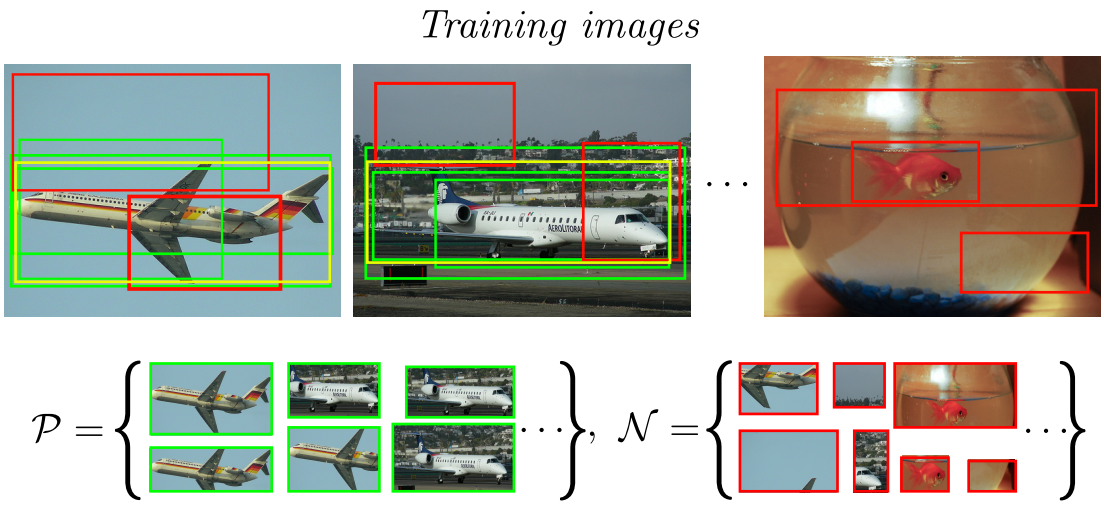


Figure 4.2: *Example of window proposals used in our calibration technique. $\mathcal{P}$ is the set of positive windows (□) and $\mathcal{N}$ is the set of negative windows (□) in the training set. Finally, (□) indicates E-SVMs ground-truth bounding-box. A window is positive if it has an intersection-over-union ≥ 0.5 with a ground-truth box.*

Two interesting exceptions to the classic EE-SVM calibration procedure (Malisiewicz, 2011) were presented recently (Gronat et al., 2013; Aubry et al., 2014a). Gronat et al. (Gronat et al., 2013) learns a per-location classifier for visual place recognition, while (Aubry et al., 2014a) learns exemplar-based 3D "chair" representations. Both works employ a calibration strategy based purely on negative samples, sidestepping the association of positive samples to E-SVMs. For completeness, we compare to (Aubry et al., 2014a) in sec. 4.5.3. All techniques reviewed above calibrate each E-SVM independently.

4.3 Joint calibration formulation

In many object detection pipelines (Dalal and Triggs, 2005a; Cinbis et al., 2013; Girshick et al., 2014) a single linear classifier $w \in R^d$ is applied to all K candidate windows $\{x\}_{i=1}^{K}$ in an image, where $x \in R^d$ is the window descriptor. The windows are then ranked according to the classifier score $w \cdot x$. An EE-SVM, instead, contains E classifiers: $\{w_j\}_{j=1}^{E}$. The score of a window x is defined as the highest score among all classifiers applied to it:

$$S(w) = \max_j (w_j \cdot x) \tag{4.1}$$

Our goal is to find a threshold θ_j for each E-SVM e_j such that (i) all positive win-

dows are scored positively by at least one E-SVM, and (ii) the number of negative windows scored positively by any E-SVM is minimized. A window x is scored positively by E-SVM e_j if $w_j \cdot x - \theta_j > 0$. We formalize these criteria in an optimization problem:

$$\begin{aligned} &\min_{\boldsymbol{\Theta}=\{\theta_j\}_{j=1}^{E}} \overbrace{\sum_{x\in\mathcal{N}} \mathbb{1}[\max_j(w_j \cdot x - \theta_j)]}^{\mathcal{L}(\boldsymbol{\Theta})} \\ &s.t.\ \mathbb{1}[\max_j(w_j \cdot x - \theta_j)] > 0, \forall x \in \mathcal{P} \end{aligned} \tag{4.2}$$

where $\mathbb{1}$ is the indicator function and $\mathcal{P}$ and $\mathcal{N}$ are the sets of positive and negative windows in the training set (fig. 4.2). We refer to the top term as the *loss function* $\mathcal{L}(\boldsymbol{\Theta})$ and the bottom terms as the *constraints*.

Calibration is performed by adjusting the thresholds $\boldsymbol{\Theta}$. Given a configuration of thresholds $\boldsymbol{\Theta} = [\theta_1, \theta_2, \ldots \theta_E]$, the loss $\mathcal{L}(\boldsymbol{\Theta})$ counts the number of negative windows scored positively after the max operation. Each constraint i ensures that a positive window x_i is scored positively by at least one E-SVM. We refer to a configuration $\boldsymbol{\Theta}$ satisfying all the constraints as a *feasible solution*.

4.4 Globally optimal and efficient solution

In this section we develop a computationally efficient algorithm to find the global optimal solution of eq. (4.2). We start in sec. 4.4.1 by analysing the space of all possible solutions of eq. (4.2). In sec. 4.4.2 we then introduce a data structure to represent this space, and finally in sec. 4.4.3 we present an efficient algorithm to search this data structure for the globally optimal solution.

4.4.1 Space of candidate thresholds

At first sight, eq. (4.2) appears to be a continuous optimization problem where each threshold can take any value in $\mathbb{R}$. However, since E-SVMs are evaluated only on a finite set of training windows, there exist an infinite set of *equivalent* thresholds leading to the same loss. For this reason, eq. (4.2) is in practice a discrete optimization problem.

Fig. 4.3 shows an example. Since each constraint in eq. (4.2) evaluates one positive window, an E-SVM needs at most $P+1$ thresholds to satisfy each of them (fig. 4.4),

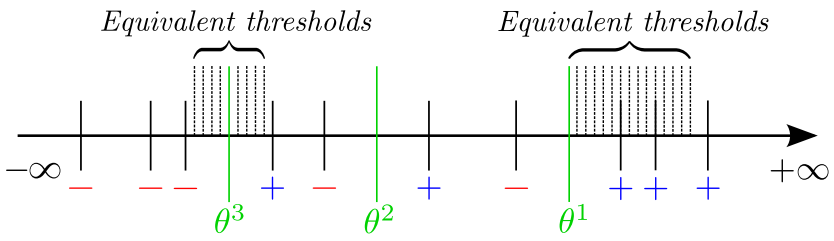


Figure 4.3: *Candidate thresholds, given the scores on positive (+) and negative (-) windows. The only thresholds worth considering according to (eq. 4.2) are the ones between a negative and positive window. Of all equivalent thresholds between two window scores, we consider only the mean of the two scores.*

where $P = |\mathcal{P}|$. Furthermore, considering a threshold between two positive samples is not necessary, because the loss only changes when new negative samples are scored positively. The only thresholds worth considering are those between the score of a negative sample and a positive (not the reverse, fig. 4.3). We denote the set of candidate thresholds for an E-SVM e_j as $[\theta_j^1, \theta_j^2, \dots \theta_j^{M_j}]$, where $1 \leq M_j \leq P+1$ and $\theta_j^a < \theta_j^b$, for $\forall a,b : 1 \leq a < b \leq M_j$. By construction, the lowest threshold $\theta_j^{M_j}$ satisfies all the constraints in eq. (4.2).

To conclude, the number of candidate thresholds for an individual E-SVM is relatively small (at most $P+1$), but the joint space of E-SVMs thresholds is nonetheless huge. In the worse case scenario (all E-SVMs have P candidate thresholds), there are P^E threshold configurations, many of which are not a feasible solution to eq. (4.2). In the next section we present a data structure to enumerate all these configurations and highlight the feasible solutions.

4.4.2 Exhaustive search tree

We represent the space of all possible solutions as a search tree (fig. 4.4).

Definition 1. (Search Tree) *Our search tree T is a perfect k-ary tree: a rooted tree where every internal node has exactly k children and all leaves are at the same depth h. Each node η contains a configuration $\boldsymbol{\Theta} = [\theta_1, \theta_2, \dots \theta_E]$ of thresholds.*

A configuration $\boldsymbol{\Theta}$ at node η is used to compute the loss $\mathcal{L}(\boldsymbol{\Theta})$ by counting how many negative windows are scored positively according to eq. (4.2). The root node has the configuration $\boldsymbol{\Theta} = [\theta_1^1, \theta_2^1, \dots \theta_E^1]$. These are the tightest thresholds for each E-SVM. We denote the set of false positives at a node η as ξ_η and $\mathcal{L}(\boldsymbol{\Theta}_\eta) = |\xi_\eta|$. Note how the root has $\xi = \emptyset$.

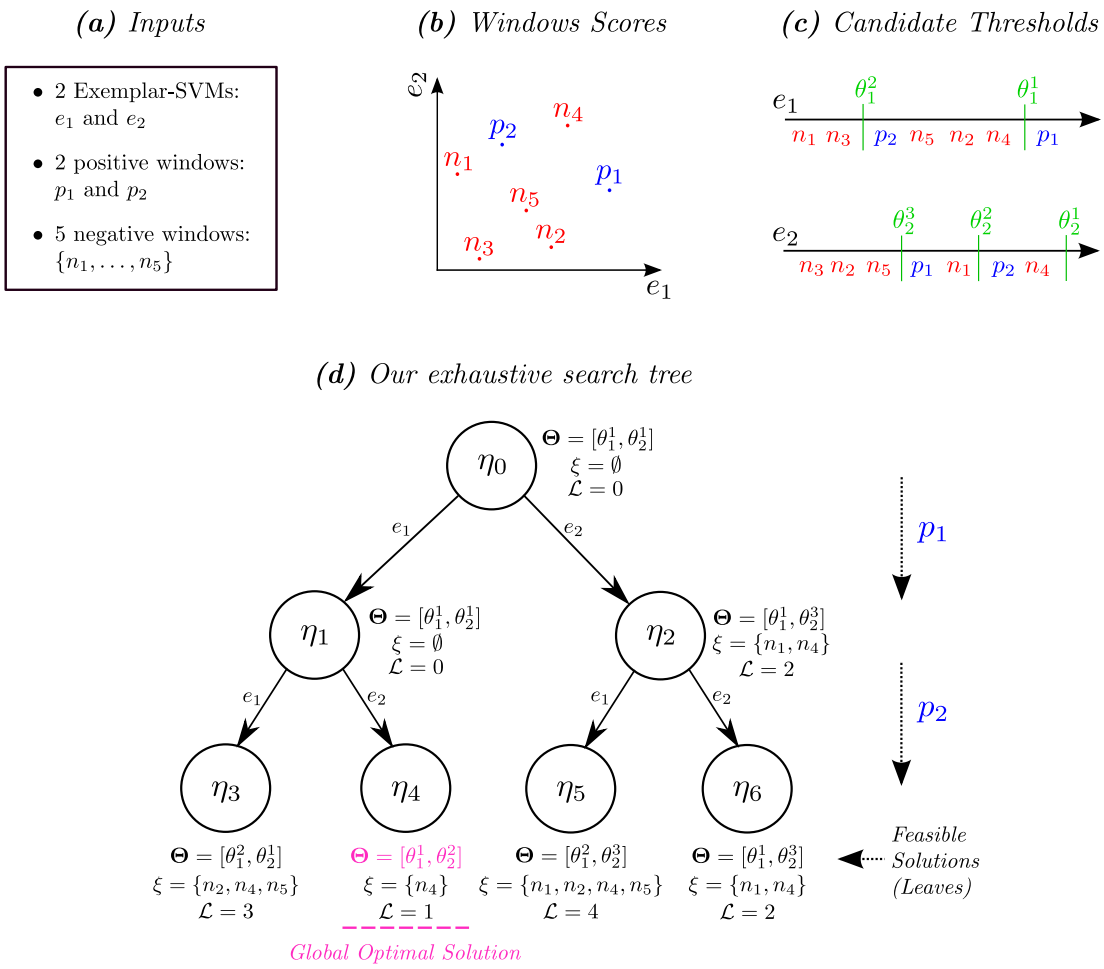


Figure 4.4: *Illustration of our joint calibration algorithm.* ***(a)*** *shows the input,* ***(b)*** *the window scores of the two E-SVMs e_1 and e_2 and* ***(c)*** *their candidate thresholds. These are $[\theta_1^1, \theta_1^2]$ and $[\theta_2^1, \theta_2^2, \theta_3^3]$, respectively. Finally,* ***(d)*** *shows the tree representing the space of all possible solutions. Note how the only feasible threshold configurations are those in the leaves.*

In our representation we have $h = P$ and $k = E$. Each level l of the tree corresponds to a positive window p_l, and each edge corresponds to an E-SVM e_j. An edge e_j indicates that e_j is responsible for p_l and it should score it positively, hence satisfying one constraint of eq. (4.2). Given an E-SVM e_j and its current threshold θ_j, the edge lowers the threshold so that $w_j \cdot x_l - \theta_j > 0$ (if this condition is already satisfied then the threshold does not change). Lowering the threshold *might* increase the loss, but not necessarily. Lowering the threshold will make E-SVM e_j score positively some negative windows, but these affect the loss only if they were not already scored positively by another E-SVM.

The deeper the level, the more constraints of eq. (4.2) are satisfied. By construction, the configuration of thresholds at a leaf satisfy all constraints, and the set of all leaves

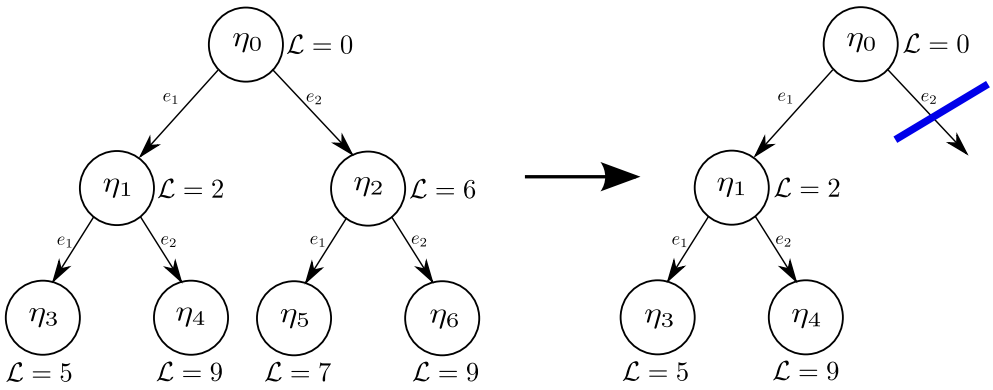


Figure 4.5: *Example of pruning by bound. Since* $\mathcal{L}(\mathbf{\Theta}_{\eta_3}) = 5 < \mathcal{L}(\mathbf{\Theta}_{\eta_2}) = 6$*, the subtree rooted at* η_2 *cannot contain an optimal solution.*

represent the set of all feasible solutions. Also note that the number of false positives always increases (or remains the same) along a path from the root to a leaf: given a node η and any child η', we have that $\mathcal{L}(\mathbf{\Theta}_\eta) \leq \mathcal{L}(\mathbf{\Theta}_{\eta'})$.

4.4.3 Efficient search

In this section we find the global optimal solution to eq. (4.2) by searching the tree. Exhaustive search is computationally prohibitive even for small problems with few E-SVMs and positive windows, as the total number of nodes in our tree is $(E^{P+1} - 1)/(E-1)$ (with E the number of E-SVMs and P the number of positive windows).

The key to our efficient algorithm is to prune the tree by iteratively removing subtrees which cannot contain the global optimal solution. In the following paragraphs we present several observations that enable a drastic reduction in the space of solutions to consider. The last paragraph of this section presents the actual search algorithm.

Observation 1. (Pruning by bound). *If* η *is a leaf and* η' *is a node not on the path from the root to* η*, and* $\mathcal{L}(\mathbf{\Theta}_\eta) \leq \mathcal{L}(\mathbf{\Theta}_{\eta'})$*, then the subtree rooted at* η' *cannot contain a better solution than* η *and can be discarded.*

The key intuition is that the loss can only increase with depth. The observation leads to two cases. First, if $\mathcal{L}(\mathbf{\Theta}_\eta) < \mathcal{L}(\mathbf{\Theta}_{\eta'})$, then η' cannot lead to an optimal solution since its loss is already higher than an already found feasible solution. Second, if $\mathcal{L}(\mathbf{\Theta}_\eta) = \mathcal{L}(\mathbf{\Theta}_{\eta'})$, η' could lead to an optimal solution, but it would be equivalent to the one in η. In both cases we can discard the subtree rooted at η', as we are interested in finding only one optimal solution.

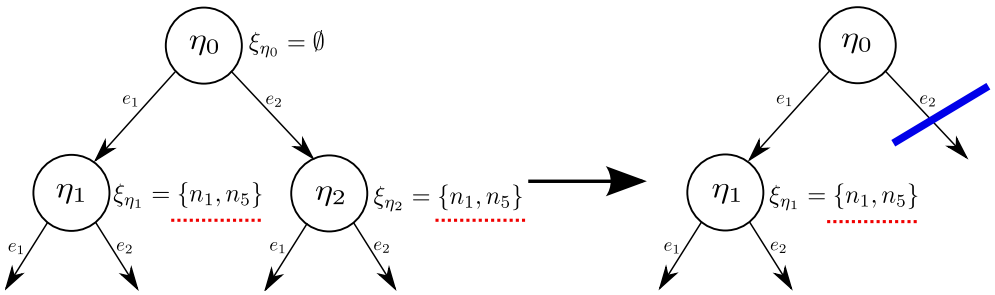


Figure 4.6: *Example of pruning by equivalence. Since the two nodes* η_1 *and* η_2 *have the same parent and* $\xi_{\eta_1} = \xi_{\eta_2}$*, they are equivalent and only one subtree needs be searched.*

Consider the example in fig. 4.5. Since $\mathcal{L}(\mathbf{\Theta}_{\eta_3}) = 5 < \mathcal{L}(\mathbf{\Theta}_{\eta_2}) = 6$, the subtree rooted at η_2 cannot contain an optimal solution: any solution in it has a loss ≥ 6, which is higher than the feasible solution in η_3.

Observation 2. (Pruning by equivalence). *If two nodes* η *and* η' *have the same parent and* $\xi_\eta = \xi_{\eta'}$*, then they are equivalent and only one subtree needs be searched.*

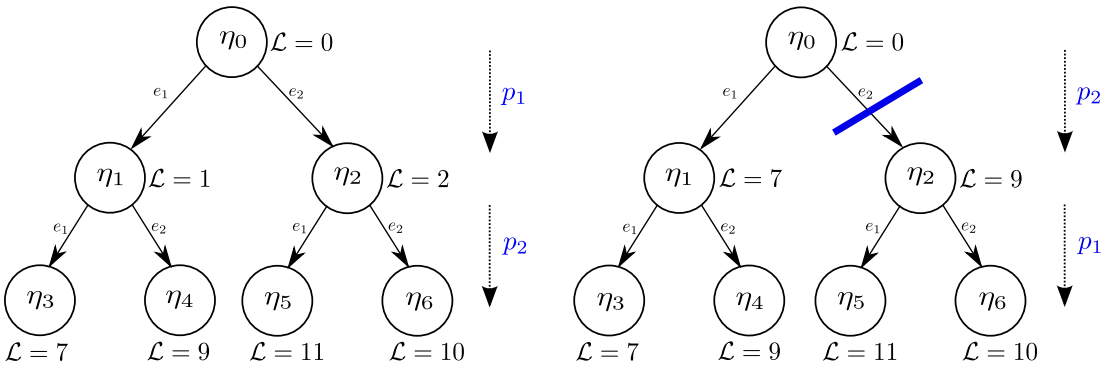
The key intuition is that the loss in eq. (4.2) only increases when *new* negative windows are scored positively. Consider the example in fig. 4.6, where $\xi_{\eta_1} = \xi_{\eta_2}$ and η_1 and η_2 have the same parent (i.e., they satisfy the same constraints of eq. 4.2). η_1 has $\mathbf{\Theta} = [\theta_1^2, \theta_2^1]$ because the edge from η_0 to η_1 adjusted the threshold of e_1. Note, however, that this configuration can be changed into $[\theta_1^2, \theta_2^2]$ without increasing the loss, as they are equivalent solutions. Because of this equivalence, both subtrees lead to equivalent feasible solutions and only one of them needs be searched.

Observation 3. (Reducing tree depth). *Given the root configuration* $\mathbf{\Theta} = [\theta_1^1, \theta_2^1, \ldots \theta_E^1]$*, there might exist some* $x \in \mathcal{P} : \max_j (w_j \cdot x - \theta_j^1) > 0$*. Since* $\mathbf{\Theta}$ *already satisfies the constraint for these positives at zero cost, these can be eliminated right away, reducing the depth of the tree.*

Consider the example in fig. 4.4. Initially, $\mathbf{\Theta} = [\theta_1^1, \theta_2^1]$. This configuration already scores p_1 positively. Whatever optimal solution $\mathbf{\Theta}$ the tree retrieves, p_1 will always be scored positively by at least E-SVM e_1. Hence, we can eliminate it from the tree to reduce its depth.

Observation 4. (Order of positive windows). *By sorting the positive windows by decreasing difficulty, pruning by bound can discard larger subtrees.*

The difficulty of a positive window x is measured as $\min_j \delta(e_j, x)$, where $\delta(e_j, x)$ counts how many false positives e_j produces by scoring the positive sample $x \in \mathcal{P}$ positively.



(a) No pruning (b) Pruning by bound

Figure 4.7: *Efficient pruning by bound can be achieved by sorting the positive windows by decreasing difficulty.*

The key intuition is that it is better to prune subtrees rooted at the higher levels of the tree, as they contain more nodes. This can be achieved by placing difficult positive windows up in the higher levels. Some positive windows lead to constraints intrinsically more difficult to satisfy than others, as any E-SVM asked to satisfy it would score positively many negatives as well, and hence incur a large loss. By sorting the tree levels according to the difficulty of positive windows, it is likely that the loss of many high-level configurations is higher than a previously found feasible solution, and therefore can be pruned (observation 1).

Consider the example in fig. 4.7. Tree (a) evaluates first p_1 and then p_2, while (b) does the opposite. Tree (a) cannot be pruned by observation 1, but tree (b) can.

In sec. 4.5.5 we evaluate experimentally how effective the above pruning techniques are on various real EE-SVM calibration problems.

4.4.3.1 Search algorithm

We present here a depth-first search algorithm to efficiently find the global optimal solution, based on the above observations (Algo. 1).

The algorithm works as follows. The initial configuration of thresholds $\boldsymbol{\Theta}$ is the one from the root node *(line 1)*. As preprocessing, the algorithm starts by reducing the tree depth using observation 3 *(line 2)* and re-ordering the positive windows using observation 4 *(line 3)*. In the first step, it does a depth-first search until it reaches a leaf η and finds a first feasible solution $\boldsymbol{\Theta}$ *(line 4)*. During this traversal, when going down a level, the algorithm always chooses the edge leading to the smallest loss. Next, the algorithm continues by going up *(line 6)* and down *(line 10)* the tree. When visiting a node, the algorithm tries to prune as many children subtrees as it can using observation

Algorithm 1 Our Efficient Search Algorithm

Input: search tree T

Output: global optimal $\boldsymbol{\Theta}$

1: $\boldsymbol{\Theta} \leftarrow [\theta_1^1, \theta_2^1, \ldots \theta_E^1]$
2: $T \leftarrow$ REDUCE TREE DEPTH(T, $\boldsymbol{\Theta}$)
3: $T \leftarrow$ REORDER POSITIVE WINDOWS(T, $\boldsymbol{\Theta}$)
4: η, $\boldsymbol{\Theta}$, $\mathcal{L} \leftarrow$ DEPTH FIRST SEARCH(T, $\boldsymbol{\Theta}$)
5: **while** T not fully searched **do**
6: $\quad \eta \leftarrow$ GO UP ONE LEVEL (T, η)
7: $\quad$ **while** $\neg$ ISLEAF(η) $\wedge$ $\neg$ HASCHILD(η) **do**
8: $\quad\quad T \leftarrow$ PRUNE BY BOUND $(T, \eta, \mathcal{L})$
9: $\quad\quad T \leftarrow$ PRUNE BY EQUIVALENCE $(T, \eta, \boldsymbol{\Theta})$
10: $\quad\quad \eta \leftarrow$ GO DOWN ONE LEVEL (T, η)
11: $\quad$ **end while**
12: $\quad$ **if** ISLEAF(η) **then**
13: $\quad\quad \boldsymbol{\Theta}$, $\mathcal{L} \leftarrow$ GET SOLUTION(η)
14: $\quad$ **end if**
15: **end while**
16: return $\boldsymbol{\Theta}$

1 *(line 8)* and observation 2 *(line 9)*. When the algorithm reaches a leaf then this must contain a better solution than the current one $\boldsymbol{\Theta}$ (in terms of the loss in eq. 4.2). Hence, it updates $\boldsymbol{\Theta}$ *(line 13)*. The algorithm continues until all nodes have been visited or pruned. The final $\boldsymbol{\Theta}$ is the global optimum of eq. (4.2).

4.4.4 Approximate search

Above we presented an efficient algorithm that guarantees global optimality. If we relax the global optimality requirement, we can improve efficiency even further. Our method follows a depth-first search and as soon as it reaches a leaf, then it finds a feasible solution. This happens periodically during the execution of the algorithm, as better and better leaves are found while the tree is searched. This behavior makes our method an *any-time* algorithm (Zilberstein, 1996; Gogate and Dechter, 2004). After a short period required to reach the first leaf, we can terminate it at any time and it

will return the best feasible solution it has found so far (although not necessarily the globally optimal one). This simple observation enables us to employ our method, essentially unchanged, to find approximate solutions as well. This becomes extremely important with large training sets. To capture the full variability of an object class, tens of thousands E-SVMs has to be trained. While it is impractical to find a global optimal solution for all these models, it is easy to find a feasible solution using our approximate search.

4.5 Experiments

4.5.1 Datasets

We present experiments on ILSVRC2014 (Russakovsky et al., 2015a) (sec. 4.5.3, 4.5.4) and PASCAL VOC 2007 (Everingham et al., 2010) (sec. 4.5.4). ILSVRC2014 contains 200 classes annotated by bounding-boxes. In our experiments we randomly sampled 10 classes: *airplane, bagel, baseball, bear, butterfly, koala, ladle, printer, sheep* and *violin*. Following (Girshick et al., 2014) we consider three disjoint subsets of the data: `train`, $\texttt{val}_1$ and $\texttt{val}_2$. Since annotations for the test set are not released, we measure performance on $\texttt{val}_2$. We use $\texttt{val}_1$ and `train` for training. In total, these sets contain $>$80k images.

PASCAL VOC 2007 contains 20 classes annotated by bounding-boxes. In our experiments we evaluate on all 20 classes. We use the subset `trainval` for training and we measure performance on `test`. In total, these sets contain about 10k images.

4.5.2 Settings

Object proposals and features. We generate class-independent object proposals using (Uijlings et al., 2013). Given an image, this produces a small set of a few thousand windows likely to cover all objects. We then extract CNN descriptors of 4096 dimensions for these proposals, as in (Girshick et al., 2014) (sec. 2.3). These descriptors are the output of a convolutional neural network (CNN) initially trained for image classification (Krizhevsky et al., 2012; Jia, 2013) and then fine-tuned for object detection (Girshick et al., 2014) (on $\texttt{val}_1$ of ILSVRC2014, or on `trainval` of PASCAL VOC 2007).

EE-SVM. We learn a separate window classifier e for each instance of an object in the training set. We set $C = 10^{-4}$ and we mine hard negatives from 2000 random training

images. In our experiments we observed that mining more images did not bring a significant improvement.

Calibration data. For each class we define $\mathcal{P}$ as the set of all positive training windows. A window is considered positive if it has intersection-over-union ≥ 0.5 with a ground-truth bounding-box of that class. $\mathcal{N}$ contains negative windows that overlap ≤ 0.2. All calibration methods below are trained from this data.

Independent sigmoid calibration. As a baseline we compare against the standard technique of Malisiewicz (2011). It operates in two steps. In step (1) it runs each E-SVM detector separately on a validation set, applies non-maximum suppression, and then eliminates all detections scoring below the -1 margin. All remaining detections are considered positives if they belong to $\mathcal{P}$, or negatives if they belong to $\mathcal{N}$. Note how this arbitrarily defines which positive training samples to associate with a certain E-SVM. In step (2), it then fits a logistic sigmoid to these data samples.

Our joint calibration. Our joint calibration also operates in two steps. In step (1), instead of arbitrarily defining the positive training samples, our technique use the thresholds found by our algorithm (sec. 4.4) to associate positive samples to E-SVMs, which is the core underlying problem at the heart of such calibration. More specifically, for an E-SVM e_j, we consider as positives all windows $x \in \mathcal{P} : w_j \cdot x > \theta_j$, and as negatives all windows in $\mathcal{N}$. Step (2) of our procedure is then the same as in (Malisiewicz, 2011), but thanks to these optimal assignments, we fit better sigmoids.

We experimentally evaluate performance after each step. We refer to the output of step (1) as *joint calibration*, and to the output of step (2) as *joint calibration with sigmoid*. As step (1) fits thresholds, it results in binary classification of test windows, while step (2) produces a continuous score which can be used for later processing stages (e.g. non-maximum suppression for object detection).

Other independent calibration techniques. For completeness, we also compare against two independent calibration techniques not commonly used for EE-SVM: isotonic regression (Zadrozny and Elkan, 2002) and the recent affine calibration approach (Aubry et al., 2014a). *Isotonic regression* fits a piecewise-constant non-decreasing function to the output of each E-SVM. We used the code of (Duembge, 2000) to train the function parameters on $\mathcal{P}$ and $\mathcal{N}$. *Affine calibration* fits an affine transformation to

the output of each E-SVM. As in (Aubry et al., 2014a), we train the affine parameters on 200k randomly sampled negative windows from $\mathcal{N}$.

Single Linear-SVM (R-CNN). Finally, we provide results for the popular R-CNN object detection model (Girshick et al., 2014) (sec. 2.3). The sole purpose is to compare performance to EE-SVM on CNN features, as previous EE-SVM works typically use weaker HOG features (Malisiewicz, 2011; Shrivastava et al., 2011; Singh et al., 2012; Aytar and Zisserman, 2012; Endres et al., 2013; Dong et al., 2013; Tighe and Lazebnik, 2013; Juneja et al., 2013; Vezhnevets and Ferrari, 2014). However, as it consists of a single linear SVM per class, R-CNN cannot associate training examples to objects detected in test images. As a result, it is not suitable for annotation transfer. We trained the model using the code and parameters of (Girshick et al., 2014). Note how this uses the same object proposals and features as our EE-SVM models.

4.5.3 Globally optimal joint calibration

We evaluate our globally optimal joint calibration technique on ILSVRC2014 (Russakovsky et al., 2015a). We train E-SVMs on $\texttt{val}_1$ for the 10 classes listed in sec. 4.5.1. Each class has between 30 and 140 E-SVMs and between 500 and 3600 positive windows $\mathcal{P}$. We evaluate two tasks: window classification and object detection.

Window classification

For this experiment, we use all positive windows ($IoU \geq 0.5$) in the test set $\texttt{val}_2$ and 1 million randomly sampled negative ones ($IoU < 0.5$). We evaluate window classification in terms of two measures: false positives at test recall, and average precision.

False positives at test recall. This measure counts how many false positives are produced on the test set, at the recall point produced by the ensemble of E-SVMs calibrated by our method. Note that this is exactly what our calibration procedure optimizes for. Given the thresholds $\boldsymbol{\Theta}$ output by our algorithm (sec. 4.4), we compute recall as the percentage of positive windows scored positively by the ensemble on the test set (top row of table 4.1). Interestingly, the thresholds generalize well to test data and lead to high recall on almost all classes.

We compare several methods at this recall point. The main four are EE-SVM with no calibration, EE-SVM with independent sigmoid calibration (Malisiewicz, 2011),

ILSVRC 2014 - trained on $\texttt{val}_1$	Airplane	Bagel	Baseball	Bear	Butterfly	Koala	Ladle	Printer	Sheep	Violin	mean
Recall	*94.1*	*90.1*	*97.9*	*93.1*	*97.5*	*96.6*	*79.5*	*88.5*	*98.9*	*85.3*	*92.2*
EE-SVM no calibration	180124	171966	499056	33664	163727	80145	250602	221117	308458	37519	195k
EE-SVM indep. sigmoid calibration (Malisiewicz, 2011)	119552	42165	234734	77099	53986	13017	56616	88390	86507	33266	80k
EE-SVM joint calibration (this work)	65182	28460	180658	22694	53927	10746	87513	36573	55923	32570	57k
EE-SVM joint calibration w/ sigmoid (this work)	64996	28140	168129	22876	53867	10722	87329	35803	50064	33899	**55k**
EE-SVM indep. isotonic regression (Zadrozny and Elkan, 2002)	54173	23625	268302	16424	43580	13507	60285	55129	76997	13814	63k
EE-SVM indep. affine calibration (Aubry et al., 2014a)	51905	24507	224003	18978	37483	9947	67288	86757	110909	18218	65k
Single Linear-SVM (R-CNN) (Girshick et al., 2014)	102676	335122	711185	109480	63849	305931	322332	469777	979131	121050	341k

Table 4.1: ***Window classification - False positives at test recall.*** *Results on a subset of ILSVRC 2014 $\texttt{val}_2$ (all positive windows and one million randomly sampled negative ones). We use the optimal thresholds found by our algorithm (sec. 4.4) to compute recall on $\texttt{val}_2$. This is the percentage of positive windows scored positively by our jointly calibrated EE-SVM. The table entries show the number of false positives produced in order to reach that recall level. Each row corresponds to a different method (ours are marked "joint calibration").*

our joint calibration fitting thresholds and our joint calibration with sigmoid (table 4.1, rows 2-5). As expected, EE-SVM with no calibration performs very poorly and some form of calibration is necessary. Our joint calibration method considerably outperforms independent calibration, and the version with sigmoid brings another small boost in performance. These results demonstrate the benefit of our joint calibration, that takes into account the max operation of the EE-SVM. Given these results, we omit EE-SVM with no calibration from further analysis. Table 4.1 also presents results for isotonic regression, affine calibration and R-CNN. On average, our joint calibration outperforms all these methods, albeit by a smaller margin.

Average precision. In table 4.2 we compare techniques in terms of average precision. As this measure requires a continuous score of test windows, we only consider our joint calibration with sigmoid. Joint calibration outperforms independent sigmoid calibration on all classes, and improves mAP by 2.3%. This further highlights the benefits of joint calibration, in a scenario that is not exactly what it was optimized for. Table 4.2 also presents results for isotonic regression, affine calibration and R-CNN. On average, joint calibration performs better than all these methods, abeit by a modest margin (about +1% mAP).

ILSVRC 2014 - trained on $\texttt{Val}_1$	Airplane	Bagel	Baseball	Bear	Butterfly	Koala	Ladle	Printer	Sheep	Violin	mAP
EE-SVM indep. sigmoid calibration (Malisiewicz, 2011)	42.8	39.7	63.3	58.7	60.8	58.2	4.5	29.0	49.5	20.3	42.7
EE-SVM joint calibration w/ sigmoid (this work)	43.3	40.1	66.5	60.6	63.9	61.1	5.0	31.6	55.1	22.9	**45.0**
EE-SVM indep. isotonic regression (Zadrozny and Elkan, 2002)	44.6	42.4	61.4	59.3	63.8	63.2	5.7	22.5	50.9	23.2	43.7
EE-SVM Indep. affine calibration (Aubry et al., 2014a)	45.6	42.0	64.1	59.2	62.6	62.2	6.3	22.9	50.4	25.9	44.1
Single Linear-SVM (R-CNN) (Girshick et al., 2014)	47.9	36.9	65.0	60.9	66.7	63.4	5.4	24.4	50.6	19.1	44.0

Table 4.2: ***Window classification - Average precision.*** *Results on a subset of ILSVRC 2014* $\texttt{Val}_2$ *(same data as table 4.1).*

ILSVRC 2014 - trained on $\texttt{Val}_1$	Airplane	Bagel	Baseball	Bear	Butterfly	Koala	Ladle	Printer	Sheep	Violin	mAP
EE-SVM indep. sigmoid calibration (Malisiewicz, 2011)	43.3	11.9	27.2	45.2	51.1	46.3	0.6	8.4	31.4	7.4	27.3
EE-SVM joint calibration w/ sigmoid (this work)	46.2	10.1	41.3	44.7	66.8	41.4	1.0	10.8	34.3	9.5	30.6
EE-SVM indep. isotonic regression (Zadrozny and Elkan, 2002)	34.9	13.0	31.4	36.1	59.2	48.6	0.6	13.0	31.0	3.8	27.2
EE-SVM indep. affine calibration (Aubry et al., 2014a)	45.8	10.3	41.0	44.1	66.1	39.5	0.8	11.4	35.9	9.6	30.5
Single Linear-SVM (R-CNN) (Girshick et al., 2014)	49.0	17.4	45.4	53.3	69.6	61.4	2.8	18.9	41.5	11.0	37.0

Table 4.3: ***Object detection - Average precision.*** *Results on ILSVRC 2014* $\texttt{Val}_2$.

Object detection

We evaluate our joint calibration method against independent sigmoid calibration on the task of object detection. Note how this task adds a layer of non-maximum suppression (NMS) to the pipeline. As our calibration procedure does not take into account NMS, it is not obvious that the benefit seen so far on window classification will carry over to object detection. As table 4.3 shows, joint calibration outperforms independent sigmoid calibration on this task as well (+3.3% mAP). Joint calibration performs equally or better on all classes but *koala*. For some classes the improvement is substantial: +14% AP on *baseball* and +14.7% AP on *butterfly*.

Furthermore, table 4.3 also presents results for isotonic regression, affine calibration and R-CNN. Isotonic regression performs comparably to independent sigmoid calibration, whereas affine calibration delivers about the same mAP as our joint calibration. Interestingly, R-CNN does considerably better than all other methods, including our EE-SVM with joint calibration. This is somewhat surprising, as EE-SVM was shown to be much better than a single linear SVM on HOG features (Malisiewicz,

ILSVRC 2014 - trained on $\texttt{Val}_1$+train	mean
Recall	85.2
EE-SVM no calibration	137k
EE-SVM indep. sigmoid calibration (Malisiewicz, 2011)	107k
EE-SVM joint calibration	82k
EE-SVM joint calibration w/ sigmoid	**54k**

Table 4.4: ***Window classification - False positives at test recall.*** *Results on a subset of ILSVRC 2014* $\texttt{Val}_2$ *(mean over 10 classes). We used all positive windows and 2 million randomly sampled negative ones.*

2011). We attribute this phenomenon to the CNN features, which are more easily linearly separable (Donahue et al., 2013; Girshick et al., 2014; Razavian et al., 2014; Chatfield et al., 2014). Besides, note that despite the high performance, R-CNN lacks the crucial ability of EE-SVM to associate training exemplars to objects detected in test images, and it is therefore not suitable for annotation transfer.

4.5.4 Approximate joint calibration

In this section we evaluate our approximate joint calibration technique (sec. 4.4.4). By relaxing global optimality, we can find a feasible solution even for large problems.

4.5.4.1 ILSVRC2014

We experiment here by training and calibrating EE-SVM on the union of $\texttt{val}_1$ and train. This results in a large number of E-SVMs. Each class has between 640 and 2000 E-SVMs, and between 3000 and 13000 positive windows $\mathcal{P}$. We report results on the task of window classification averaged over the 10 classes. Note that these results cannot be compared to the ones in tables 4.1 and 4.2 because here we have larger training and test sets.

False positives at test recall. As table 4.4 shows, our approximate joint calibration procedure still achieves high recall while returning much fewer false positives than no calibration and independent sigmoid calibration. When adding a sigmoid our calibration improves even further by a good margin. This shows that our method provides an excellent association between positive windows and E-SVMs.

ILSVRC 2014 - trained on $\texttt{Val}_1$+train	mAP
EE-SVM indep. sigmoid calibration (Malisiewicz, 2011)	39.7
EE-SVM joint calibration w/ sigmoid	**42.9**

Table 4.5: ***Window classification - Average precision.*** *Results on a subset of ILSVRC 2014* $\texttt{Val}_2$ *(mean over 10 classes, same data as table 4.4).*

PASCAL VOC 2007 test	Feature	Calibration	mAP
EE-SVM	HOG	independent	19.8
	CNN	independent	40.8
	CNN	joint	42.7

Table 4.6: ***Object detection - Average precision.*** *Results on PASCAL VOC 2007* `test` *(mean over 20 classes). EE-SVM HOG results are from (Malisiewicz, 2011).*

Average precision. Results are presented in table 4.5. Joint calibration improves over independent sigmoid calibration by +3.2% mAP.

4.5.4.2 PASCAL VOC 2007

In order to compare against the original EE-SVM of Malisiewicz (2011), we experiment here on the PASCAL VOC 2007 dataset. We train and calibrate EE-SVM on the `trainval` subset and evaluate them on `test`. We report results on object detection in terms of mAP over the 20 classes (table 4.6). We compare traditional EE-SVM on HOG features (19.8 mAP, as reported by Malisiewicz (2011)), independently calibrated EE-SVM on CNNs (40.8 mAP), and our joint calibration on the same features (42.7 mAP). These results highlight two points: (1) joint calibration improves over independent calibration by +2% mAP, confirming what observed on ILSVRC2014; (2) CNN features bring a huge improvement over HOG to EE-SVM models (doubling mAP in this case). This confirms recent findings (Girshick et al., 2014) about the benefits of CNN features for object detection.

4.5.5 Pruning statistics and runtimes

Pruning statistics. Here we experimentally evaluate how effective our pruning techniques of sec. 4.4.3 are. Observation 3 (*line 2*, Algo. 1) reduces the depth of the tree by 20%, on average. Observation 4 (*line 3*) improves pruning by bound immensely. In a

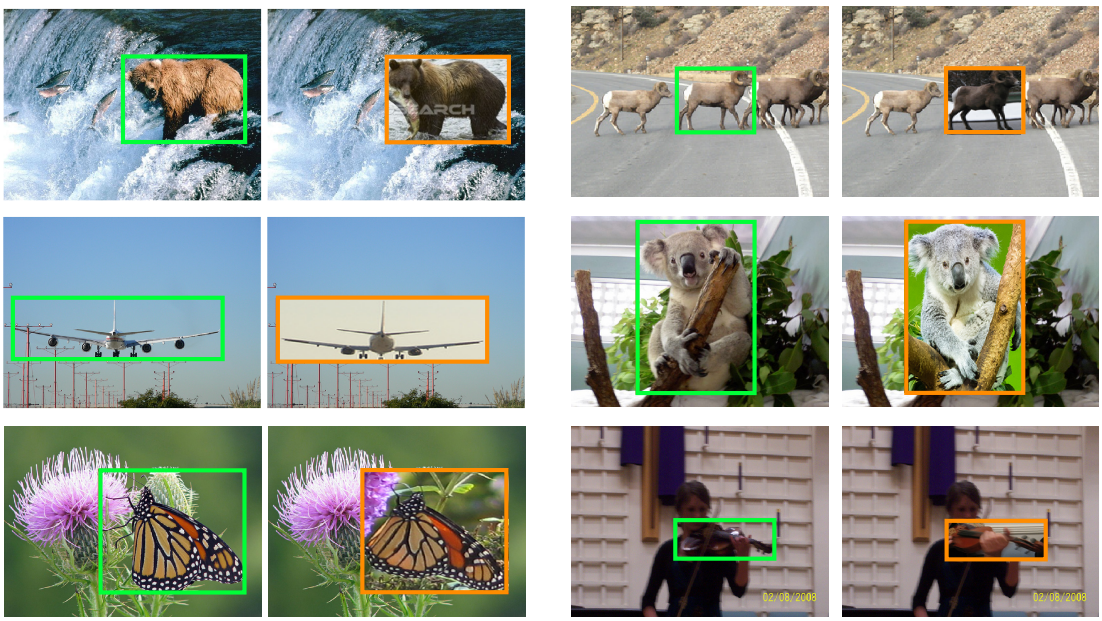


Figure 4.8: ***Association between detected objects and training exemplars.*** *Our globally optimal joint calibration is good at transferring annotations from exemplars onto test windows. In these figure we show detections (green) and their associated training exemplar superimposed on them (orange).*

small problem with 100 E-SVMs we tried ordering the positive windows $\mathcal{P}$ randomly. The algorithm took two hours to find the global solution. On the other hand, when sorting $\mathcal{P}$ according to observation 4, the algorithm found the same solution in about 2 minutes. Observations 1 and 2 also bring a substantial speed-up. After finding a first feasible solution (*line 1*), the algorithm (*lines 8,9*) prunes 40% of the nodes it visits on problems with 100 E-SVMs, and 70% on problems with 1000 E-SVMs.

Runtime. We measure runtimes on a 4-core Intel Core i5 2.0GHz. Exhaustive search is extremely inefficient and takes 15h to find the globally optimal solution for a tiny problem with 15 E-SVMs and 50 positive windows. Our efficient and exact algorithm (sec. 4.4.3) finds the same solution in just a few seconds. This algorithm scales up to problems with about 200 E-SVMs and 4k positive windows in reasonable time (a few hours). For larger problems we rely on our approximate search algorithm (sec. 4.4.4). While we let it run for several hours, in most cases the loss stops decreasing significantly after just a few minutes.

4.6 Conclusion and outlook

In this chapter we presented a method for calibrating the Ensemble of Exemplar SVMs model. While the standard approach calibrates each SVM independently, our method optimizes their joint performance as an ensemble. We formulated joint calibration as a constrained optimization problem and devised an efficient optimization algorithm to find its global optimum. In order to make the optimization computationally feasible, the algorithm dynamically discards parts of the solution space that cannot contain the optimum, by exploiting four observations about the structure of the problem.

We presented experiments on 10 classes from the ILSVRC 2014 dataset and 20 from PASCAL VOC 2007. Our joint calibration procedure outperforms the classic independent sigmoid calibration by a considerable margin on the task of classifying windows as belonging to an object class or not. On object detection, this better window classifier leads to an improvement of about 3% mAP.

The joint calibration technique presented in this chapter could be improved and extended in several ways:

Faster approximate calibration. To scale to large datasets, we showed how to relax the global optimality of our joint calibration procedure to attain good approximate solutions. Our approximation is based on a simple early-stopping heuristic: after finding a feasible solution, we let the process run for a few hours and retain the best solution found up to that point. The reason we let the calibration run for so long is that we observed that different classes require different amount of time. For most of the classes the loss stopped decreasing significantly already after few minutes, but for others the calibration needed a bit longer. This is probably related to how accurately the E-SVMs can score the positive training windows positively and the negative training windows negatively. In the future we would like to study this matter quantitatively and try to understand under what conditions the calibration is slow and under what conditions it is fast. We believe that by analyzing and experimenting with different rankings of E-SVMs scores we can learn to predict this behaviour. This would be an important addition to our joint calibration, as it would enable different early-stopping criteria based on the current data, not wasting important time and resources.

Increasing the robustness of our optimization problem. One theoretical weakness

of our optimization problem (eq. 4.2) is that it is constrained such that all positive windows have to be scored positively by at least one Exemplar-SVM. This condition can make the method sensitive to noise in the training data. For example, there may exist an outlier positive window scored very lowly by all E-SVMs, yet our calibration has to lower the threshold of one E-SVM to cover it. This will result in the inclusion of many negative windows, leading to a greater loss and potentially even worse detection performance. A potential solution would be to relax this constraint so that the calibration covers most of the positive windows, but not necessarily all. An idea would be to change the constraint to cover at most 80-90% of the positive windows of each ground-truth object bounding-box (fig. 4.2). Another idea would be to weight each positive window based on its importance (e.g., its IoU score with its ground-truth-bounding box) and score positively only the most important ones in the training set.

Joint calibration of E-SVMs and R-CNN. In sec. 4.5 we observed that the R-CNN object class detector performs better than EE-SVM. We visually inspected their detections and observed that, while R-CNN is an overall stronger detector, EE-SVM is better at localizing difficult instances of objects (e.g., truncated, occluded and at low resolution). It would therefore be interesting to combine these two detectors to hopefully achieve higher detection performance, compared to running them independently. Luckily, our constrained optimization problem goes beyond the calibration of the EE-SVM technique and can be generally applied to any problem exploiting interplay between scores through a max operation. In this spirit, one could calibrate R-CNN and EE-SVM jointly. R-CNN could be seen as an additional E-SVM and our joint calibration would learn a threshold for it. While we have not tried this yet, we expect that our joint calibration will assign most of the easy object instances to R-CNN and leave some more difficult instances to some specialised E-SVMs.

Chapter 5

Learning semantic part-based models from Google Images

5.1 Introduction

Part-based models have gained significant attention in the last few years. The key advantages of exploiting part representations is that parts have lower intra-class variability than whole objects, they deal better with pose variation and their configuration provides useful information about the aspect of the object. Parts localization has therefore been addressed in the context of several vision tasks (sec. 1.2.4), achieving state-of-the-art results in many of them.

Part-based methods can be grouped into two sets. The first set of works define an object part as any patch that is discriminative for the object class (Fergus et al., 2003b; Felzenszwalb et al., 2010b; Arbeláez et al., 2012; Endres et al., 2013; Juneja et al., 2013; Song et al., 2014b; Zhang et al., 2014b; Tsai et al., 2015; Wang and Yuille, 2015). These works typically discover parts in the training images automatically, without human supervision. However, their resulting parts do not have a meaning for humans (e.g., a patch straddling between the wheel and the chassis of a car (Felzenszwalb et al., 2010b)). The second set of works define parts semantically (e.g., "wheel") (Wah et al., 2011; Sun and Savarese, 2011; Ukita, 2012; Liu et al., 2012; Parkhi et al., 2012; Zhang et al., 2013, 2014b; Vedaldi et al., 2014; Chen et al., 2014; Liu et al., 2014; Wang and Yuille, 2015). These are more interpretable for a human and are necessary to obtain fine descriptions of objects and their interactions. For example, "the headlights of the bus are turned on" and "the cat is touching the TV with its tail". Moreover, part localization is necessary for a robot to correctly grasp an object (e.g. grasp a mug by

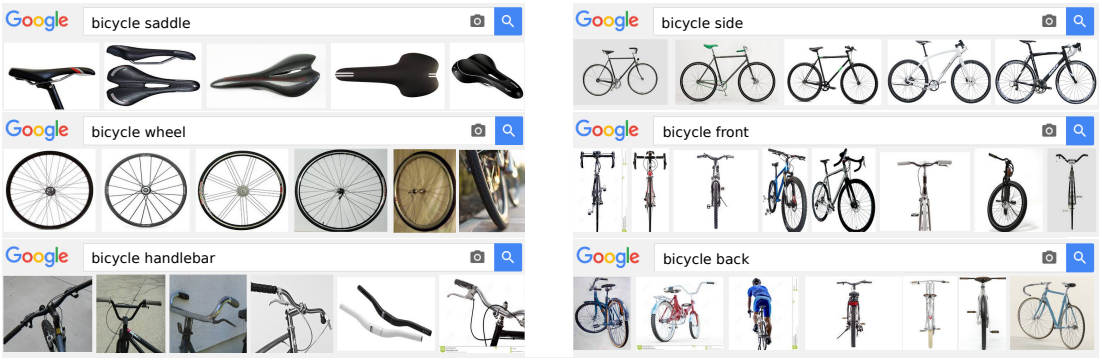


Figure 5.1: *Images returned from Google Images. On the left examples of queries for object parts, while on the right queries for an object under different viewpoints. Note how the instances are correct, clean and they mostly appear under a uniform background.*

the handle). However, existing works on semantic part detection require part location annotations in the training images, which are very expensive to obtain.

In this chapter we aim to get the best of both worlds by proposing a novel method to train semantic part models of object classes without manual location annotations. We train these models on images automatically collected from Google Images. We represent an object class as a mixture over multiple viewpoints. We learn a collection of semantic part appearance models, and models of their spatial arrangement on the object, specific to each viewpoint. Moreover, we also train models capable of predicting the viewpoint of the object, which we then use to select an appropriate location model to guide part localization on novel instances. Learning from the web has been addressed before, but only at the level of object classes (Fergus et al., 2004, 2005; Vijayanarasimhan and Grauman, 2008; Wang et al., 2009a; Li and Fei-Fei, 2010; Schroff et al., 2011; Tsai et al., 2011; Li et al., 2013; Chen et al., 2013b; Divvala et al., 2014; Chen and Gupta, 2015). To the best of our knowledge, we are the first to attempt to learn complex semantic part-based models from the web.

We learn these rich models fully automatically, entirely from Google Images, by collecting training instances for both *parts* and *objects* (fig. 5.1), and automatically connecting the two levels. Our technique incrementally learns from easy examples first, and then gradually adapts to harder examples. This adaptation is done within Google Images, where part images offer easy examples (fig. 5.1 left), and then harder examples are mined from object instances (fig. 5.1 right). The move from part images to object images also enables us to learn the spatial arrangement of parts on the object (location models). In a final step, we further adapt our models on an external, non-Google image domain to adapt to even harder examples, e.g., on PASCAL

VOC (Everingham et al., 2010).

We demonstrate the effectiveness of our incremental learning algorithm on the PASCAL VOC 2010 dataset (Everingham et al., 2010), which was annotated with part locations in (Chen et al., 2014). Interestingly, the performance of our part models increases at every step of the learning, with the final models more than doubling the initial performance (from 13.5 to 29.9 AP). Moreover, we compare to two other webly-supervised works (LEVAN (Divvala et al., 2014) and NEIL (Chen et al., 2013b)) and show that our part models perform better. Finally, we also show that our part models can help object detection performance by enriching the R-CNN detector (Girshick et al., 2014) with parts.

The rest of the chapter is organized as follows. We start by reviewing related work in sec. 5.2. Then, in sec. 5.3 we present an overview of our approach and in sec. 5.4 we describe the technical details of its components. Finally, we present our experiments in sec. 5.5 and conclude in sec. 5.6.

This work has been submitted for publication to TPAMI (Modolo and Ferrari, 2016).

5.2 Related work

Many works learn non-semantic part models, where the parts are simply arbitrary patches that are discriminative for an object class (Felzenszwalb et al., 2010b; Arbeláez et al., 2012; Endres et al., 2013; Juneja et al., 2013; Zhang et al., 2014b; Tsai et al., 2015). Our work is more related to semantic part-based models, and to techniques for learning object classes from image search engines.

Semantic part-based models. There is a considerable amount of recent work on using semantic parts to help recognition tasks. The largest part of this has recently focused on the fine-grained recognition problem in several animal domains, such as birds (Wah et al., 2011; Farrell et al., 2011; Zhang et al., 2012, 2013; Gavves et al., 2013; Liu and Belhumeur, 2013; Goering et al., 2014; Liu et al., 2014; Zhang et al., 2014a; Simon et al., 2014; Lin et al., 2015; Shih et al., 2015) and pets (Parkhi et al., 2011, 2012; Liu et al., 2012). In these works, an object is treated as a collection of parts that models its shape and appearance. Semantic parts help capturing subtle object appearance differences that could not be captured by a monolithic object model. These differences are crucial to discriminate between animal breeds.

In the bird domain, some works transfer part labels from training samples to test images (Gavves et al., 2013; Liu and Belhumeur, 2013; Goering et al., 2014; Liu et al., 2014). All these works transfer 15 semantic bird parts (e.g., beak, head, breast, wing, etc.) from the CUB-200-2011 dataset (Wah et al., 2011). Other works detect fewer semantic parts (head and torso), either defined with 3D representations (Farrell et al., 2011; Zhang et al., 2012) or 2D bounding-boxes (Zhang et al., 2013, 2014a; Simon et al., 2014; Lin et al., 2015; Shih et al., 2015). Similarly, in the pet domain, some works distinguish between cat and dog breeds by looking at their heads (Parkhi et al., 2012, 2011) and head sub-parts (eyes, nose, etc.) (Liu et al., 2012).

Other applications where semantic part-based models have been used are object class detection (Chen et al., 2014), object segmentation (Wang and Yuille, 2015), person recognition (Joon Oh et al., 2015), articulated human and animal pose estimation (Sun and Savarese, 2011; Ukita, 2012; Liu et al., 2014) and attribute prediction (Zhang et al., 2013; Vedaldi et al., 2014; Gkioxari et al., 2015). In object class detection, object segmentation and person recognition, parts help dealing with deformed, occluded and low resolution objects. In articulated pose estimation, parts help identifying objects in special configurations (e.g., jumping and sitting) as opposed to canonical ones. Finally, in attribute prediction, attributes are predicted best by the part containing direct evidence about them.

All the above mentioned methods require accurate part location annotations for training, either in terms of keypoints (Wah et al., 2011; Liu et al., 2012; Gavves et al., 2013; Liu and Belhumeur, 2013; Liu et al., 2014; Goering et al., 2014) or bounding-boxes (Sun and Savarese, 2011; Farrell et al., 2011; Parkhi et al., 2011; Zhang et al., 2012; Ukita, 2012; Parkhi et al., 2012; Zhang et al., 2013; Chen et al., 2014; Vedaldi et al., 2014; Zhang et al., 2014a; Simon et al., 2014; Lin et al., 2015; Shih et al., 2015). Our framework instead does not require annotations of part positions nor extent and automatically learns from Google Images.

Learning from image search engines. Several works have tried to learn visual models from training samples collected automatically from image search engines (Fergus et al., 2004, 2005; Vijayanarasimhan and Grauman, 2008; Wang et al., 2009a; Li and Fei-Fei, 2010; Tsai et al., 2011; Schroff et al., 2011; Li et al., 2013; Chen et al., 2013b; Divvala et al., 2014; Chen and Gupta, 2015). Most of them tackle image classification (Fergus et al., 2004, 2005; Vijayanarasimhan and Grauman, 2008; Wang et al., 2009a; Li and Fei-Fei, 2010; Tsai et al., 2011; Schroff et al., 2011; Li et al., 2013) and

develop algorithms to find good training samples and learn class models iteratively.

Some works try to learn object class detectors from the web (Chen et al., 2013b; Divvala et al., 2014; Chen and Gupta, 2015). Chen and Gupta (2015) consider only objects and not parts. LEVAN (Divvala et al., 2014) leverage Google Books Ngrams to discover all the appearance variations of an object class and trains an object detector with a separate component per variation. While some of the components happen to represent parts (e.g., *horse-head*), these are treated just like other independent components (at the same level as "jumping horse" and "racing horse"). NEIL (Chen et al., 2013b) mines web images to discover common sense relationships between object classes (e.g., "car is found in raceway"), including also some part-of relations (e.g., "wheel is part of car"). Importantly, both LEVAN and NEIL learn simple object class detectors consisting of "root filters" only. Instead we learn more complex, structured models of object classes, which include semantic part appearance models and their spatial arrangements within the object, conditioned on object viewpoint. Note how (Chen et al., 2013b; Divvala et al., 2014; Chen and Gupta, 2015) do not report quantitative localization results for part detection.

Finally, we believe our work is complementary to (Chen et al., 2013b; Divvala et al., 2014; Chen and Gupta, 2015). The frameworks of (Chen et al., 2013b; Divvala et al., 2014) could provide a list of the parts that belong to an object class, which could be passed on to our technique to learn more complex models. Moreover, our part models could be used in combination with the strong R-CNN root filters learned from the web by (Chen and Gupta, 2015) (analogous to sec. 5.5.4).

5.3 Overview of our approach

We present here an overview of our framework for automatically learning compositional semantic part models. We learn these models for each object class separately. For each class, we use Google Images to collect images of the object under several pre-defined viewpoints, and images of its parts. We use the part samples to train initial part appearance models, which are later used to learn the connection between the parts and the whole object. Learning this association is the key to our compositional part models.

For each class, we learn part appearance models $\mathcal{A}$, part location models $\mathcal{L}$, and object viewpoint classifiers $\mathcal{V}$. Our framework operates in four stages: $\mathcal{T}_0 - \mathcal{T}_3$ (fig. 5.2). In the first stage $\mathcal{T}_0$, we collect training samples from Google Images (objects and

parts, fig. 5.1 and table 5.1). Then, we iteratively learn the components of our part models, each time learning from harder examples: ($\mathcal{T}_1$) images containing only parts from Google, ($\mathcal{T}_2$) part examples mined from object images from Google, and ($\mathcal{T}_3$) part examples mined from object images from the PASCAL VOC 2010 dataset (Everingham et al., 2010). Every stage is fully automatic and does not require human intervention.

For each object part, we learn one appearance model and V location models, one for each viewpoint in our predefined set. A single location model is not sufficient to capture the position of a part with respect to the object, as this is strongly affected by viewpoint changes. For example, the front view of a bicycle has one wheel on the bottom-center of the bicycle, while a bicycle from the side has two wheels on the bottom left and right (fig. 5.1, right).

For simplicity, in the rest of the chapter we use superscripts to indicate the stage a model component is trained at. For example, our part appearance model $\mathcal{A}^2$ is trained at stage $\mathcal{T}^2$, while $\mathcal{A}^3$ at stage $\mathcal{T}^3$.

$\mathcal{T}^0$: Collecting data. Our framework queries Google Images for the object and its parts (sec. 5.4.1). More specifically, it collects images of the object under canonical viewpoints and images of each of its parts.

These images are biased towards simple representations, in a uniform background and they reliably contain the wanted object (or part). However, one image may contain multiple instances or objects not appearing nicely in the centre (fig 5.3). It would be better if each object/part instance would be enclosed in a tight bounding-box. Bounding-boxes around parts help learning accurate appearance models as they exclude background pixels, and around objects they help learning accurate part location models as they provide a stable coordinate frame common to all instances. We therefore devise a simple, yet effective algorithm to fit a tight bounding-box around each part/object instance (sec. 5.4.1). Finally, we consider each bounding-box as a separate image, obtaining our initial training set. We denote with $O_j \in O$ the set of images of the object under viewpoint v_j and with $\mathcal{P}_i \in \mathcal{P}$ the set of images of part p_i.

$\mathcal{T}^1$: Learning from Google's easy examples. For each part p_i, our framework learns an appearance model $\mathcal{A}_i^1$ on the part images $\mathcal{P}_i$ (sec. 5.4.2). These are the easiest samples, as in these images the part often appears isolated from the object and against a clean background (fig. 5.1 left).

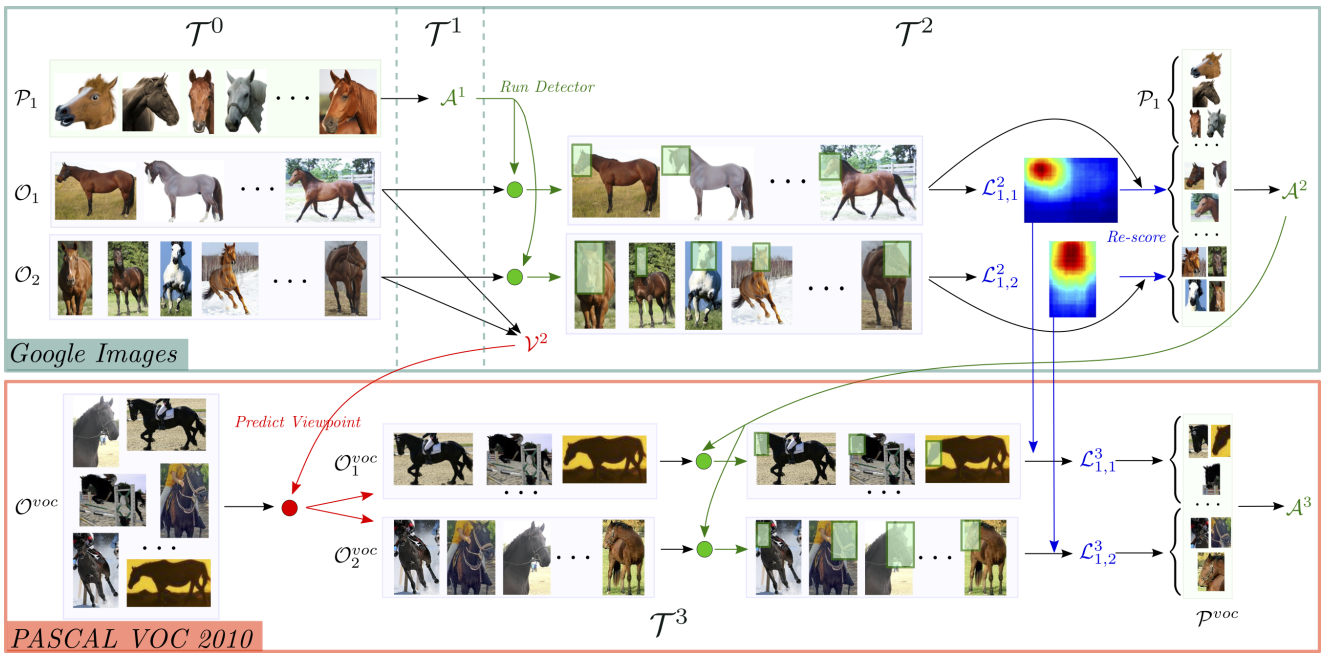


Figure 5.2: *Schema of our framework for object class "horse", simplified to just one part "head" and two viewpoints "left" and "front". At $\mathcal{T}^0$ the framework downloads horse instances (O_1 and O_2, for the two viewpoints) and head instances ($\mathcal{P}_1$) from Google Images. At $\mathcal{T}^1$ it learns a first head appearance model $\mathcal{A}^1$. At $\mathcal{T}^2$ it then hunts for new head instances from the horse images in O_1 and O_2 to first train two head location models $\mathcal{L}^2_{1,1}$ and $\mathcal{L}^2_{1,2}$, one for each horse viewpoint, and later to re-train a more accurate head appearance model $\mathcal{A}^2$. Finally, it also learns a viewpoint classifier $\mathcal{V}^2$. At $\mathcal{T}^3$ it then predicts the viewpoint of objects in O_{voc} using $\mathcal{V}^2$ and hunts for more part instances from them. These are then used to train our final part appearance model $\mathcal{A}^3$ and part location models $\mathcal{L}^3_{1,1}$ and $\mathcal{L}^3_{1,2}$. Note how at time $\mathcal{T}^1$ the framework has only seen part instances and has no information to learn neither $\mathcal{L}^1$ nor $\mathcal{V}^1$*

$\mathcal{T}^2$: Learning from Google's harder examples. In this stage our framework moves on to object images O. It learns part location models $\mathcal{L}^2$ and updates all part appearance models by using additional samples from O (sec. 5.4.2). Moreover, it trains an object viewpoint classifier $\mathcal{V}^2$ on O (sec. 5.4.4).

For each viewpoint v_j and part p_i, it learns $\mathcal{L}^2_{i,j}$. The key idea is to run $\mathcal{A}^1_i$ on the object images O_j. The top-scored part detections are likely to be correct and, importantly, they are now *localized within an object image*. Therefore, they provide valuable training samples for the location of the part within the object (sec. 5.4.3). The intuition here is that objects captured under the same viewpoint have parts in similar spatial arrangements. For example, all horses from the side have the head on the left

side of the image, mostly on top (fig. 5.2). Since all objects in O_j are in the same viewpoint v_j, correct detections of a part will be consistently found at similar locations across different object instances.

Subsequently, the framework mines part samples automatically from each $O_j \in O$ using the appearance $\mathcal{A}_i^1$ and the corresponding location model $\mathcal{L}_{i,j}^2$ (sec. 5.4.5). The process looks for detections that have a high score according to $\mathcal{A}_i^1$ and are at the right location according to $\mathcal{L}_{i,j}^2$. By combining these two sources of information, we consistently discover correct part samples. Finally, the framework uses these samples to update part appearance models to $\mathcal{A}_i^2$. Note how these new samples are more difficult than the ones in $\mathcal{T}^1$, since come from images showing whole objects and against natural backgrounds. Lastly, the framework trains an object viewpoint classifier $\mathcal{V}^2$ on O, by using each set of images $O_j \in O$ as training set for viewpoint v_j (sec. 5.4.4).

Finally, note how at this stage the framework has trained a complete, rich model (part appearance $\mathcal{A}^2$, part location $\mathcal{L}^2$, object viewpoint $\mathcal{V}^2$) entirely and automatically from Google Images (fig. 5.2, top). The key is to associate parts to their object and learn the connections.

$\mathcal{T}^3$: Learning from PASCAL VOC. In this final stage the framework refines all $\mathcal{A}^2$ and $\mathcal{L}^2$ using even more difficult training samples from another domain (sec. 5.4.2, 5.4.3). These samples are mined automatically from the PASCAL VOC dataset, which contains photographs depicting challenging objects in natural scenes, often occluded or truncated (fig. 5.2, bottom). These are much harder than the ones in stage $\mathcal{T}^2$, where each image had a single whole object. The framework mines positives as in step $\mathcal{T}^2$, but using the updated $\mathcal{A}^2$ instead of the initial $\mathcal{A}^1$ (sec. 5.4.5). Similarly to other works (Parkhi et al., 2012; Lin et al., 2015), we only search for parts inside the ground-truth bounding-boxes of the object class (which are provided with PASCAL VOC). We call this set O^{voc}. Furthermore, we call the set of mined positives $\mathcal{P}^{voc}$. In order to use our viewpoint-specific location models, we need to determine the viewpoint of the images in O^{voc}. We automatically predict v_j for each image in O^{voc} using the object viewpoint classifier $\mathcal{V}^2$. Finally, after mining new positives, the framework finally trains final location models $\mathcal{L}_{i,j}^3$ and part appearance models $\mathcal{A}_i^3$ (sec. 5.4.5).

object+side	object+part$_1$
object+front	...
object+back	object+part$_n$

Table 5.1: *Queries used by our framework to automatically collect images of objects and object parts.*

5.4 The components of our approach

We detail below the components of our approach. In sec. 5.4.1 we describe our data collection mechanism. In sec. 5.4.2, 5.4.3 and 5.4.4 we describe how to train $\mathcal{A}$, $\mathcal{L}$ and $\mathcal{V}$, respectively. In sec. 5.4.5 we then present our procedure to automatically mine new part instances from objects.

5.4.1 Data collection and preprocessing

This section describes how we download part and object images from Google and how we fit a tight bounding-box around each part/object instance in them.

Querying Google Images. We collect images of an object under multiple viewpoints and of its parts (fig. 5.1) using the queries listed in table 5.1. We keep the top 100 retrieved images for each object viewpoint and the top 25 for each object part. We observed these numbers to produce good, clean images. Collecting more than 25 part images sometimes delivers spurious images without the part, which would introduce noise in the learning process.

For each object class, we use the names of its parts as listed in the PASCAL-Part Dataset (Chen et al., 2014) and the viewpoint names specified by PASCAL VOC 2010 (Everingham et al., 2010) (*front, back, left, right*). As *left* and *right* is not a level of granularity satisfied by Google Images yet, we query for a generic *side* viewpoint (fig. 5.1 right-top) and then automatically split the retrieved images into left and right subsets. In order to do this, we first augment the image set by mirror flipping all images horizontally, and then we cluster them into two sets by minimizing the intra-cluster HOG compactness, similarly to (Felzenszwalb et al., 2010b).

Finally, note that, differently from previous methods that learn object class models from the web (Vijayanarasimhan and Grauman, 2008; Schroff et al., 2011; Li et al., 2013), our approach ignores text surrounding the image HTML tag and other meta-data, and only relies on visual cues.

Fitting bounding-boxes. As mentioned in sec. 5.3, we want to fit a tight bounding-box around each object/part instance. These bounding-boxes help learning accurate appearance and location models for the parts.

Fortunately, Google Images results are biased towards whole objects in a uniform background (fig. 5.3a) and unoccluded. These are easy to localize. We formulate this task as a pixel labelling problem, where each pixel ϕ_i can take a label $l_i \in \{0,1\}$ (background or foreground). We aim at finding the best labelling $\psi^* = \text{argmin}_L E(\psi)$. Similar to other segmentation works (Rother et al., 2004; Kuettel and Ferrari, 2012; Rubinstein et al., 2013; Rosenfeld and Weinshall, 2011), we define an energy function:

$$E(\psi) = \sum_i M_i(l_i) + \sum_i G_i(l_i) + \alpha \sum_{i,j} V(l_i, l_j) \tag{5.1}$$

As in (Rother et al., 2004; Kuettel and Ferrari, 2012; Rubinstein et al., 2013; Rosenfeld and Weinshall, 2011), the pairwise potential V encourages smoothness by penalising neighbouring pixels taking different labels and the unary potential G_i evaluates how likely a pixel i is to take label l_i according to an appearance model which consists of two GMMs (Rother et al., 2004) (one for foreground and one for background). Inspired by (Kuettel and Ferrari, 2012), we produce an initial rough estimate M of which pixels lie on the object, and use it both to estimate the appearance models G, and as a unary potential of its own. The latter helps stabilizing the segmentation process by anchoring it to the initial rough estimate.

While (Kuettel and Ferrari, 2012) estimated M by transferring segmentation masks from a database of manually segmented images, here we do it in an unsupervised manner, based purely on the spatial distribution of object proposals (Uijlings et al., 2013) in the image (fig. 5.3b). We define the likelihood $M_i(1)$ of a pixel i to be foreground as the number of proposals that contain it, divided by the total number of proposals in the image (conversely, the background likelihood is $M_i(0) = 1 - M_i(1)$). The idea is that if a pixel is contained in many proposals, then it is likely to belong to the object. Finally, instead of thresholding M, we compute G by weighting each pixel based on its probability values $(M_i(1), M_i(0))$.

As in GrabCut (Rother et al., 2004), we iteratively alternate between minimizing the energy (eq. 5.1) to obtain a segmentation, and updating the appearance models based on this segmentation. After a few iterations this process converges and we fit a tight bounding-box around each connected component in the final segmentation

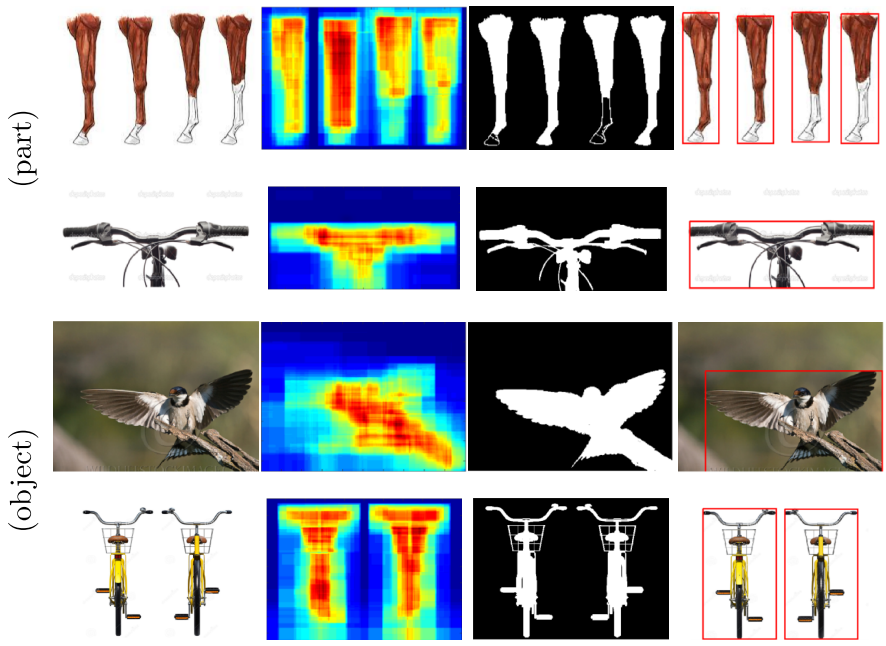


Figure 5.3: *Examples of the steps of our procedure to fit bounding-boxes to object/part instances in web images (sec. 5.4.1). (a) is the input image; (b) is the initial rough foreground estimate M; (c) is the output of the segmentation process; and (d) are the bounding-boxes fit to connected components in the segmentation. Note how the two images "horse-leg" and "bicycle-front" have multiple part/object instances and our method is able to fit a separate bounding-box around each of them.*

(fig. 5.3d). We apply this procedure to all images collected from Google, obtaining the initial training set of object images $\mathcal{O}$ and part images $\mathcal{P}$ (treating each bounding-box as a separate image).

Part proposals. We generate class-independent part proposals inside each image in $\mathcal{O}$ using (Uijlings et al., 2013). As observed by Zhang et al. (2014a), these proposals achieve low recall on small semantic parts. In order to overcome this difficulty, we changed the standard settings of Uijlings et al. (2013) to return smaller proposals and increase part recall. This results in about 2000 proposals per object image in $\mathcal{O}$, likely to cover all parts. In the rest of the chapter we use $\mathcal{W}$ to refer to the set of all part proposals over all object images.

5.4.2 Training part appearance models $\mathcal{A}$

Each stage of our learning framework updates the part appearance models of the object class. We describe here how these models are trained at each stage.

Stage $\mathcal{T}^1$. We train $\mathcal{A}^1$ on the image set $\mathcal{P}$, containing simple part images. As appearance model we use a convolutional neural network (CNN) and train it to distinguish between the P parts. More specifically, we start from AlexNet pre-trained on the ImageNet 2012 classification challenge (Krizhevsky et al., 2012) and replace its original 1000-way `fc8` classification layer with a P-way `fc8` layer. We then finetune the whole network for part classification on the images in $\mathcal{P}$. Note how $\mathcal{P}$ only contains 25 samples per part. In order to avoid overfit we decrease the learning rate to 10^{-4} and introduce early stopping (1000 iterations, 5 epochs). Higher learning rates cause the parameters to vary abruptly over iterations, whereas 10^{-4} results in a smooth learning curve. At test time we use the softmax at layer `fc8` to predict how likely a proposal is to contain each of the parts. At all times we use the publicly available CNN implementation (Jia, 2013).

Stage $\mathcal{T}^2$. At this stage we learn $\mathcal{A}^2$ on a larger training set containing examples from both part images and part samples automatically mined from object images using the appearance model from stage $\mathcal{T}^1$ and the location model from stage $\mathcal{T}^2$ (sec. 5.4.5, fig. 5.2). As appearance model we train a similar CNN to the one of stage $\mathcal{T}^1$, but with a difference: we use a richer $(P+1)$-way `fc8` layer, where the additional output is used to classify background patches of the object. Note how by mining for positive part instances in object images (sec. 5.4.5) we indirectly discover negative proposals (those having intersection-over-union ≤ 0.3 with mined positives).

Stage $\mathcal{T}^3$. In the last stage we train $\mathcal{A}^3$ on the harder image set $\mathcal{P}^{voc}$, using as training samples parts automatically mined from O^{voc} using the part detector from stage $\mathcal{T}^2$ (sec. 5.4.5). As appearance model we train a CNN as in $\mathcal{T}^2$, but increase the number of iterations to 5000.

5.4.3 Learning part location models $\mathcal{L}$

The appearance model $\mathcal{A}$ scores part proposals in an image based on their appearance only. We build location models to capture complementary knowledge about likely positions and scales of the object parts within the the coordinate frame of the object. In stage $\mathcal{T}^2$ we learn the location models purely from Google Images and in stage $\mathcal{T}^3$ we adapt them to a different domain. In this subsection we use $\mathcal{W}_j$ to refer to the set of all

part proposals in object images O_j, i.e. under viewpoint v_j.

Part training samples. For each viewpoint v_j and part p_i, we learn a separate location model $\mathcal{L}_{i,j}$ from a set of training proposals $\mathcal{W}_{j,i} \in \mathcal{W}_j$ likely to contain part p_i. We describe here how we acquire these part samples $\mathcal{W}_{i,j}$ at stage $\mathcal{T}^2$, i.e. from object images O_j. The key idea is to run the part detector $\mathcal{A}_i^1$ on these images and retain the top-scored part detections. As these detections are localized within an object image, they provide examples of the location of the part within the object. More precisely, for each image we score all part proposals with the appearance model $\mathcal{A}_i^1$, perform non-maximum suppression, and pick up to 3 detections per image (the top scored ones, if they score above a minimum confidence threshold). These detections form $\mathcal{W}'_{i,j}$. This way of picking detections strikes a good trade-off between keeping all correct locations, but without including too many false-positives. At $\mathcal{T}^3$, we enrich the sample set $\mathcal{W}_{j,i}$ with the top detections produced by running the appearance model $\mathcal{A}_i^3$ on O^{voc}. More specifically, each of these detections gets assigned to the $\mathcal{W}_{j,i}$ of viewpoint predicted by $\mathcal{V}^2$. These new samples are used to train the refined location models $\mathcal{L}_{i,j}^2$.

Training a location model. The location model $\mathcal{L}_{i,j}$ scores on an input part proposal w' by the density of the training set $\mathcal{W}_{j,i}$ at w'

$$\mathcal{L}_{i,j}(w') = \frac{1}{|\mathcal{W}_{j,i}| \cdot h} \sum_{w \in \mathcal{W}_{j,i}} K\left(\frac{D(w',w)}{h}\right) \tag{5.2}$$

where $K(u)$ is the uniform density function

$$K(u) = \frac{1}{2}\mathbb{1}(|u| \leq 1) \tag{5.3}$$

and $D(w',w)$ is distance between two windows proposals:

$$D(w',w) = 1 - \frac{w' \cap w}{w' \cup w} \tag{5.4}$$

In this formulation $\mathcal{L}_{i,j}(w')$ has an intuitive interpretation as the percentage of proposals in $\mathcal{W}_{j,i}$ which are close to w' ($IoU < h$). If many training proposals are near w', then $\mathcal{L}_{i,j}(w')$ will be large, indicating that w' is likely to contain the part. Conversely, if only a few proposals are near w', then $\mathcal{L}_{i,j}(w')$ will be small, indicating it is more likely to cover a background patch. The bandwidth h controls the degree of smoothing and in our experiments we set it to $h = 0.5$.

Note how $D(w', w)$ compares part proposals across different images. For this to be meaningful, D operates in a coordinate frame common to all images in O_j (by normalizing it by the average width and height of all images). This normalization is specific to a viewpoint v_j, so it preserves its aspect-ratio.

Model behaviour. Fig. 5.4 shows examples of some location models learned at stage $\mathcal{T}^2$. Thanks to the way we build them, our location models are robust to errors in the training set: correct training samples tend to cluster around the right locations of a part, whereas incorrect ones tend to scatter across the whole object. This results in strong peaks at the correct locations in the model, with only lower values everywhere else (e.g., *headlight* for Bus Front). Moreover, note how our location model is suitable for a variety of cases. Unique parts of rigid objects form unimodal distributions (*bicycle-saddle*, *car-license plate*), while *bicycle-wheel* and *horse-leg* form bimodal ones. Even in the hard case of highly movable parts of deformable object classes (e.g., *cat-tail*), the model learns that they can appear over broader regions and spreads the density accordingly.

5.4.4 Training the viewpoint classifier $\mathcal{V}^2$

During stage $\mathcal{T}^2$ we train classifiers $\mathcal{V}^2$ on O to predict the viewpoint of the object in an image. We train a CNN to distinguish between the four viewpoints (*front, back, left, right*) for which we collected object images from Google in sec. 5.4.1 (fig. 5.1 right). We used these images to train $\mathcal{V}^2$ and, as for $\mathcal{A}$ (sec. 5.4.2), we took the CNN pre-trained on the ImageNet classification challenge and replaced its original 1000-way `fc8` classification layer with a 4-way `fc8`.

During stage $\mathcal{T}^3$, the viewpoint classifier is useful to select an appropriate location model for object images O^{voc}. Given an input image, we select the viewpoint with the highest probability given by the softmax at `fc8`. Note that the PASCAL VOC 2010 dataset has manual viewpoint annotations for some objects ($\sim 60\%$ for the classes we consider). We use these annotations in sec. 5.5.2 to evaluate how well our viewpoint classifier $\mathcal{V}^2$ works.

5.4.5 Mining for new part instances

In stages $\mathcal{T}^2$ and $\mathcal{T}^3$ we mine for new part instances in O and O^{voc}, respectively. For simplicity, we describe the process to mine from O. Given each set of images $O_j \in O$

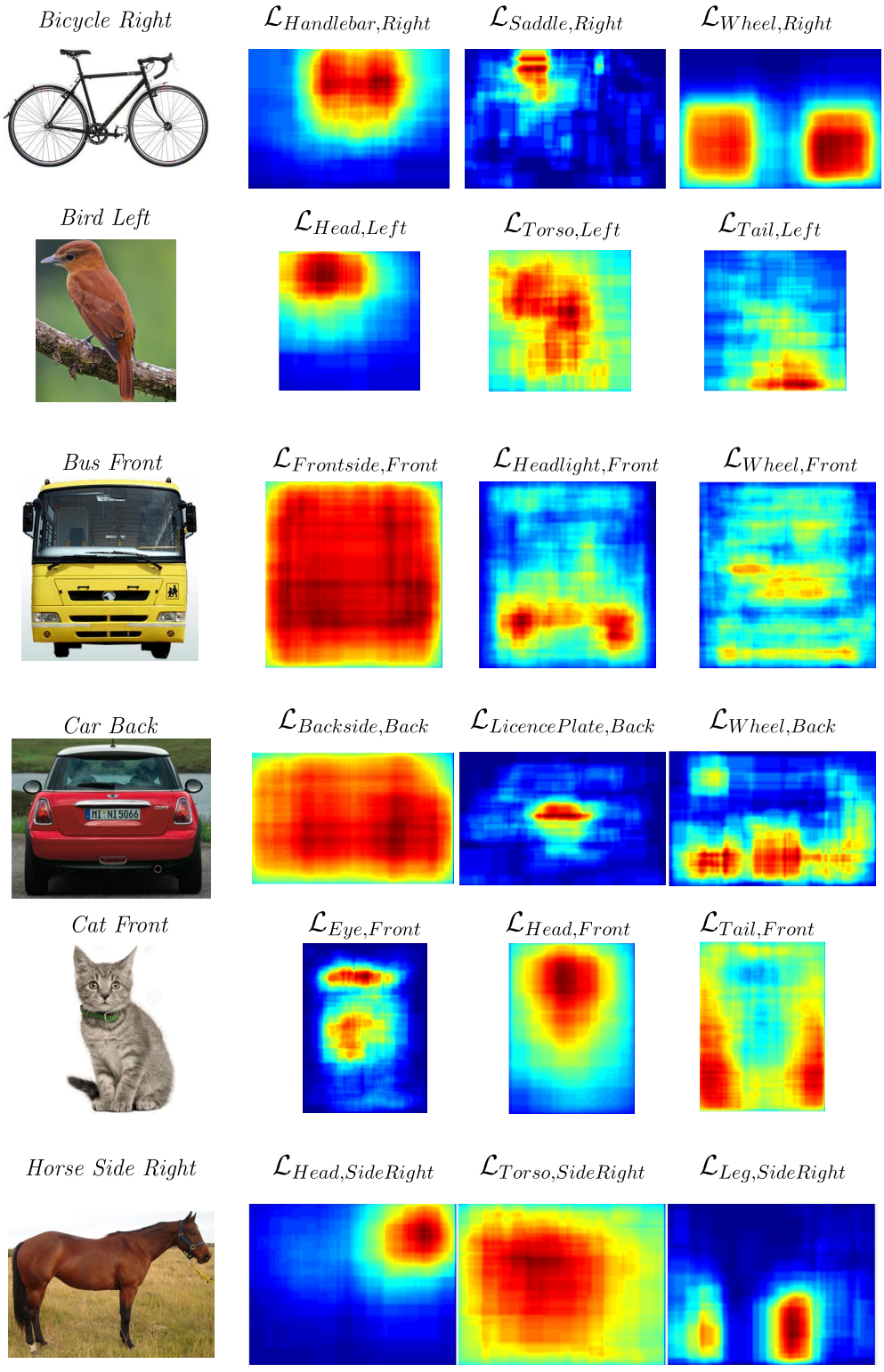


Figure 5.4: *Examples of our location models $\mathcal{L}$. We show a canonical image of each object captured under one of our viewpoints and the location models of their parts. These models nicely capture the average position of each part within the object in that viewpoint. Note how these are automatically learnt from Google Images. For visualization, we show a 2D projection of the location models, which however live in a 4D space defined not only by the (x, y) position of a proposal, but also by its scale and aspect ratio.*

showing viewpoint v_j, we mine positives for part p_i using the appearance model $\mathcal{A}_i^1$ and the location model $\mathcal{L}_{i,j}^2$, similarly to sec. 5.4.3. More specifically, for each image we score all its part proposals with the appearance model $\mathcal{A}_i^1$ and perform non-maximum suppression. Finally, we enrich the score of all surviving proposals using $\mathcal{L}_{i,j}^2$ (eq. 5.2) and retain only those with high score. We repeat this for all $O_j \in O$ and obtain our final set of new samples. Importantly, we mine for new part instances within object bounding boxes only. Even though the initial appearance models $\mathcal{A}^1$ were trained on 25 samples only, they still manage to localize new part instances, as the search space is very limited.

Mining from O^{voc} is analogous, but requires an extra step, where we use the viewpoint classifier $\mathcal{V}^2$ (sec. 5.4.4) to predict the otherwise unknown viewpoints of objects O^{voc}.

This mining process is able to add new part instances that can look significantly different than those in the initial set of easy examples $\mathcal{P}$ (e.g., a frontoparallel wheel against a white background vs a out-of-plane rotated wheel on an actual car, fig. 5.5). This is because a new part instance can be selected if at the right location according to $\mathcal{L}$, even when $\mathcal{A}$ is not confident about it.

5.5 Experiments

5.5.1 Datasets

We evaluate our framework and all its intermediate stages on the recent PASCAL-Part dataset (Chen et al., 2014), which augments PASCAL VOC 2010 (Everingham et al., 2010) with pixelwise semantic part annotations. For evaluation we fit a bounding-box to each part segmentation mask. Finally, the dataset contains a `train` and a `validation` subsets. We mine new part instances from `train` in stage $\mathcal{T}^3$, and measure the performance of our framework on `validation`.

We evaluate on six diverse object classes (*bicycle, bird, bus, car, cat, horse*), three parts each (table 5.3). We treat each leg as a separate instance, rather than grouping them into a "super-part" (as done by Chen et al. (2014)). Note how previous works evaluating on PASCAL-Parts consider fewer classes/parts (Chen et al., 2014; Wang and Yuille, 2015; Hariharan et al., 2015) and operate in a fully supervised scenario (training from manual part location annotations).

Accuracy	$\mathcal{V}^2$	$\mathcal{V}^{FS}$
Bicycle	51.0	42.6
Bird	48.1	39.3
Bus	58.4	57.8
Car	43.2	38.6
Cat	59.0	55.4
Horse	53.3	44.5
Total	**52.2**	46.4

Table 5.2: *Viewpoint prediction results. $\mathcal{V}^2$ is trained purely on Google Images, while $\mathcal{V}^{FS}$ is trained using human annotations.*

5.5.2 Viewpoint prediction

In this section we evaluate our viewpoint classifier $\mathcal{V}^2$ trained purely from Google Images (sec. 5.4.4). We compare it against a viewpoint classifier $\mathcal{V}^{FS}$ trained using manual annotations from PASCAL VOC 2010 `train`. In both cases we use the same CNN model and training procedure (sec. 5.4.4). We evaluate both classifiers in terms of accuracy on `validation` (table 5.2).

Results show that our viewpoint classifier $\mathcal{V}^2$ considerably outperforms the fully supervised classifier $\mathcal{V}^{FS}$. Results are not surprising, as objects in the PASCAL VOC dataset appear often truncated or occluded and sometimes labelled with the wrong viewpoint. Instead, the images from Google have clean objects with well defined viewpoints (fig. 5.1). Moreover, objects appear as a whole, leading to better prediction performance.

5.5.3 Part localization

In this section we evaluate how good our part models are at localizing parts in novel images. We evaluate part localization in terms of average precision (AP) on the PASCAL VOC 2010 `validation` set (which was never seen by our learning procedure). As in (Chen et al., 2014), a part is considered correctly localized if it has an intersection-over-union ≥ 0.4 with a ground-truth bounding-box.

Note how our location models are conditioned on the object viewpoint, which is however unknown for the objects in `validation`. We apply our viewpoint classifier $\mathcal{V}^2$ (sec. 5.4.4) on all objects in `validation` and select what location model to use based on its predictions. When detecting parts we use a linear combination of the score given by the appearance and location models ($\mathcal{A} + \mathcal{L}$).

		$\mathcal{T}^0$	$\mathcal{T}^1$	$\mathcal{T}^2$			$\mathcal{T}^3$			$\mathcal{A}^{FS1}$	$\mathcal{A}^{FS2}$	LEVAN	NEIL
		$\mathcal{A}^0$	$\mathcal{A}^1$	$\mathcal{A}^1+\mathcal{L}^2$	$\mathcal{A}^2$	$\mathcal{A}^2+\mathcal{L}^2$	$\mathcal{A}^3$	$\mathcal{A}^3+\mathcal{L}^2$	$\mathcal{A}^3+\mathcal{L}^3$				
	Wheel	37.2	39.6	50.5	53.9	58.7	56.6	64.0	63.9	75.7	74.2	24.7	43.1
Bicycle	*Saddle*	4.2	9.8	14.3	14.0	14.5	17.2	20.6	20.7	35.5	31.7	-	-
	Handlebar	2.2	5.9	3.5	5.9	5.9	8.5	8.7	9.9	25.6	21.1	-	-
	Head	16.4	16.4	13.5	22.4	21.0	22.7	22.6	22.7	55.3	52.0	-	-
Bird	*Torso*	2.6	5.2	10.1	38.0	48.4	48.9	53.7	55.5	60.8	56.3	-	-
	Tail	0.2	0.8	2.0	0.5	0.5	1.4	2.1	2.0	8.7	5.1	-	-
	Frontside	40.6	43.1	59.5	60.5	68.2	65.2	69.1	69.3	82.2	80.0	-	-
Bus	*Headlight*	0.8	1.8	2.1	2.6	2.9	3.5	3.8	4.0	25.5	21.2	-	-
	Wheel	15.1	19.8	18.2	23.1	22.9	27.1	27.0	27.5	50.6	47.3	5.3	4.9
	Backside	13.8	14.7	20.0	18.6	28.4	23.2	28.5	28.4	43.7	42.2	-	-
Car	*Licence Plate*	15.3	15.3	12.6	15.5	15.0	20.5	20.2	21.5	41.0	38.4	5.2	-
	Wheel	20.2	22.2	19.2	26.5	26.5	30.4	30.4	31.0	59.3	59.2	5.5	16.4
	Head	25.4	36.9	36.6	48.8	48.2	54.7	54.1	54.6	77.2	76.1	10.9	-
Cat	*Eye*	10.0	10.0	10.5	16.7	16.7	21.2	21.2	21.4	45.6	43.9	1.4	-
	Tail	0.1	0.4	1.1	1.5	1.6	2.3	2.2	2.4	15.5	9.8	-	-
	Head	30.7	33.8	33.5	35.7	34.9	37.9	37.2	37.4	64.4	63.9	22.2	-
Horse	*Torso*	8.1	16.0	46.1	47.2	53.2	48.4	55.7	59.4	71.9	68.9	-	-
	Leg	0.6	12.0	14.7	4.6	5.3	4.6	6.9	7.1	14.3	11.5	-	-
	mAP	**13.5**	**16.9**	**20.4**	**24.2**	**26.3**	**27.5**	**29.3**	**29.9**	**47.4**	**44.5**	-	-

Table 5.3: *Part detection results (average precision) on the* `validation` *set of PASCAL-Part dataset.*

We evaluate each component of our system at each stage of the learning, from $\mathcal{T}^0$ to $\mathcal{T}^3$. For the sake of evaluation, we trained additional part appearance models $\mathcal{A}^0$ directly on images retrieved by Google, *before* fitting a bounding-box around each training instance (sec. 5.4.1). Furthermore, for reference we train two fully supervised part models: $\mathcal{A}^{FS1}$ and $\mathcal{A}^{FS2}$. The former uses manual part location annotations from PASCAL-Part `train`, while the latter also uses the part instances from $\mathcal{T}^0$ collected from the web. Similarly to sec. 5.4.2 class we took AlexNet CNN and replaced its last layer with a $(P+1)$-way `fc8` (P parts an one background class). These models provide an upper-bound on what can be achieved by any weakly supervised procedure on this dataset.

Results are presented in table 5.3 and fig. 5.5. Naively using images as returned by Google Images ($\mathcal{A}^0$) leads to an AP of only 13.5. This reveals how challenging is the task of localizing object parts on a dataset like PASCAL VOC. Our refined models $\mathcal{A}^1$ perform already better and improve $\mathcal{A}^0$ by +3.4, showing that our polishing process is useful and provides cleaner examples that lead to better performance. The really interesting leap however is achieved in stage $\mathcal{T}^2$ when our framework associates the object to its parts and learns the connection. More precisely, learning the location of the parts under the different viewpoints increases AP to 20.4 ($\mathcal{A}^1+\mathcal{L}^2$). Using

this information to mine for more part instances and update the appearance model improves performance even further to 24.2 ($\mathcal{A}^2$). Ultimately, the combination of these two models ($\mathcal{A}^2 + \mathcal{L}^2$) brings the performance to 26.3. This is double the initial AP of naively training detectors directly from part images ($\mathcal{A}^0$). Importantly, at this point we have a complete class model (appearance, location, viewpoint) trained entirely and automatically from Google Images. Finally, if we additionally migrate to the PASCAL VOC domain ($\mathcal{T}^3$) and adapt appearance and location models to it, the performance further improves to a final AP of 29.9 ($\mathcal{A}^3 + \mathcal{L}^3$). The steady improvement exhibited from stage $\mathcal{T}^0$ to $\mathcal{T}^3$ by our incremental learning framework demonstrates its potential to learn complex part models automatically.

Our final part detector achieves 29.9 AP, which is 62% of the performance of a fully supervised model $\mathcal{A}^{FS}$. This result is very encouraging, given that we require no part location annotations for training, whereas $\mathcal{A}^{FS}$ inputs one bounding-box around each part instance (for a total of 10K bounding-boxes on PASCAL-Part `train` to cover our 6 classes with 3 parts each). These take a lot of time as the parts are small and difficult to annotate.

Interestingly, $\mathcal{A}^{FS2}$ performs a little worse than $\mathcal{A}^{FS1}$, despite being trained from more data. We attribute this the considerable difference between the type of images in PASCAL-Part and on the web.

Finally, we point out how a 62% performance ratio is well in line with analogue results on weakly supervised object detection, which typically report detectors performing about half as well as their fully supervised counterparts (Song et al., 2014b; Deselaers et al., 2010; Wang et al., 2015).

Comparison to LEVAN (Divvala et al., 2014) and NEIL (Chen et al., 2013b). LEVAN and NEIL learn detectors from the web and their original papers do not present quantitative evaluation on part detection. Nonetheless, a few of their models represent semantic parts. We evaluate them in this section, using their DPM models (Felzenszwalb et al., 2010b) they released online [1].

LEVAN learns multi-component object class detectors. The components within each object model are labelled with a name, like "horse jumping" or "horse head". We downloaded the detectors for our six object classes and selected all components matching our parts. For example, to detect *car-licence plate* we run the models labelled as 'plate_car_super3' and 'plate_car_super6'. NEIL, instead, learns a collection

[1] `levan.cs.washington.edu`, `www.neil-kb.com`

Figure 5.5: *Example detections obtained by running $\mathcal{A}^1 + \mathcal{L}^2$ on object images from Google (left) and $\mathcal{A}^3 + \mathcal{L}^3$ on the* `validation` *set of PASCAL-Parts (right).*

of separate models, some representing object classes and others part classes, as well as part-of relation between them. We downloaded all NEIL's models and selected those in a part-of relation with any of the object classes we consider. This only matches one part *wheel*. Only one generic wheel model is available, not associated to a specific object class.

We run all these part models on PASCAL VOC 2010 `validation` and show results in table 5.3 (rightmost two columns). Note how most of the parts we consider are missing from the components learned by NEIL and LEVAN. On the few parts that they learned, our part detectors outperform LEVAN and NEIL by a large margin. The main reason is that their models are trained from part instances downloaded from the web with no (or minimal) refinement: LEVAN uses instances similar to our $\mathcal{T}^0$, and NEIL uses something in between our $\mathcal{T}^1$ and $\mathcal{T}^2$. A second reason is that LEVAN and NEIL's components are based on simple HOG features, which are weaker than CNNs.

5.5.4 Object detection

In this section we augment the R-CNN object class detector (Girshick et al., 2014) (sec. 2.3) with our part models. The standard R-CNN detector scores each object proposal w in an image with a root filter $\mathcal{R}$ covering the whole object. Inspired by (Felzenszwalb et al., 2010b) we add a collection of parts arranged in a deformable configuration:

$$\text{score}(w) = \mathcal{R}(w) + \sum_{i}^{N} \max_{w' \in \Upsilon} (\alpha_i \cdot \mathcal{A}_i(w') + \beta_i \cdot \mathcal{L}_{i,\mathcal{V}(w)}(w')) \tag{5.5}$$

where Υ is the set of part proposals inside w. For each part i, the max operation looks for the best fitting proposal $w' \in \Upsilon$ according to the part appearance model ($\mathcal{A}_i$) and location model ($\mathcal{L}_i$), measuring how likely part i is to appear at the location w'. As we have a separate location model per viewpoint, we use our classifier $\mathcal{V}$ to select which one to use on w. We use the same set of proposals for both objects and parts (sec. 5.4.1). We set the weights α and β by cross-validation on `train`. This overall object class model is similar to (Felzenszwalb et al., 2010b), but instead of using Gaussian part location models, we have full probability distributions given by kernel density estimators (eq. 5.2).

We train the root filer on PASCAL-Parts `train` as in (Girshick et al., 2014). The other elements of the model are learned from the web and PASCAL-Parts using our

Model	*R-CNN Train Set*	*Test*	*Bicycle*	*Bird*	*Bus*	*Car*	*Cat*	*Horse*	mean
R-CNN$_1$	Obj$_1$	VOC10	64.6	46.8	63.5	56.3	69.0	55.1	59.2
R-CNN$_2$	Obj$_2$	VOC10	64.1	43.7	62.5	56.1	67.3	55.0	58.1
R-CNN$_1$ + parts	Obj$_1$	VOC10	66.9	49.3	65.6	58.4	70.8	58.0	**61.5**
R-CNN$_1$	Obj$_1$	VOC12	63.5	44.4	62.2	55.5	68.1	53.5	57.9
R-CNN$_1$ + parts	Obj$_1$	VOC12	66.1	47.2	64.1	58.0	69.6	56.7	**60.3**

Table 5.4: *Object detection results (average precision). "Obj$_1$" are the object instances in PASCAL-Part train, while "Obj$_2$" are the object instances from both PASCAL-Part* and *the images we downloaded from the web ($\mathcal{T}^0$). We evaluate on two sets: PASCAL VOC 10* `validation` *and PASCAL VOC 12* `validation`.

technique (sec. 5.4), i.e. $\mathcal{A}^3$ as part filters, $\mathcal{L}^3$ as location models and $\mathcal{V}^2$ as viewpoint classifiers. No manual part location annotations is used for training.

We report object detection results on `validation` of PASCAL VOC 10 and PASCAL VOC 12, in terms of AP in table 5.4. Compared to using the R-CNN root filter alone (R-CNN$_1$), adding parts increases its performance by 2-3% on all classes, and on both test sets (R-CNN$_1$+ parts). This shows that our part models can help object class detection, even when added to an already strong fully supervised detector like R-CNN. This is an interesting result, especially considering that our parts are designed to be *semantic*, as opposed to arbitrary patches discriminative for the object class Endres et al. (2013); Felzenszwalb et al. (2010b).

For a fully fair comparison, we also train another R-CNN model on object instances from both PASCAL-Part *and* the images we downloaded from the web ($\mathcal{T}^0$). Interestingly, training using this additional data decreases performance by 1.1% (R-CNN$_2$). Again, we attribute this to the difference between the type of images in PASCAL-Part and on the web.

5.6 Conclusions and outlook

In this chapter we presented a technique for learning part-based models from the web. It operates by collecting object and part instances and by automatically connecting them in an incremental learning procedure. Our models encompass the appearance of parts and their spatial arrangement on the object, specific to each viewpoint. We reported results on the challenging PASCAL-Parts which show that our technique is

able to learn good part detectors from the web. Finally, we demonstrated the value of our part models by enriching the R-CNN object detector with parts, which improved its performance.

Our technique could be improved and extended in several ways:

Learning more parts than what Google Images can offer. One drawback of our approach is that it is limited by the quality of Google Images. While we were able to collect good part instances for some part classes, this is not the case in general. Querying for semantic parts can sometimes return incorrect samples, which would negatively affect the behaviour of our procedure. There are many solutions that can alleviate this issue. The easiest would be to wait for Google Images to offer better retrieval sets, but it is not intellectually stimulating and it may take long time. More interesting research solutions should exploit the fact that Google Images, while not reliable for semantic parts, is already good at retrieving images of object instances. We categorize our ideas into two lines of works: transfer learning (Pan and Yang, 2010; Rohrbach et al., 2010; Tommasi et al., 2010) and humans in the loop (Branson et al., 2010; Russakovsky et al., 2015b; Papadopoulos et al., 2016).

The first idea would be to transfer the part appearance models of an object class to similar ones. For example, we could transfer the appearance models *horse-torso* and *horse-leg* trained in sec. 5.5, to the object class *cow*, for which we were not able to find decent samples for training. We then could use these appearance models to mine part samples of *cow-torso* and *cow-leg* from cow instances from Google Images. Because this process only searches within the bounding-boxes of cow instances, top-scored detections have the potential to be good part samples. These could then be used to learn proper appearance models for *cow-torso* and *cow-leg*. Finally, we could strengthen this mining process even further by also transferring part locations models for a viewpoint, as many object classes have parts in similar locations (e.g., both *horse-head* and *cow-head* often appear on the top-left corner on instances of objects with side left viewpoint).

The second idea follows the trend of (Russakovsky et al., 2015b; Papadopoulos et al., 2016). After training an initial weak part appearance model, it would employ an active learning strategy that introduces humans in the mining loop. Humans can be used to verify bounding-boxes produced automatically by the weak part detectors. This would quickly lead to the collection of high-quality annotations, as annotators would

merely need to decide whether a bounding-box is correct or not and maybe to what degree. Moreover, it would be interesting to expand this idea and create a hyper active learning process that automatically decides for what semantic parts it is necessary to ask humans and what part instances to show them, similar to (Branson et al., 2010). For example, we would imagine that for easy parts like *bicycle-wheel* the system would only ask human judgements on some difficult instances, while for more difficult parts like *bicycle-handlebar*, the system would ask humans more questions.

Viewpoint specific part appearance models. Given a semantic part, our technique trains a separate location model for each object viewpoint. It would be interesting to extend our framework to also train viewpoint specific part appearance models. Similar to the location of a part, its appearance variation within an object viewpoint is also smaller than across all viewpoints. For example, *bicycle-wheel* changes appearance considerably depending on the viewpoint of the bicycle: if the bicycle is sideways the wheel resemble a circle, if the bicycle is photographed from the front, the wheel is a thin vertical line. Each viewpoint specific appearance model could focus on one of these part variations, potentially improving part detection performance. Importantly, this extension is straightforward in our framework, as our mining procedure already collects object viewpoint specific part instances.

Learning the number of part instances in an object viewpoint. Another possible extension to our framework would be to learn the number of part instances that appear within an object viewpoint, as this number tends to be quite constant. For example, two eyes are usually visible in an image of a cat from the front, one from a cat from the side and zero from a cat from the back. Currently, we let our location models learn these differences automatically. As we observed in sec. 5.4.3, our location model tends to have strong peaks for visible parts and scattered distributions for non-visible ones. Nonetheless, we believe that explicitly modelling this information within our framework would further improve the final object detection performance. For example, at training time it could strengthen our mining process and at test time it could be used to remove false positive detections.

Adding interaction between parts. In our framework we consider each part independently. However, parts are highly correlated and their interactions should be exploited. An idea that could improve weak detectors would be to run part-based models in a

cascade (Viola and Jones, 2001), from strongest to weakest, each time reducing the search space of the next detector. For example, we could run our good *bicycle-wheel* detector (sec. 5.5) first and our weaker *bicycle-handlebar* one afterwords. Importantly, the second detector would only run on regions of the bicycle images not containing instances of wheels. This would prevent our mining procedure from collecting false positives at unlikely locations and at test time to remove wrong detections. Furthermore, one could learn pairwise information between two parts, capturing their spatial relation, as these are stable within the same object viewpoint. For example, *horse-torso* always appears on top of *horse-leg*. In the example described earlier, we could find *bicycle-wheel* instances using our strong detector and use its learnt spatial relation with *bicycle-handlebar* to suggest where to look for instances of this part.

Chapter 6

Do semantic parts emerge in Convolutional Neural Networks?

6.1 Introduction

Semantic parts are object regions interpretable by humans (e.g., wheel, leg) and play a fundamental role in several visual recognition tasks. For this reason, semantic part-based models have gained significant attention in the last few years (sec. 1.2.4). The key advantages of exploiting semantic part representations is that parts have lower intra-class variability than whole objects, they deal better with pose variation and their configuration provides useful information about the aspect of the object.

Recently, convolutional neural networks (CNNs) have achieved impressive results on many visual recognition tasks, like image classification, object detection, semantic segmentation and fine-grained recognition (sec. 2.3). Thanks to these outstanding results, CNN-based representations are quickly replacing hand-crafted features, like SIFT (Lowe, 2004) and HOG (Dalal and Triggs, 2005a).

In this chapter we look into these two worlds and address the following question: *"does a CNN learn semantic parts in its internal representation?"* In order to answer it, we investigate whether the network's convolutional filters learn to respond to semantic parts of objects. Some previous works (Zeiler and Fergus, 2014; Simonyan et al., 2014) have suggested that semantic parts do emerge in CNNs, but only based on looking at some filter responses on a few images. Here we go a step further and perform two quantitative evaluations that examine the different stimuli of the CNN filters and try to associate them with semantic parts. First, we take advantage of the available ground-truth part location annotations in the PASCAL-Part dataset (Chen et al., 2014)

to count how many of the annotated semantic parts emerge in a CNN. Second, we use human judgments to determine what fraction of all filters systematically fire on any semantic part (including parts that might not be annotated in PASCAL-Part).

For the first evaluation we use part ground-truth location annotations in the PASCAL-Part dataset (Chen et al., 2014) to answer the following question: *"how many semantic parts emerge in CNNs?"*. As an analysis tool, we turn filters into part detectors based on their responses to stimuli. If some filters systematically respond to a certain semantic part, their detectors will perform well, and hence we can conclude that they do represent the semantic part. Given the difficulty of the task, while building the detectors we assist the filters in several ways. The actual image region to which a filter responds typically does not accurately cover the extent of a semantic part. We refine this region by a regressor trained to map it to a part's ground-truth bounding-box. Moreover, as suggested by other works (Simon et al., 2014; Simon and Rodner, 2015; Xiao et al., 2015), a single semantic part might emerge as distributed across several filters. For this reason, we also consider filter combinations as part detectors, and automatically select the optimal combination of filters for a semantic part using a Genetic Algorithm. We present an extensive analysis on AlexNet (Krizhevsky et al., 2012) finetuned for object detection (Girshick et al., 2014). Results show that 34 out of 105 semantic parts emerge. This is a modest number, despite all favorable conditions we have engineered into the evaluation and all assists we have given to the network. This result demystifies the impressions conveyed by (Zeiler and Fergus, 2014; Simonyan et al., 2014) and shows that the network learns to associate filters to part classes, but only for some of them and often to a weak degree. In general, these semantic parts are those that are large or very discriminative for the object class (e.g., torso, head, wheel). Finally, we analyze different network layers, architectures, and supervision levels. We observe that part emergence increases with the depth of the layer, especially when using deeper architectures such as VGG16 (Simonyan and Zisserman, 2015). Moreover, emergence decreases when the network is trained for tasks less related to object parts, e.g., scene classification (Zhou et al., 2014).

Our second quantitative evaluation answers the converse question: *"what fraction of all filters respond to any semantic part?"*. As PASCAL-Parts is not fully annotated (e.g., car door handle is missing), we answer it using human judgments. For each filter, we show human annotators the 10 images with the highest activations per object class. We highlight the regions corresponding to the activations and ask the annotators whether they systematically cover the same concept (e.g., a semantic part, a

background, a texture, a color, etc.). In the case of a positive answer, we ask them to name the concept (e.g., horse hoof). In general, the majority of the filters do not seem to systematically respond to any concept. On average per object class, 7% of the filters correspond to semantic parts (including several filters responding to the same semantic part). About 10% of the filers systematically respond to other stimuli such as colors, subregions of parts or even assemblies of multiple parts. Finally, we also compare the semantic parts emerging in this evaluation with the 34 parts annotated in PASCAL-Part that emerged in the first evaluation. We find that nearly all the parts that emerge according to the detection performance criterion used in the first evaluation also emerge according to human judgments. However, more semantic parts emerge according to human judgments, including several parts that are not annotated in PASCAL-Part.

Finally, we also investigate how discriminative network filters and semantic parts are for the objects the network is trained to recognize. We explore the possibility that some filters respond to "parts" as recurrent discriminative patches, rather than truly semantic parts. We find that, for each class, there are on average 9 discriminative filters that are largely responsible for recognizing it. Interestingly, 40% of these are also semantic according to human judgments, which is a much greater proportion than the 7% found when considering *all* filters. The overlap between which filters are discriminative and which are semantic might be the reason why previous works (Zeiler and Fergus, 2014; Simonyan et al., 2014) have suggested a stronger emergence of semantic parts, based on qualitative visual inspection. We also investigate to what degree the emergence of semantic parts in the network correlates with their discriminativeness for recognition. Interestingly, these are highly correlated: semantic parts that are discriminative emerge much more than other semantic parts. While this is generally assumed in the community, ours is the first work presenting a proper quantitative evaluation that turns this assumption into a fact.

The rest of the chapter is organized as follows. Sec. 6.2 discusses some related work. Sec. 6.3 presents our quantitative evaluation using PASCAL-Part bounding-boxes, while evaluation using human judgments is presented in sec. 6.4. The discriminativeness of filters is investigated in sec. 6.5, while the discriminativeness of semantic parts in sec. 6.6. Finally, sec. 6.7 summarizes the conclusions of our study and presents some ideas of how to extend it.

This work has been submitted for publication to IJCV (Gonzalez-Garcia et al., 2016). Authorship is shared with fellow PhD student Abel Gonzalez-Garcia. Abel and I worked full-time on this project and contributed substantially to it (60-40). We de-

signed and developed all the analysis presented in this chapter together. Nonetheless, for the purpose of explicitly dividing our contributions, I take credit for the following analyses: sec. 6.3.2.2 ("differences between part sizes"), sec. 6.3.3, sec. 6.4.2 (as annotator), sec. 6.4.3 and sec. 6.6.

6.2 Related Work

Analyzing CNNs. CNN-based representations are unintuitive and there is no clear understanding of why they perform so well or how they could be improved. In an attempt to better understand the properties of a CNN, some recent vision works have focused on analyzing their internal representations (Szegedy et al., 2014; Yosinski et al., 2014; Lenc and Vedaldi, 2015; Mahendran and Vedaldi, 2015; Zeiler and Fergus, 2014; Simonyan et al., 2014; Agrawal et al., 2014; Zhou et al., 2015; Eigen et al., 2013). Some of these investigated properties of the network, like stability (Szegedy et al., 2014), feature transferability (Yosinski et al., 2014), equivariance, invariance and equivalence (Lenc and Vedaldi, 2015), the ability to reconstruct the input (Mahendran and Vedaldi, 2015) and how the number of layers, filters and parameters affects the network performance (Agrawal et al., 2014; Eigen et al., 2013).

More related to this work are (Zeiler and Fergus, 2014; Simonyan et al., 2014; Agrawal et al., 2014; Zhou et al., 2015), which look at the convolutional filters. Zeiler and Fergus (2014) use deconvolutional networks to visualize locally optimal visual inputs for individual filters. Simonyan et al. (2014) use a gradient-based visualization technique to highlight the areas of an image discriminative for an object class. Agrawal et al. (2014) show that the feature representations are distributed across object classes. Zhou et al. (2015) show that the layers of a network learn to recognize visual elements at different levels of abstraction (e.g., edges, textures, objects and scenes). Most of these works make an interesting observation: filter responses can often be linked to semantic parts (Zeiler and Fergus, 2014; Simonyan et al., 2014; Zhou et al., 2015). These observations are however mostly based on casual visual inspection of few images (Zeiler and Fergus, 2014; Simonyan et al., 2014). (Zhou et al., 2015) is the only work presenting some quantitative results based on human judgments, but not focused on semantic parts. Instead, we present an extensive quantitative analysis on whether filters can be associated with semantic parts and to which degree. We transform the filters into part detectors and evaluate their performance on ground-truth part bounding-boxes from the PASCAL-Part dataset (Chen et al., 2014). Moreover, we present a

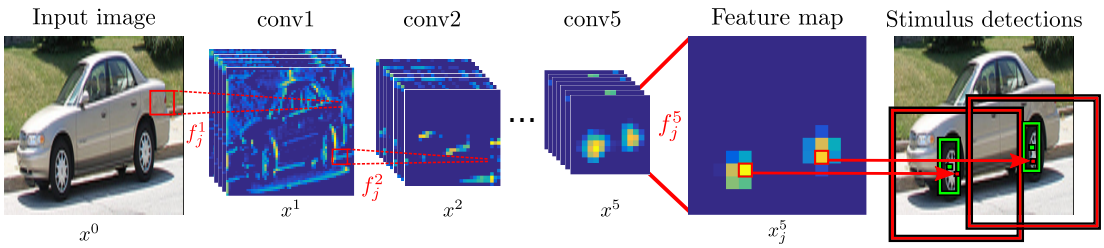


Figure 6.1: *Overview of our approach for a layer 5 filter. Each local maxima of the filter's feature map leads to a stimulus detection (red). We transform each detection with a regressor trained to map it to a bounding-box tightly covering a semantic part (green).*

second quantitative analysis based on human judgments where we categorize filters into semantic parts. We believe this methodology goes a step further than previous works and supports more conclusive answers to the quest for semantic parts.

Filters as intermediate part representations for recognition. Several works use filter responses for recognition tasks (Simon et al., 2014; Gkioxari et al., 2015; Simon and Rodner, 2015; Xiao et al., 2015; Oquab et al., 2015). Simon et al. (2014) train part detectors for fine-grained recognition, while Gkioxari et al. (2015) train them for action and attribute classification. Furthermore, Simon and Rodner (2015) learn constellations of filter activation patterns, and Xiao et al. (2015) cluster groups of filters responding to different bird parts. All these works assume that the convolutional layers of a network are related to semantic parts. In this chapter we try to shed some light on this assumption and hopefully inspire more works which exploit the network's internal structure for recognition.

6.3 PASCAL-Parts emergence in CNNs

Our goal is understanding whether the convolutional filters learned by the network respond to semantic parts. In order to do so, we investigate the image regions to which a filter responds and try to associate them with a particular part.

Network architecture. Standard image classification CNNs such as (Krizhevsky et al., 2012; Simonyan and Zisserman, 2015) process an input image through a sequence of layers of various types, and finally output a class probability vector. Each layer i takes the output of the previous layer x^{i-1} as input, and produces its output x^i by apply-

ing up to four operations: convolution, nonlinearity, pooling, and normalization. The convolution operation slides a set of learned filters of different sizes and strides over the input. The nonlinearity of choice for many networks is the Rectified Linear Unit (ReLU) (Krizhevsky et al., 2012), and it is applied right after the convolution.

6.3.1 Methodology

Fig. 6.1 presents an overview of our approach. Let f_j^i be the j-th convolutional filter of the i-th layer, including also the ReLU. Each pixel in a feature map $x_j^i = f_j^i(x^{i-1})$ is the activation value of filter f_j^i applied to a particular position in the feature maps x^{i-1} of the previous layer. The resolution of the feature map depends on the layer, decreasing as we advance through the network. Fig. 6.1 shows feature maps for layers 1, 2, and 5. When a filter responds to a particular stimulus in its input, the corresponding region on the feature map has a high activation value. By studying the stimuli that cause a filter to fire, we can characterize them and decide whether they correspond to a semantic object part.

6.3.1.1 Stimulus detections from activations

The value $a_{c,r}$ of each particular activation α, located at position (c,r) of feature map x_j^i, indicates the response of the filter to a corresponding region in its input x^{i-1}. By recursively back-propagating this region down the layers, we can reconstruct the actual receptive field on the input image, i.e. the whole image region on which the filter acted. The size of the receptive field varies depending on the layer, from the actual size of the filter for the first convolutional layer, up to a much larger image region on the top layer. For each feature map, we select all its local maxima as activations with high response. Each of these activations will lead to a stimulus detection in the image. The location of such detection is defined by the center of the receptive field of the activation, whereas its size varies depending on the layer. Fig. 6.1 shows an example, where the two local maxima of feature map x_j^5 lead to the stimulus detections depicted in red.

Regressing to part bounding-boxes. The receptive field of an activation gives a rough indication about the location of the stimulus. However, it rarely covers a part tightly enough to associate the stimulus with a part instance (fig. 6.2). In general, the receptive field of high layers is significantly larger than the part ground-truth bounding-

box, especially for small classes like ear. Moreover, while the receptive field is always square, some classes have other aspect ratios (e.g., legs). Finally, the response of a filter to a part might not occur in its center, but at an offset instead (e.g., on the bottom area, fig. 6.2d-e).

In order to factor out these elements, we assist each filter with a bounding-box regression mechanism that refines its stimulus detection for each part class. The regressor applies a 4D transformation, i.e. translation and scaling along width and height. We believe that if a filter fires systematically on many instances of a part class at the same relative location (in 4D), then we can grant that filter a "part detector" status. This implies that the filter responds to that part, even if the actual receptive field does not tightly cover it. For the rest of the chapter, all stimulus detections include this regression step unless stated otherwise.

We train one regressor for each part class and filter. Let $\{G^l\}$ be the set of all ground-truth bounding-boxes for the part in the training set. Each instance bounding-box G^l is defined by its center coordinates (G^l_x, G^l_y), width G^l_w, and height G^l_h. We train the regressor on K pairs of activations and ground-truth part bounding-boxes $\{\alpha^k, G^k\}$. Let (c_x, c_y) be the center of the receptive field on the image for a particular feature map activation α of value $a_{c,r}$, and let w, h be its width and height ($w = h$ as all receptive fields are square). We pair each activation with an instance bounding-box G^l of the corresponding image if (c_x, c_y) lies inside it. We then learn a 4D transformation d_x, d_y, d_w, d_h to predict a part bounding-box G' from α's receptive field

$$G'_x = x + d_x(\gamma(\alpha)) \qquad G'_w = d_w(\gamma(\alpha))$$
$$G'_y = y + d_y(\gamma(\alpha)) \qquad G'_h = d_h(\gamma(\alpha))$$

where $\gamma(\alpha) = (c_x, c_y, a_{c-1,r-1}, a_{c-1,r}, ..., a_{c+1,r+1})$. Therefore, the regression depends on the center of the receptive field and on the values of the 3x3 neighborhood of the activation on the feature map. Note that it is independent of w and h as these are fixed for a given layer. Each d_* is a linear combination of the elements in $\gamma(\alpha)$ with a weight vector w_*, where $*$ can be x, y, w, or h.

We set regression targets $(t^k_x, t^k_y, t^k_w, t^k_h) = (G^k_x - c^k_x, G^k_y - c^k_y, G^k_w, G^k_h)$ and optimize the following weighted least squares objective

$$w_* = \underset{w'_*}{\operatorname{argmin}} \sum_{k=1}^{K} a^k_{c,r}(t^k_* - w'_* \cdot \gamma(\alpha^k))^2. \tag{6.1}$$

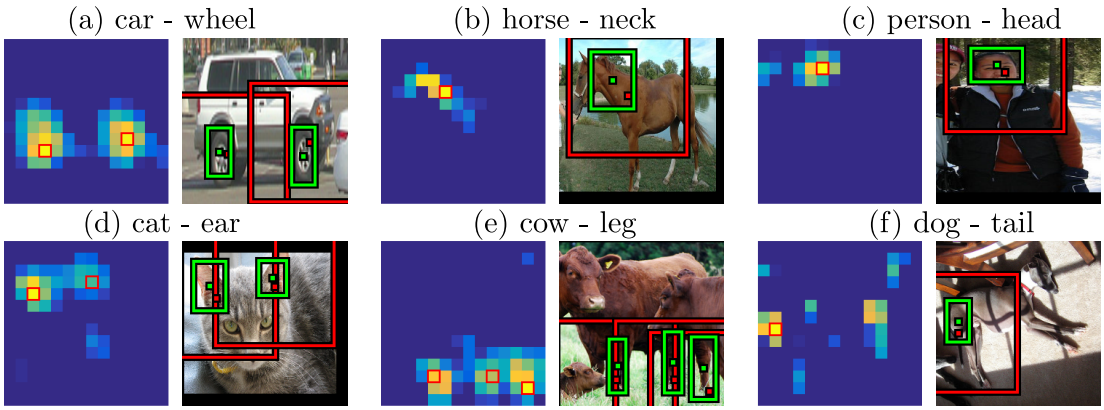


Figure 6.2: *Examples of stimulus detections for layer 5 filters. For each part class we show a feature map on the left, where we highlight the strongest activation in red. On the right, instead, we show the corresponding original receptive field and the regressed box.*

In practice, this tries to transform the position, size and aspect-ratio of the original receptive field of the activations into the bounding-boxes in $\{G^l\}$.

Fig. 6.2 presents some examples of our bounding-box regression for 6 different parts. For each part, we show the feature map of a layer 5 filter and both the original receptive field (red) and the regressed box (green) of some activations. We can see how given a strong activation on the feature map, the regressor not only refines the center of the detection, but also successfully captures its extent. Some classes are naturally more challenging, like *dog-tail* in fig. 6.2f, due to higher size and aspect-ratio variance or lack of satisfactory training examples.

Evaluating filters as part detectors. For each filter and part combination, we need to evaluate the performance of the filter as a detector of that part. We take all the local maxima of the filter's feature map for every input image and compute their stimulus detections, applying Non-Maxima Suppression (Felzenszwalb et al., 2010b) to remove duplicate detections. We consider a stimulus detection as correct if it has an intersection-over-union ≥ 0.4 with any ground-truth bounding-box of the part, which is the usual condition for part detection (Chen et al., 2014). All other detections are considered false positives A filter is a good part detector if it has high recall but a small number of false positives, indicating that when it fires, it is because the part is present. Therefore, we use Average Precision (AP) to evaluate the filters as part detectors, following the PASCAL VOC (Everingham et al., 2010) protocol.

6.3.1.2 Filter combinations

Several works (Agrawal et al., 2014; Zhou et al., 2015; Xiao et al., 2015) noted that one filter alone is often insufficient to cover the spectrum of appearance variation of an object class. We believe that this holds also for part classes. For this reason, we present here a technique to automatically select the optimal combination of filters for a part class.

For a given network layer, the search space consists of binary vectors $\mathbf{z} = [z_1, z_2, ..., z_N]$, where N is the number of filters in the layer. If $z_i = 1$, then the i-th filter is included in the combination. We consider the stimulus detections of a filter combination as the set union of the individual detections of each filter in it. Ideally, a good filter combination should make a better part detector than the individual filters in it. Good combinations should include complementary filters that jointly detect a greater number of part instances, increasing recall. At the same time, the filters in the combination should not add many false positives. Therefore, we can use the collective AP of the filter combination as objective function to be maximized:

$$\mathbf{z} = \underset{\mathbf{z}'}{\operatorname{argmax}} \operatorname{AP}\Big(\bigcup_{i \in \{j | z'_j = 1\}} \det_i \Big), \tag{6.2}$$

where $\det_i$ indicates the stimulus detections of the i-th filter.

We use a Genetic Algorithm (GA) (Mitchell, 1998) to optimize this objective function. GAs are iterative search methods inspired by natural evolution. At every generation, the algorithm evaluates the "fitness" of a set of search points (population). Then, the GA performs three genetic operations to create the next generation: selection, crossover and mutation. In our case, each member of the population (chromosome) is a binary vector $\mathbf{z}$ as defined above. Our fitness function is the AP of the filter combination. In our experiments, we use a population of 200 chromosomes and run the GA for 100 generations. We use Stochastic Universal Sampling (Mitchell, 1998). We set the crossover and mutation probabilities to 0.7 and 0.3, respectively. We bias the initialization towards a small number of filters by setting the probability $P(z_i = 1) = 0.02, \forall i$. This leads to an average combination of 5 filters when $N = 256$, in the initial population.

6.3.2 AlexNet for object detection

In this section we analyze the role of convolutional filters in AlexNet and test whether some of them can be associated with semantic parts. In order to do so, we design our settings to favor the emergence of this association.

6.3.2.1 Experimental settings

Dataset. We evaluate filters on the recent PASCAL-Part dataset (Chen et al., 2014), which augments PASCAL VOC 2010 (Everingham et al., 2010) with pixelwise semantic part annotations. For our experiments we fit a bounding-box to each part segmentation mask. We use the `train` subset and evaluate all parts listed in PASCAL-Part with some minor refinements: we discard fine-grained labels (e.g., "car wheel front-left" and "car wheel back-left" are both mapped to *car-wheel*), merge contiguous subparts of the same larger part (e.g., "person upper arm" and "person lower arm" become a single part *person-arm*), discard very tiny parts (average of widths and heights over the whole training set ≤ 15 pixels, like "person eyebrow"), and discard parts with less than ≤ 10 samples in `train` (like 'bicycle headlight', which has only one annotated sample). The final dataset contains 105 parts of 16 object classes.

AlexNet. One of the most popular networks in computer vision is the CNN model of Krizhevsky et al. (Krizhevsky et al., 2012), winner of the ILSVRC 2012 image classification challenge (Russakovsky et al., 2015a). It is commonly referred to as AlexNet. This network has 5 convolutional layers followed by 3 fully connected layers. The number of filters at each of the convolutional layers L is: 96 (L1), 256 (L2), 384 (L3), 384 (L4), and 256 (L5). The filter size changes across layers, from 11x11 for L1, to 5x5 to L2, and to 3x3 for L3, L4, L5.

Training. We use the publicly available AlexNet network of Girshick et al. (2014). The network was initially pre-trained for image classification on the ILSVRC12 dataset and subsequently finetuned for object class detection (for the 20 classes in PASCAL VOC + background) using ground-truth bounding-boxes. Note how these bounding-boxes provide a coordinate frame common across all object instances. This makes it easier for the network to learn parts as it removes variability due to scale changes (the convolutional filters have fixed size) and presents different instances of the same part

class at rather stable positions within the image. We refer to this network as *AlexNet-Object*. The network is trained on the `train` set of PASCAL VOC 2012. Note how this set is a superset of PASCAL VOC 2010 `train`, on which we analyze whether filters correspond to semantic parts.

Finally, we assist each of its filters by providing a bounding-box regression mechanism that refines its stimulus detections to each part class (sec. 6.3.1.1) and we learn the optimal combination of filters for a part class using a GA (sec. 6.3.1.2).

Evaluating settings. We restrict the network inputs to ground-truth object bounding-boxes. More specifically, for each part class we look at the filter responses only inside the instances of its object class and ignore the background. For example, for *cow-head* we only analyze *cow* ground-truth bounding-boxes. Furthermore, before inputting a bounding-box to the network we follow the R-CNN pre-processing procedure (Girshick et al., 2014), which includes adding a small amount of background context and warping to a fixed size. An example of an input bounding-box is shown in fig. 6.1. These settings are designed to be favorable to the emergence of parts as we ignore image background that does not contain parts and, more importantly, we use object instances seen by AlexNet-Object during training.

6.3.2.2 Results

Table 6.1 shows results for few parts of ten object classes in terms of average precision (AP). For each part class and network layer, the table reports the AP of the best individual filter in the layer ("Best"), the increase in performance over the best filter thanks to selecting a combination of filters with our GA ("GA"), and the number of filters in that combination ("nFilters"). Moreover, the last row of the table reports the mAP over all 105 part classes. Several interesting facts arise from these results.

Need for regression. In order to quantify how much the bounding-box regression mechanism of sec. 6.3.1.1 helps, we performed part detection using the non-regressed receptive fields. On AlexNet-Object layer 5, taking the single best filter for each part class achieves an mAP of 6.1. This is very low compared to mAP 19.0 achieved by assisting the filters with the regression. Moreover, results show that the receptive field is only able to detect large parts (e.g., *bird-torso, bottle-body, cow-torso*, etc.). This

Class	Part	Layer 1 (96)			Layer 2 (256)			Layer 3 (384)			Layer 4 (384)			Layer 5 (256)		
		Best	**GA**	**nFilters**	**Best**	**GA**	**nFilters**	**Best**	**GA**	**nFilters**	**Best**	**GA**	**nFilters**	**Best**	**GA**	**nFilters**
aero	body	17.7	+3.4	12	23.7	+10.2	33	29.4	+9.5	62	34.0	+9.2	49	29.3	+17.0	49
	stern	10	+1.5	14	13.6	+5.1	33	21.4	+2.5	45	19.2	+5.2	32	15.0	+9.4	21
	wing	4.2	+1.2	12	5.6	+5.3	41	6.0	+7.0	63	6.9	+3.9	36	4.7	+9.6	38
	engine	2.1	+0.9	14	2.7	+1.6	5	4.2	+1.7	37	4.5	+2.5	53	1.6	+5.4	25
bike	wheel	14.2	+0.7	19	41.4	+0.0	30	49.2	+0.0	3	60.0	+0.0	10	57.1	+6.1	16
	saddle	1.0	+0.5	5	1.7	+0.5	18	1.6	+0.6	43	1.7	+0.0	4	2.1	+2.5	16
	handlebar	2.8	+0.4	17	3	+2.2	21	4.0	+2.0	37	3.2	+3.1	40	4.1	+5.8	38
	chainwheel	0.6	+0.1	4	0.6	+0.6	24	1.9	+0.0	46	1.7	+1.0	17	3.0	+0.6	6
bottle	cap	1.8	+0.6	13	4.4	+2.2	20	6.4	+0.9	21	11.2	+0.0	11	6.6	+4.6	15
	body	73.0	+0.6	4	80.9	+0.0	9	87.6	+0.0	10	83.4	+0.0	3	81.0	+6.3	25
bird	head	5.7	+0.0	1	8.0	+0.4	5	14.7	+5.0	16	24	+2.0	17	23.8	+6.5	23
	beak	0.8	+0.0	1	0.9	+0.4	35	1.1	+2.4	57	1.4	+3.2	45	2.1	+4.5	28
	wing	4.6	+1.5	11	7.0	+7.8	38	9.7	+5.2	39	9.3	+8.3	33	7.7	+8.3	36
	tail	1.8	+0.2	9	2.4	+1.8	39	2.9	+3.7	39	4.6	+3.8	25	7.0	+4.3	17
car	backside	10.8	+0.0	1	16.1	+0.4	30	21.0	+0.0	5	14.8	+1.3	41	17.6	+4.5	25
	liplate	1.2	+0.0	2	1.2	+0.8	7	4.2	+0.0	10	3.1	+0.2	24	4.3	+1.5	7
	door	5.3	+0.3	5	7.8	+1.4	36	11.1	+4.0	35	13.0	+4.6	55	16.5	+7.5	23
	wheel	3.5	+0.0	1	9.5	+3.2	8	27.7	+3.8	6	30.0	+4.8	15	35.4	+4.0	17
cat	head	16.8	+0.0	1	21.2	+0.0	8	30.6	+6.0	8	44.5	+1.1	15	53.9	+5.2	10
	eye	0.6	+0.0	4	10.4	+1.6	3	10.8	+0.0	2	3.8	+0.5	18	4.3	+1.3	4
	ear	1.9	+0.6	10	4.4	+0.3	14	4.9	+5.7	12	10.7	+5.1	13	17.5	+2.8	10
	torso	35.8	+0.7	6	40.2	+ 1.7	4	43.6	+2.4	32	46.7	+6.1	32	50.8	+4.2	25
	paw	0.6	+0.1	6	0.7	+0.4	18	1.9	+1.2	21	3.2	+0.0	11	1.5	+1.9	16
	tail	1.0	+0.0	1	1.3	+1.4	35	2.1	+2.8	71	2.3	+4.0	42	2.2	+3.9	24
cow	head	12.9	+0.4	12	15.8	+3.8	34	21.2	+8.5	71	22.9	+4.9	50	24.6	+17.1	34
	muzzle	3.4	+0.2	14	4.9	+2.9	37	15.4	+0.0	10	15.6	+1.9	14	16.7	+9.5	22
	torso	43.1	+0.0	1	56.6	+0.0	45	62.0	+9.0	42	63.6	+9.9	53	65.2	+13.6	42
	tail	0.9	+0.0	9	2.9	+1.1	19	2.9	+2.2	16	7.0	+2.5	30	3.7	+0.8	6
horse	head	5.4	+0.3	3	7.6	+1.8	22	10.7	+3.1	52	15.3	+5.2	27	16.1	+11.6	22
	ear	0.9	+0.3	11	1.3	+1.1	18	4.3	+1.6	9	2.7	+3.2	12	6.1	+0.5	4
	muzzle	3.2	+1.6	6	2.7	+2.3	29	4.9	+3.2	48	8.2	+5.4	30	12.1	+5.4	19
	torso	48.7	+0.9	9	52.7	+4.1	22	63.8	+0.0	11	63.0	+4.4	27	65.2	+7.1	29
	leg	6.3	+0.2	10	10.7	+3.6	6	14.2	+7.5	8	23.0	+5.7	9	23.4	+9.6	14
person	head	6.6	+0.0	1	8.7	+0.0	3	33.8	+0.0	5	44.9	+0.0	6	58.2	+0.0	1
	hair	3.9	+0.1	4	5.1	+0.0	3	18.0	+0.0	11	28.7	+0.0	4	30.6	+0.0	1
	torso	16.1	+0.0	1	21.7	+1.8	10	23.7	+6.8	10	32.8	+1.7	9	38.3	+4.4	8
	arm	3.7	+0.0	1	4.7	+1.5	6	4.5	+3.5	13	5.4	+1.4	19	8.5	+4.7	7
	foot	0.4	+0.0	1	0.7	+0.0	6	1.8	+0.0	6	1.3	+0.3	2	1.6	+1.0	6
train	head	42.5	+2.0	4	51.8	+5.6	41	53.2	+6.2	19	58.7	+6.5	31	64.0	+11.4	39
	frontside	14.9	+2.3	10	22.1	+9.1	27	28.7	+11.0	49	30.1	+13.3	26	27.9	+20.1	47
	roofside	7.5	+1.0	12	6.8	+3.7	22	14.9	+0.0	6	8.8	+0.7	17	13.2	+5.7	4
	headlight	1.5	+0.4	6	1.5	+0.6	24	1.1	+0.8	27	1.1	+0.3	23	0.8	+0.2	4
mean (105 parts)		9.0	+0.6 (9.6)	6.9	12.4	+2.0 (14.4)	19.1	16.5	+2.6 (19.1)	24.0	17.8	+3.3 (21.1)	24.6	19.0	+5.2 (24.2)	18.8

Table 6.1: *Part detection results in terms of AP on the* `train` *set of PASCAL-Part for AlexNet-Object.* Best *is the AP of the best individual filter whereas* GA *indicates the increment over* Best *obtained by selecting the combination of* (nFilters) *filters.*

is not surprising, as the receptive field of layer 5 covers most of the object surface (fig. 6.2). Instead, filters with regressed receptive fields can detect much smaller parts (e.g., *cat-ear, cow-muzzle, person-hair*), as the regressor shrinks the area covered by the receptive field and adapts its aspect ratio to the one of the part. We conclude that the receptive field alone cannot perform part detection and regression is necessary.

Differences between layers. Generally, the higher the network layer, the higher the performance. This is consistent with previous observations (Zeiler and Fergus, 2014; Zhou et al., 2015) that the first layers of the network respond to generic corners and other edge/color junctions, while higher levels capture more complex structures. Nonetheless, it seems that some of the best individual filters of the very first layers can already perform detection to a weak degree when helped by our regression (e.g., *bike-wheel*).

Differences between part classes. Performance varies greatly across part classes. For example, some parts (e.g., *aeroplane-engine, bike-chainwheel* and *person-foot*) are clearly not represented by any filter nor filter combination, as their AP is very low across all layers. On other parts (e.g., *bike-wheel, cat-head* and *horse-torso*), instead, the network achieves good detection performance, proving that some of the filters can be associated with these parts.

Differences between part sizes. Another factor that seems to influence the performance is the average size of the part. For example, the AP achieved on *horse-torso* is much higher than on the smaller *horse-ear*. In order to understand if this is common across all parts, we looked at how AP changes with respect to the average size of a part (fig. 6.3). Interestingly, these two are indeed correlated and have a Pearson product-moment correlation coefficient (PPMCC) of 0.7 (Pearson, 1895). This shows that smaller parts emerge less in the CNN than larger ones. Nonetheless, small size does not always imply low detection performance: there are some rather small parts (around 20% of the object area) which have high AP (around 60), like *bicycle-wheel*, *motorbike-wheel* and *person-head*. As we show in sec. 6.6, these are parts that are very discriminative for recognizing the objects they belong to, which justifies their emergence within the network.

Filter combinations. Performing part detection using a combination of filters (GA)

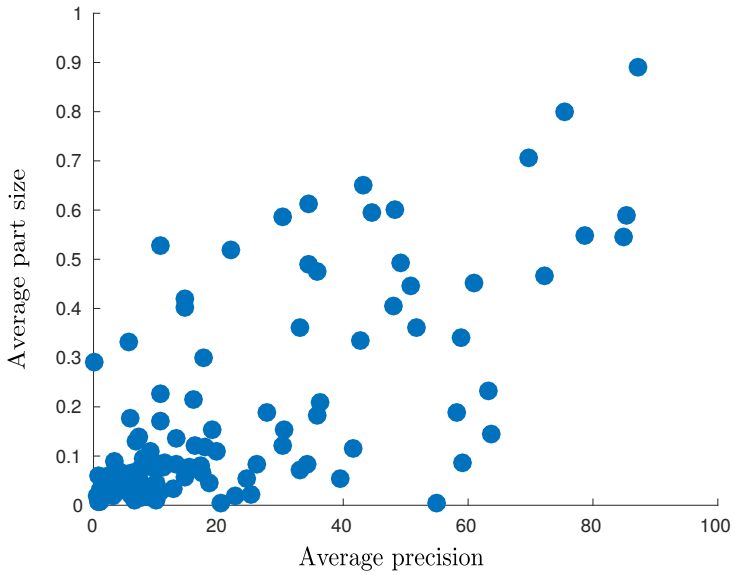


Figure 6.3: *Correlation between the size of a part class, averaged over all its instances, and part detection performance (AP for the best combination of layer 5 filters found by GA, Table 6.1). The part size is normalized by the average size of the object class it belongs to. Each point corresponds to a different part class. These are highly correlated: Pearson product-moment correlation coefficient of 0.7.*

always performs better (or equal) than the single best filter. This is interesting, as it shows that different filters learn different appearance variations of the same part class. Moreover, combining multiple filters improves part detection performance more for deeper layers. This suggests that they are more class-specific, i.e. they dedicate more filters to learning the appearance of specific object/part classes. This can be observed by looking not only at the improvement in performance brought by the GA, but also at the number of filters that the GA selects. Clearly, filters in L1 are so far from being parts that even selecting many filters does not bring much improvement (+0.6 mAP only). Instead, in L4 and L5 there are more semantic filters and the GA combination helps more (+3.3 mAP and +5.2 mAP, respectively). Interestingly, for L5 the improvement is higher than for L4, yet fewer filters are combined. This further shows that filters in higher layers better represent semantic parts.

GA analysis. The AP improvement provided by our GA for some parts is remarkable, like for *aeroplane-body* (+17.0), *horse-leg* (+9.6) and *cow-head* (+17.1). While these results suggest that our GA is doing a good job in selecting filter combinations, here we compare against a simpler method that selects the top few best filters for a

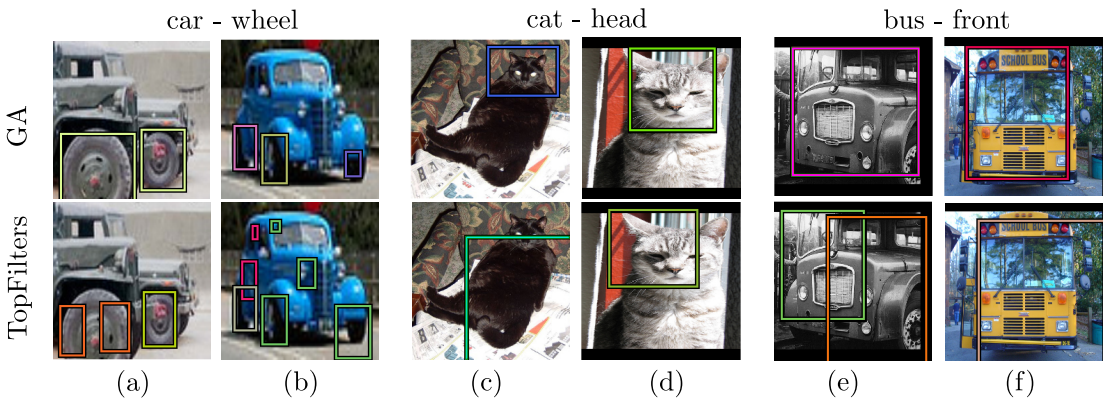


Figure 6.4: *Part detection examples obtained by combination of filters selected by our GA (top) or by TopFilters (bottom). Different box colors correspond to different filters' detections. Note how the GA is able to better select filters that complement each other.*

part class. We refer to it as TopFilters. We let both methods select the same number of filters and evaluate their combinations in terms of mAP. Our GA consistently outperforms TopFilters (24.2 vs 18.8 mAP, layer 5). The problem with TopFilters is that often the top individual best filters capture the same visual aspect of a part. Instead, our GA can select filters that complement each other and work well jointly (indeed 57% of the filters it selects are not among those selected by TopFilters). We can see this phenomenon in fig. 6.4. On the blue car, TopFilters detects two wheels correctly, but fails to fit a tight bounding-box around the third wheel that appears much smaller (fig. 6.4a). Similarly, on the other car TopFilters fails to correctly localize the large wheel (fig. 6.4b). Instead, the GA localizes all wheels correctly in both cases. Furthermore, the GA fits tighter bounding-boxes for more challenging parts, achieving more accurate detections (fig. 6.4c-h). Finally, note how TopFilters does not even improve over selecting the single best filter (19.0), as more filters bring more false positive detections.

Filter sharing across part classes. We looked into which filters were selected by our GA and noticed that some are shared across different part classes. By looking at these filters' detections, it is clear that some filters are representative for a generic part and work well on all object classes containing it (fig. 6.5).

Instance coverage. Table 6.1 shows high AP results for several part classes, showing how some filters can indeed act as part detectors. However, as AP conflates both recall

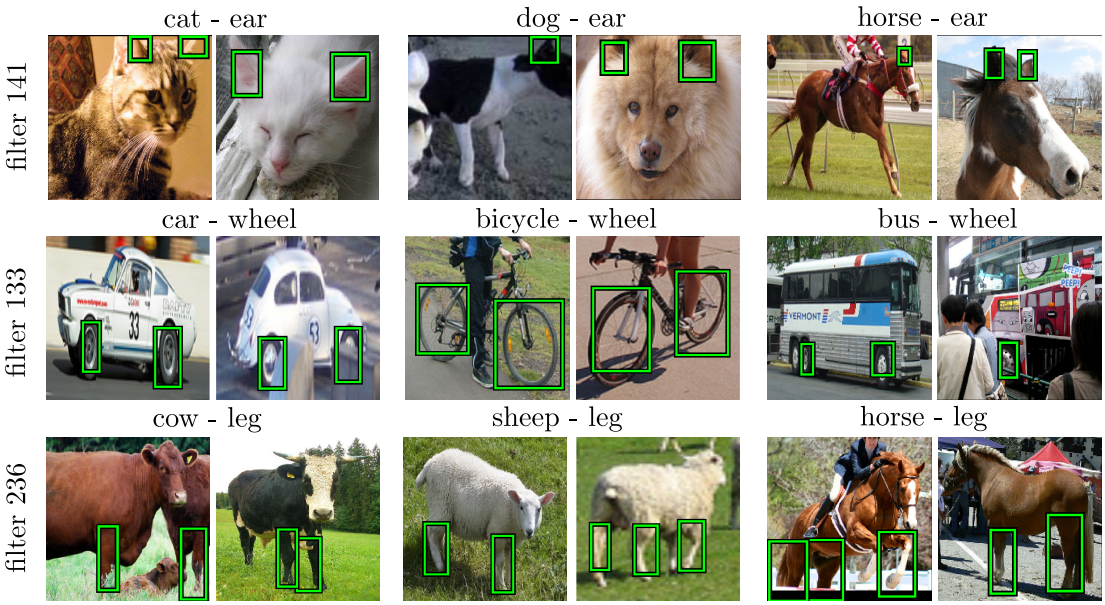


Figure 6.5: *Detections performed by filters 141, 133, and 236 of AlexNet-Object, layer 5. The filters are specific to a part and they work well on several object classes containing it.*

and false-positives, it does not reveal how many part instances the filters cover. To answer this question, fig. 6.6 shows recall vs. false-positives curves for several part classes. For each part class, we take the top three filters of layer 5, and compare them to the filter combination returned by the GA. We can see how the combination reaches higher AP not only by having fewer false positives in the low recall regime, but also by reaching considerably higher recall levels than the individual filters. For some part classes, the filter combination covers as many as 80% of its instances (e.g., *car-door, bike-wheel, dog-head*). For the more challenging part classes, neither the individual filters nor the combination achieve high recall levels, suggesting that the convolutional filters have not learned to respond to these parts systematically (e.g., *cat-eye, horse-ear*).

How many semantic parts emerge in AlexNet-Object? So far we discussed part detection performance for individual filters of AlexNet-Object and their combinations. Here we want to answer the main bottomline question: for how many part classes does a detector emerge? We answer this for two criteria: AP and instance coverage.

For AP, we consider a part to emerge if the detection AP for the best filter combination in the best layer (L5) exceeds 30. This is a rather generous threshold, which represents the level above which the part can be somewhat reliably detected. Accord-

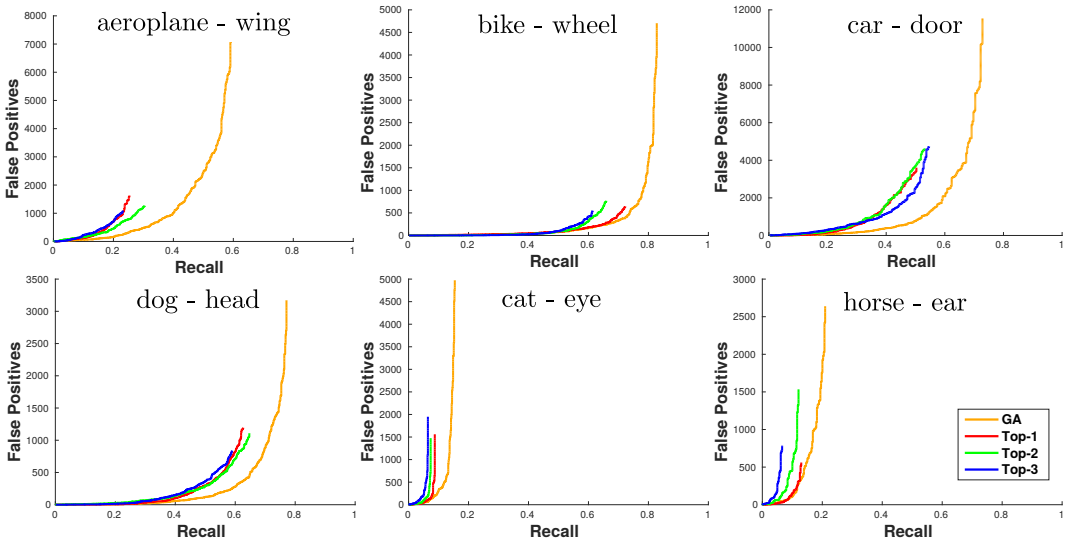


Figure 6.6: *Recall vs. false positives curves for six part classes using AlexNet-Object's layer 5 filters. For each part class we show the curve for the the top three individually best filters and for the combination of filters selected by our GA.*

ing to this criterion, 34 out of the 105 semantic part classes emerge. This is a modest number, despite all favorable conditions we have engineered into the evaluation and all assists we have given to the network (including bounding-box regression and optimal filter combinations).

According to the instance coverage criterion, instead, results are more positive. We consider that a filter combination covers a part when it reaches a recall level above 50%, regardless of false-positives. According to this criterion, 71 out of the 105 part classes are covered, which is greater than the number of part detectors found according to AP. This indicates that, although for many part classes there is a filter combination covering many of its instances, it also fires frequently on other image regions (leading to high false positives rates).

Based on all this evidence, we conclude that the network does contain filter combinations that can cover some part classes well, but they do not fire exclusively on the part, making them weak part detectors. This demystifies the observations drawn through casual visual inspection, as in (Zeiler and Fergus, 2014). Moreover, the part classes covered by such semantic filters tend to either cover a large image area, such as torso or head, or be very discriminative for their object class, such as wheels for vehicles and wings for birds. Most small or less discriminative parts are not represented well in the network filters, such as headlight, eye or tail.

6.3.3 Other network architectures and levels of supervision

We now explore how the level of supervision provided during training and the network architecture affect what the filters learn.

Networks and training. We consider several additional networks with different supervision levels (*AlexNet-Image*, *AlexNet-Scenes*, and *AlexNet-Object←Scratch*), and a different architecture (*VGG16-Object*).

AlexNet-Image (Krizhevsky et al., 2012) is trained for image classification on 1.3M images of 1000 object classes in ILSVRC 2012 (Russakovsky et al., 2015a). Note how this network has not seen object bounding-boxes during training. For this reason, we expect its filters to learn less about semantic parts than AlexNet-Object. On the opposite end of the spectrum, AlexNet-Scene (Zhou et al., 2014) is trained for scene recognition on 205 categories of the Places database (Zhou et al., 2014), which contains 2.5M scene-centric images. As with AlexNet-Image, this network has not seen object bounding-boxes during training. But now the training images show complex scenes composed of many objects, instead of focusing on individual objects as in ILSVRC 2012. Moreover, while the network might learn to use objects as cues for scene recognition (Zhou et al., 2015), the task also profits from background patches (e.g., *water* and *sky* for beach). For these reasons, we expect object parts to emerge even less in AlexNet-Scene. For both AlexNet-Image and AlexNet-Scene we use the publicly available models from (Jia, 2013).

We introduce AlexNet-Object←Scratch in order to assess the importance of pre-training in AlexNet-Object. We directly train this network for object detection on PASCAL VOC 2012 from scratch, i.e. randomly initializing its weights instead of pre-training on ILSVRC 2012. The rest of the training process remains identical to the one for AlexNet-Object (sec. 6.3.2.1).

Finally, VGG16-Object is the 16-layer network of Simonyan and Zisserman (2015), finetuned for object detection (Girshick et al., 2014) on PASCAL VOC 2012 (like AlexNet-Object). While its general structure is similar to AlexNet, it is deeper and the filters are smaller (3x3 in all layers), leading to better image classification (Simonyan and Zisserman, 2015) and object detection (Girshick, 2015) performance. Its convolutional layers can be grouped in 5 blocks. The first two blocks contain 2 layers each, with 64 and 128 filters, respectively. The next block contains 3 layers of 256 filters. Finally, the last 2 blocks contain 3 layers of 512 filters each.

Network name	Training		Results - Layer		
	Pre-train	Train	**L3**	**L4**	**L5**
AlexNet-Object	ILSVRC12	VOC12	19.1	21.1	24.2
AlexNet-Image	-	ILSVRC12	20.0	21.5	23.2
AlexNet-Scene	-	Places	17.5	17.6	18.1
AlexNet-Object←Scratch	-	VOC12	14.5	16.2	18.2
VGG16-Object	ILSVRC12	VOC12	12.9	21.5	26.1

Table 6.2: *Part detection results (mAP). For VGG-16, L3, L4, and L5 correspond to L3_3, L4_3, and L5_3, respectively.*

Results. Table 6.2 presents results for all networks we consider. For the AlexNet architectures, we focus on the last three convolutional layers, as we observed in sec. 6.3.2.2 that filters in the first two layers correspond poorly to semantic parts. Analogously, for VGG16-Object we present the top layer of each of the last 3 blocks of the network (L3_3, L4_3, and L5_3). We report mAP results obtained by the GA filter combination, averaged over all part classes.

Both AlexNet-Image and AlexNet-Object present reasonable part emergence across all their layers. This shows that the network's inclination to learn semantic parts is somewhat already present even when trained for whole image classification, suggesting that object parts are useful for that task too. However, the part emergence on L5 for AlexNet-Object is higher. This indicates that parts become more important when the network is trained for object detection, affecting particularly higher layers, near the final classification layer that can use the responses of these filters to recognize the object.

Interestingly, parts emerge much less when training the network for scene recognition, as the results of AlexNet-Scene indicate (-6.1% mAP compared to AlexNet-Object). The relative performance of the three networks AlexNet-Object, AlexNet-Image, AlexNet-Scene suggest that the network seems to learn parts to the degree it needs them for the task it is trained for, a remarkable behaviour indeed.

AlexNet-Object←Scratch performs clearly worse than AlexNet-Object, which is likely due to the fact that PASCAL VOC 2012 is too small for training a complex CNN, and so pre-training on ILSVRC is necessary (Agrawal et al., 2014; Girshick et al., 2014). Finally, parts emerge more in the deeper VGG16-Object than in AlexNet-Object (L5). This suggests that having additional layers encourages learning filters that

better model semantic parts, which might actually be the reason for the better image classification performance of VGG16 (Simonyan and Zisserman, 2015).

All the networks but AlexNet-Scene confirm the trend observed for AlexNet-Object: filters in higher layers are more responsive to semantic parts. This is especially notable for VGG16-Object. As this network has many more layers, the levels of semantic abstraction are more spread out. For example, L3_3 has very low emergence as there are six more layers above it instead of two in AlexNet. Therefore, the network can postpone the development of semantic part filters to later layers. AlexNet-Scene displays the same, rather low level of responsiveness to semantic parts in all layers considered. We hypothesize this is due to semantic object parts playing a smaller role for scene recognition.

6.4 Part emergence in CNNs according to humans

Our quantitative evaluation presented in sec. 6.3 uses the semantic part annotations available in PASCAL-Part dataset (Chen et al., 2014) to determine how many semantic parts emerge in the CNNs. We address now the converse question: *"what fraction of all filters respond to any semantic part?"* Despite being the best existing parts dataset, PASCAL-Part is not complete: some semantic parts are not annotated (e.g., the door handle of a car). For this reason, we cannot answer this new question using it, as a filter might be responding to an unannotated semantic part.

We propose here a human-based experiment that goes beyond the semantic parts annotated in PASCAL-Part. For each object class we ask human annotators if activations of a filter systematically correspond to a semantic part of that object class, and, if yes, to name the part (sec. 6.4.1). This data provides a mapping from filters to semantics parts, which is only limited by the semantic parts known by the annotator. Using this mapping, we can now answer the proposed question (sec. 6.4.2). Moreover, this mapping also allows us to compare the parts emerging in this human experiment with the parts that emerged according to the PASCAL-Part annotations(sec. 6.3). This enables to discern whether some other parts emerge besides those annotated in PASCAL-Part (sec. 6.4.3).

Figure 6.7: *Example of a question shown to our annotators, filter id: 186, class:* car. *We show the top 10 images with the highest activation of the filter on this object class, along with the corresponding activation maps.*

6.4.1 Methodology

For each class and filter, we present an image like fig. 6.7 to an annotator. The image shows the top 10 activations of the filter on object instances of the class (*car*, in the figure). We show the image shaded and overlay the activation map by setting the transparency value of the shading proportionally to the activation value at a pixel. This highlights the activation map and helps the annotator to quickly see high activation regions in the context of the rest of the image. We also indicate the maximum of the activation map with a green square to emphasize the strongest activation (which is typically in the middle of the region).

The task consists in answering the question: *"Do the highlighted areas in the images systematically cover the same concept, and if so, which?"*. This is the case if the highlighted areas cover the same concept in at least seven out of the ten images. The possible answers are the following:

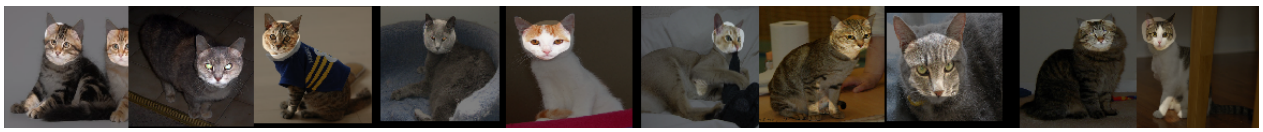
1. Yes - Semantic part
2. Yes - Background
3. Yes - Other
4. No
5. Not sure

In the case of an affirmative response, the annotator needs to specify one of three types of concepts: semantic part (e.g., wheel), background (e.g., grass) or anything else (e.g., white color). Additionally, we ask them to name the concept by typing it in a free-text field. The idea behind this protocol is to distinguish filters that fire on a variety of different image structures (fig. 6.8j-k), including occasionally some semantic parts, from genuine part detectors, which fire systematically on a particular part. Furthermore, we have expanded our experiment beyond semantic parts (i.e. background and other) in order to achieve a more comprehensive understanding of the filter stimuli. The last option ("Not sure") allows the annotator to skip ambiguous cases, which we later reject.

We ask a question for each combination of object class and network filter. We explore L5 filters of AlexNet-Object (256) and we consider the 16 object classes used in sec. 6.3, leading to a total of 256×16 questions. We use two expert annotators to process half of the object classes each. To measure agreement, they also process one of the object classes from the other annotator's set. Their agreement on the types of filters is high: in 79% of the questions, the two annotators clicked on the same answer (out of the 5 possible answers above).

6.4.2 Results

With this experiment we derive a distribution over the types of filters for each object class. Fig. 6.9 shows these distributions for five example object classes (*bird*, *car*, *cat*, *horse* and *sheep*) as well as the average result for all 16 object classes. Additionally, fig. 6.8 shows some example human answers. The majority of the filters do not seem to systematically respond to any identifiable concept (fig. 6.8j-k). Among the filters that do respond to concepts systematically, only an average of 7% (18 filters) correspond to semantic parts of a particular object class (fig. 6.8a-c). Only 3% of the filters respond systematically to background patches, examples include "grass", "road", and "sky"



a) filter 251 - cat: systematic - semantic part, “face”

b) filter 243 - person: systematic - semantic part, “arm”

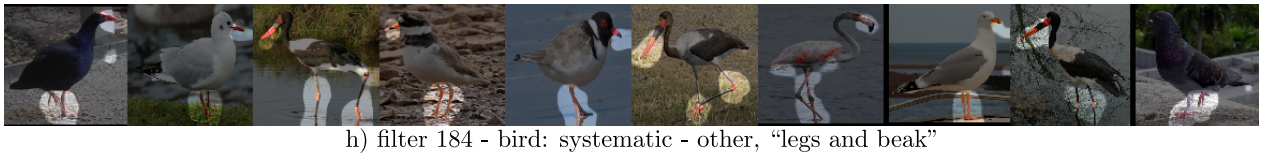
c) filter 246 - bicycle: systematic - semantic part, “wheel”

d) filter 191 - cow: sytematic - background, “grass”

e) filter 3 - motorbike: sytematic - background, “road”

f) filter 66 - person: systematic - other, “white”

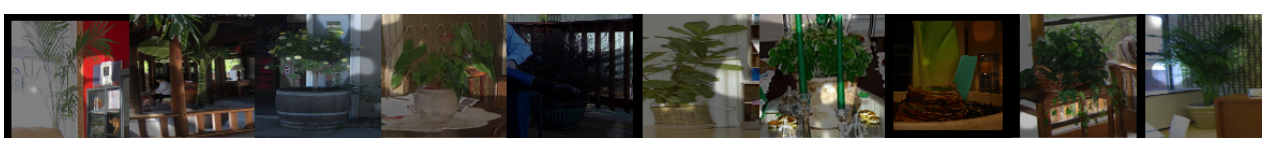
g) filter 196 - dog: systematic - other, “area between eyes and nose”



h) filter 184 - bird: systematic - other, “legs and beak”

i) filter 4 - train: systematic - other, “top with sky”

j) filter 60 - car: not systematic



k) filter 161 - pottedplant: not systematic

Figure 6.8: *Example of human annotation for different filters and classes.*

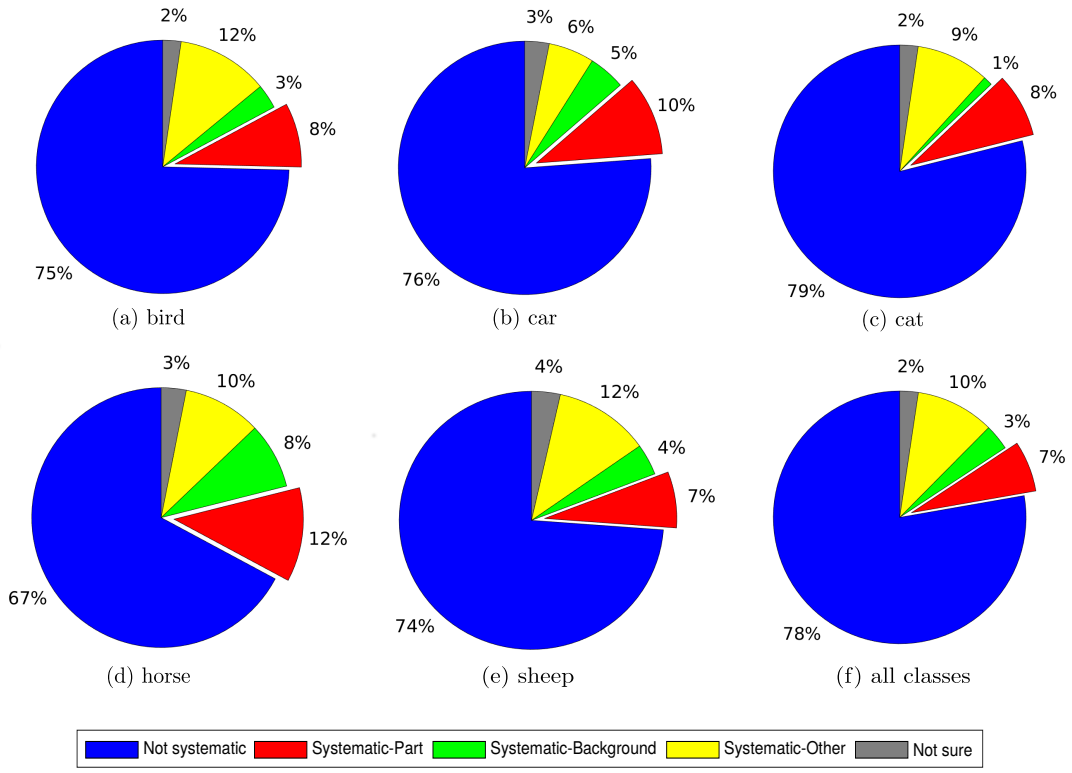


Figure 6.9: *Filter distributions for bird, car, cat, horse, sheep and the average of all 16 classes.*

(fig. 6.8d-e). Finally, most of the systematic filters, 10% overall, respond to some other concepts. Among these, we most often find colors and textures (fig. 6.8f), but also subregions of a part (fig. 6.8g), assemblies of multiple semantic parts or their subregions (fig. 6.8h), or even regions straddling between a part and a background patch (fig. 6.8i).

By further inspection, we found that background filters are consistent across images of different object classes. For example, a filter that systematically fires on "grass" patches does it for most classes commonly found outdoors. Similarly, some of the "other" systematic filters, especially the ones responding to colors, also exhibit the same behavior across object classes. In contrast, the situation for systematic filters responding to semantic parts is mixed. Although in some cases a filter responds to similar semantic parts of different object classes (like "wheel" or "leg", as in fig. 6.5), in some other cases this does not hold. For example, there is a filter that responds to "wheel" (in *car* images), "leg" (in *cow* images), and "paw" (in *cat* images). This indicates that the filter is responding to several types of stimuli simultaneously, possibly due to a higher order stimulus of which humans are not aware.

6.4.3 Comparison to PASCAL-Part

In this section we compare the part emergence observed in sec. 6.3 with the part emergence from this human experiment. Moreover, we also look at what semantic parts emerge according to our annotators, but are not present in PASCAL-Parts.

Emergence of PASCAL-Part classes. In sec. 6.3 we observed that 34 out of the 105 semantic parts of PASCAL-Part emerge in AlexNet-Object according to our AP criterion (layer 5, sec. 6.3.2.2). 24 out of these 34 parts also emerge according to the human judgments. Of the missing 10, four are animals *torso*, for which humans prefer more localized names like "back" and "belly", four are vehicles viewpoints rather than actual semantic parts (e.g., *bus-leftside* and *car-frontside*). The remaining two are *aeroplane-stern* and *bottle-body*. Hence, nearly all of the actual semantic parts that emerged according to detection AP also emerge according to human judgments.

Overall, 59 of the semantic parts annotated in PASCAL-Parts emerge according to human judgments. This is substantially more than the 34 that emerged according to detection AP. The reason lie on the fact that it is easier for a filter to count as a semantic part in the human experiment, because it is tested only on the 10 images with the highest activations per object class (fig. 6.7). The AP criterion is more demanding: it takes into account all instances of the part in the dataset, it also counts false-positive detections, and a detection has to be spatially quite accurate to be considered correct (IoU≥ 0.4). This might be a reason why works based on looking at responses on a few images, such as (Zeiler and Fergus, 2014), claimed that filters correspond to semantic parts: they only observed a few strong activations in which this happens, but without examining how the filter behaves over the entire dataset.

Emergence of other semantic part classes. In our human experiment, annotators are free to recognize and name *any* semantic part. Table 6.3 lists the 29 semantic parts that emerge in AlexNet-Object according to our annotators, but that are not annotated in PASCAL-Part. Interestingly, 9 of them concentrate on two object classes: *bicycle* (5) and *bottle* (4). This can be explained by two observations. First, the new parts of the bicycle are mostly sub-parts on the wheel, which is the most discriminative part for the detection of the object (sec. 6.5). And second, the new parts of the bottle are all sub-part of the *bottle-body* part as annotated in PASCAL-Part. As *bottle-body* is essentially the whole object, the network prefers to learn finer-grained, actual parts.

aeroplane	nose	dog	forehead
bicycle	frame, hub, spokes, tire, tube	horse	belly, forehead, shoulder
bird	belly	motorbike	rim
bottle	base, finish, neck, shoulder	person	crotch
bus	hood	pottedplant	pot rim, soil
car	fender, grill	sheep	belly
cat	back	train	engine, headlight, window
cow	belly	tv monitor	-

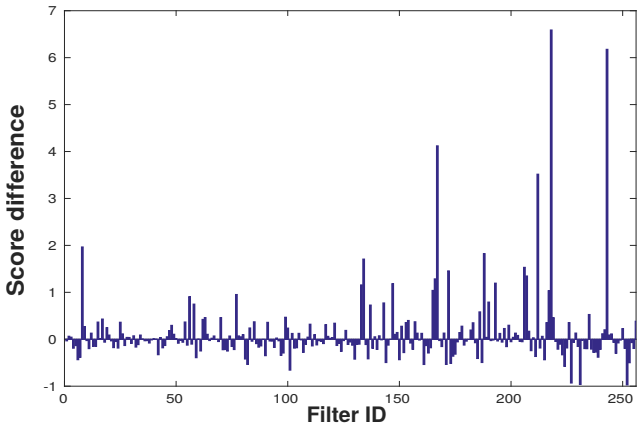
Table 6.3: *List of semantic parts that emerge in AlexNet-Object (layer 5) according to our human experiment, but that are not annotated in the PASCAL-Part dataset.*

Furthermore, two semantic parts often emerging in animal classes are "belly" and "back". Their emergence shows again how the network prefers more localized parts, rather than a larger "torso", as in the PASCAL-Part annotations. Finally, we hypothesize that many of the remaining parts emerge because of their distinctive shapes. For example, *aeroplane-nose* resemble a cone, and *person-crotch* a triangle, *car-fender* a semi-circle and *motorbike-rim* a circle. Moreover, *car-grill* and *train-window* have a characteristic grid pattern.

6.5 Discriminativeness of filters for object recognition

The training procedure of the CNNs we considered maximizes an objective function related to recognition performance, e.g., image classification or object detection. Therefore, the network filters are likely to learn to respond to image patches discriminative for the object classes in the training set. However, these discriminative filters need not correspond to semantic parts. In this section we investigate to which degree the network learns such discriminative filters.

We investigate whether layer 5 filters of AlexNet-Object respond to recurrent discriminative image patches, by assessing how discriminative each filter is for each object class. We use the following measure of the discriminativeness of a filter f_j for a particular object class. First, we record the output score s_i of the network on an input image I_i. Then, we compute a second score s_i^j using the same network but ignoring filter f_j. We achieve this by zeroing the filter's feature map x_j, which means $a_{c,r} = 0$, $\forall a_{c,r} \in x_j$. Finally, we define the discriminativeness of filter f_j as the score difference averaged over the set I of all images of the object class



(a) Score difference δ

(b) Discriminative activations

Figure 6.10: *Discriminative filters for object class car. (a) Shows how discriminative the filters of AlexNet-Object (layer 5) are for car detection (higher values are more discriminative). (b) Shows the activations of the five most discriminative filters.*

$$\delta_j = \frac{1}{|I|} \sum_{I_i \in I} s_i - s_i^j. \tag{6.3}$$

In practice, δ_j indicates how much filter f_j contributes to the classification score of the class. Fig. 6.10a shows an example of these score differences for class car. Only a few filters have high δ values, indicating they are really discriminative for the class. The remaining filters have low values attributable to random noise. We consider f_j to be a discriminative filter if $f_j > 2\sigma$, where σ is the standard deviation of the distribution of δ over the 256 filters in L5. For the car class, only 7 filters are discriminative under this definition. Fig. 6.10b shows an example of the receptive field centers of activations of the top 5 most discriminative filters, which seem to be distributed on several locations of the car. Interestingly, on average over all classes, we find that only 9 out of 256 filters in L5 are discriminative for a particular class. The total number of discriminative filters in the network, over all 16 object classes amounts to 105. This shows that the discriminative filters are largely distributed across different object classes, with little sharing, as also observed by Agrawal et al. (2014). Hence, the network obtains its discriminative power from just a few filters specialized to each class.

Fig. 6.11 shows examples for other classes, where we can observe some other interesting patterns. For example, wheels are extremely discriminative for class bicycle, in contrast to class car, where discriminative filters are more equally distributed across

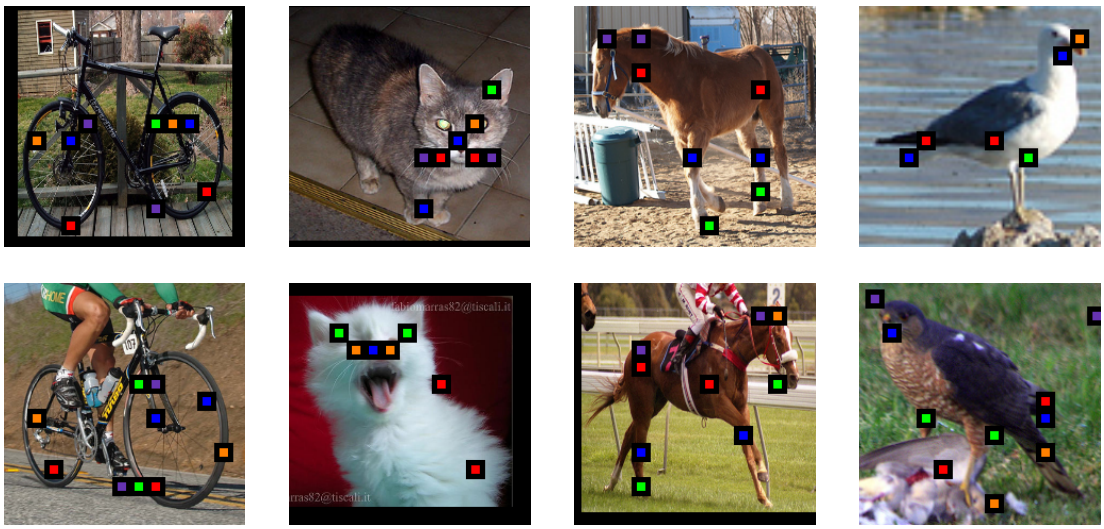


Figure 6.11: *Example activations of the five most discriminative filters for object class* bicyle, cat, horse, bird*, respectively.*

the whole surface of the object. Since wheels are generally big for bicycle images, some filters specialize to subregions of the wheel, such as its bottom area. Another interesting observation is that the discriminativeness of a semantic part might depend on the object class to which it belongs. For example, class cat accumulates most of its most discriminative filters on parts of the head. Interestingly, Parkhi et al. (2011) observed a similar phenomenon with HOG features, where the most discriminative parts of cats and dogs were found to be the heads. On the other hand, class horse tends to prefer parts of the body, such as the legs, devoting very few discriminative filters to the head. Besides firing on subregions of parts, some discriminative filters fire on assemblies of multiple parts or on a part with some neighboring region (e.g., the red filter for class bird is associated with both wing and tail).

How many discriminative filters are also semantic? We categorize now the found discriminative filters into the filter types defined in sec. 6.4, using the data collected in the human experiment. This enables us to determine what fraction of discriminative filters are also semantic, which in turns reveals whether semantic parts are important for recognition. Moreover, as we have defined filter types that go beyond semantic parts, we can obtain a complete list of the filter stimuli that give the network its discriminative power.

Figure 6.12 shows the distribution of discriminative filters over our filter types for three object classes, and the average for all object classes. On average, 40% of the discriminative filters are also semantic, which translates in about 4 out of the 9

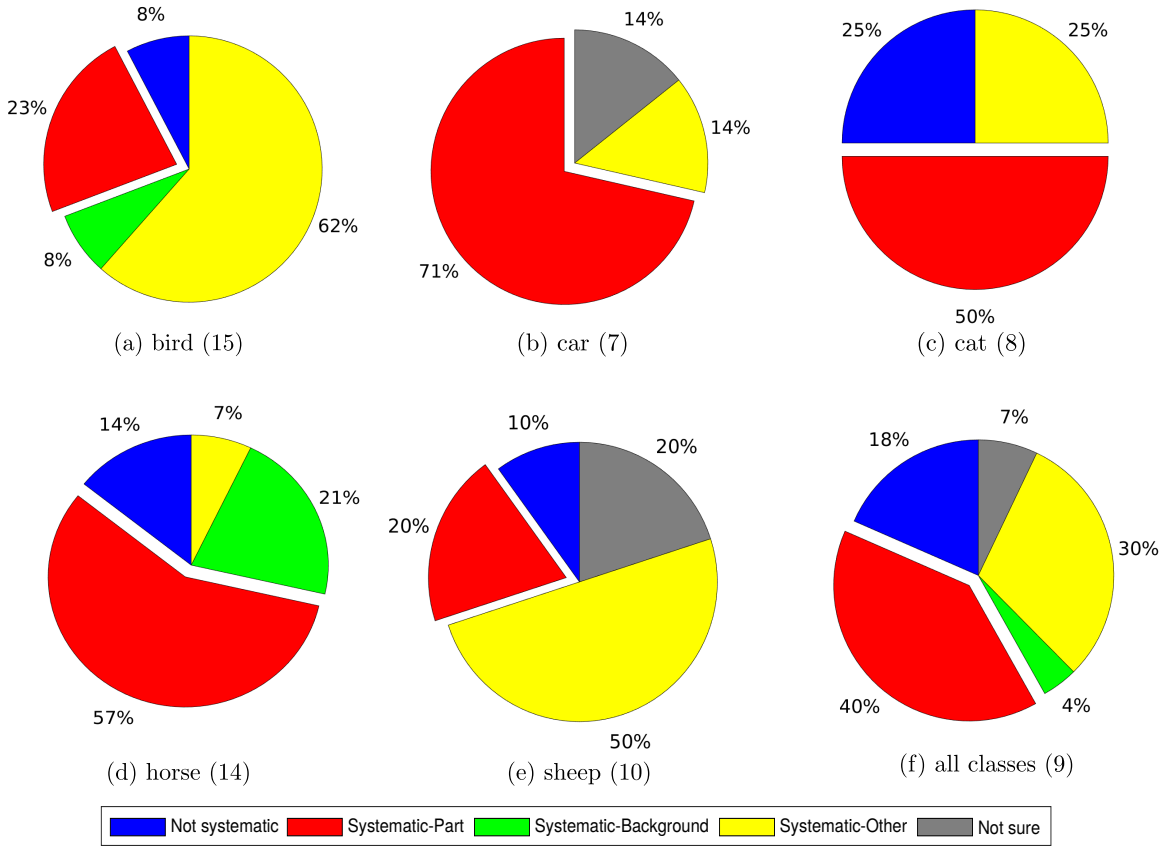


Figure 6.12: *Discriminative filter distributions for bird, car, cat, horse, sheep and for the average of all 16 classes. The number between parenthesis indicates the number of discriminative filters for each case.*

filters that are discriminative for each object class, on average. This is a very high fraction, considering that we found only 7% of all filters to be semantic (fig 6.9). This clearly indicates that the network is using semantic parts as powerful discriminative cues for recognizing object classes. Additionally, about 4% of the filters systematically respond to background patches, and another 30% of the filters systematically respond to some other concept (mostly subregions or assemblies of parts). Finally, 18% of the filters do not correspond to any concept, a massive drop compared to the 78% statistics over all filters (fig 6.9). This distribution confirms our intuition drawn through visual inspection (fig. 6.11): filters that discriminate for the network are often stimulated by some semantic part, but also by other discriminative patches such as subregions of parts.

Figure 6.13: *Examples of input images for the network. The left-most shows the original airplane, while the others shows the same airplane, but with parts blacked out.*

6.6 Discriminativeness of semantic parts for object recognition

In the previous section we investigated how discriminative each filter is for each object class. In this section, instead, we investigate the discriminativeness of semantic parts. We look at how much each part contributes to the classification score of its object class. We measure discriminativeness as in sec. 6.5, but instead of ignoring a specific filter, we now ignore a semantic part. We use the same formulation of eq. (6.3), but with different meanings for j, I and s_i^j. Given a semantic part j, I now indicates all images containing j, and s_i^j is the score given by the network to image I_i with part j blacked out.

We use the segmentation masks available in PASCAL-Parts to set to zero all the pixels of a part j in each input image I_i. In this way, part j is ignored and does not contribute to the classification score of the object, as all convolutional filters output 0 on blacked out regions. The network can only rely on information from the rest of the image. If it is no longer confident about the prediction of the object class, it means that the blacked out part is discriminative for it.

We evaluate the 105 semantic parts of PASCAL-Part (sec. 6.3.2.1). Fig. 6.14 shows results for some examples parts of 9 object classes. Interestingly, similar classes do not necessarily have similar discriminative parts. For example, the most discriminative parts for class car are *door* and *wheel*. But these are not very important for class bus, which is largely discriminated by the part *window*. Moreover, *torso* is very discriminative for some animals (e.g., horse and cow), but less so for others (e.g., cat and dog). We offer two explanations for this phenomenon. First, the network seems to consider discriminative parts that are clearly visible across many instances of the object class. For example, the wheels of a car are often visible in PASCAL images,

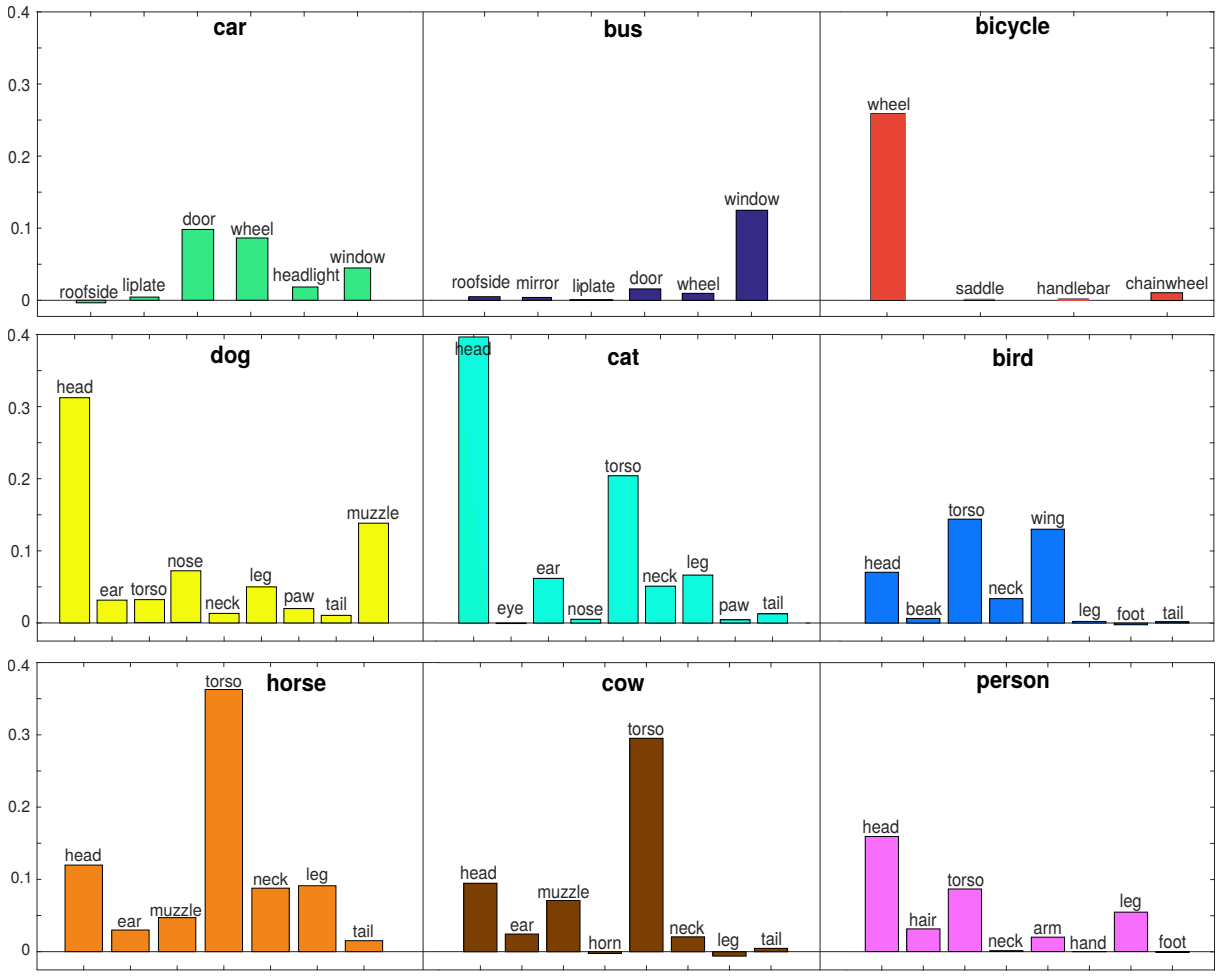


Figure 6.14: *Discriminativeness of PASCAL-Parts for the classification of their objects. The vertical axis indicates the difference δ between the classification score for the original image, and the score for the image after blacking out the part. We report averages over all images in an object class (higher values mean more discriminative).*

while the wheels of a bus are often occluded in the PASCAL dataset. Similarly, many images of pet animals (e.g., cat and dog) are biased towards close-ups (often occluding the body), while images of other animals typically show the whole body (e.g., horse, cow, bird). Second, the network seems to consider more discriminative parts that have lower intra-class variation. For example, the torso is very similar across all horses, while the torso of a dog varies considerably depending on breed and size.

We also observe a strong relation between these quantitative results in fig. 6.14 and the visual results of fig. 6.11. The top most discriminative filters activate on the most discriminative parts. For example, the only discriminative semantic part for bicycle is *wheel*, which is exactly where all the activations of the most discriminative filters are. Analogously, *head* is very discriminative for the class cat, *wing* is discriminative for

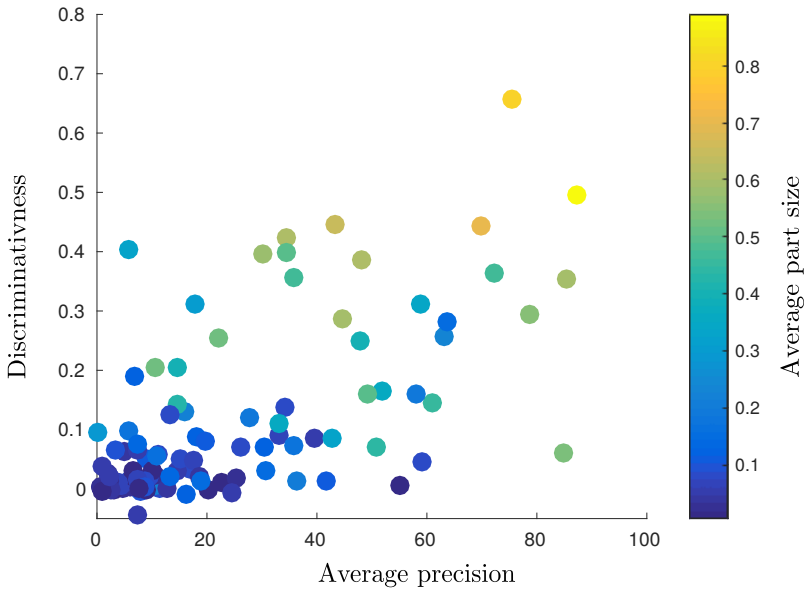


Figure 6.15: *Correlation between the discriminativeness of a semantic part, the average precision results of sec. 6.3.2.2 (mAP, GA, layer 5, Table 6.1) and the average size of a part. The latter is normalized by the average size of its object. Each point corresponds to a different part. Interestingly, these measures are all highly correlated.*

bird and *leg* is somehow discriminative for *horse*.

Correlation to average precision and part size. In this paragraph we look at the correlation between the discriminativeness of a semantic part, the average size of a part and the detection performance results of sec. 6.3.2.2 (AP in table 6.1, GA, layer 5). Results are shown in fig. 6.15. Two interesting facts emerge. First, discriminativeness tends to increase with the average size of a part (very high PPMCC of 0.87) and second, discriminativeness correlates with how much parts emerge in the CNNs according to AP detection performance (PPMCC of 0.65). These are important correlations that support our analysis of sec. 6.3, where we observed that CNNs only learn some semantic parts in their internal representation.

Finally, note how future works that use filters as intermediate part representations (as in Simon et al. (2014); Gkioxari et al. (2015); Simon and Rodner (2015); Xiao et al. (2015); Oquab et al. (2015)) will now be able to exploit these findings to create better models for recognition.

6.7 Conclusions and outlook

We have analyzed the emergence of semantic parts in CNNs. We have investigated whether the network's filters learn to respond to semantic parts. In order to do so, we have associated filter stimuli to semantic parts, using two different quantitative evaluations. In the first one, we have used ground-truth part bounding-boxes to determine how many parts emerge in the CNN for different layers, network architectures and supervision levels. Despite promoting this emergence by providing favorable settings and multiple assists, we found that only 34 out of 105 semantic parts in PASCAL-Part dataset (Chen et al., 2014) emerge in AlexNet (Krizhevsky et al., 2012) finetuned for object detection (Girshick et al., 2014). In the second one, we study how many filters systematically respond to semantic parts for each object class. We found that, on average, 7% of the filters respond to semantic parts, whereas 13% systematically respond to other concepts, such as subregions of parts or background patches. Finally, we have studied how discriminative network filters and semantic parts are for the task of object recognition. We have found that the network discriminates using only a few filters specialized to each class, about 40% of which also correspond to semantic parts, and that most of the parts that emerge in the network are those that are discriminative.

The analysis presented in this chapter can be extended in several ways:

Further analysis on semantic part emergence across network layers. In the analysis of sec. 6.3.2 we observed that higher layers achieve higher semantic part detection performance than lower ones (table 6.1) . While this is true on average, it is not the case for some semantic parts. For example, for *cow-tail* the GA of layer 4 achieves AP of 9.5, which is much higher than the 4.5 AP achieved by the GA of layer 5. In the future we would like to study this phenomena and understand what the part emergence at different layers depends on. For example, we could discover a hierarchy across layers, where more generic parts (e.g., head) are found in a layer and more specific ones (e.g., eye, mouth) in the following ones. Finally, as the filters of different layers have different receptive fields, they are likely to capture different information, hopefully complementary. It would therefore be interesting to train one single GA across all layers, with the freedom to pick any filter in the network. This can potentially improve semantic part detection performance a bit more.

Further study on correlation between part size and part emergence. In sec. 6.3.2.2 we observed a correlation between the average size of a part and its emergence within the network. Our emergence was defined in terms of AP-IoU detection performance. One may argue that this correlation with the part size is due to the fact that considering a part as correctly detected if IoU$\geq$ 0.4 is overly harsh for small parts, as it is very difficult to estimate their spatial extent accurately enough. Therefore, it would be interesting to explore a different strategy to compute AP. An idea would be to use a protocol from landmark-based localization (Zhu and Ramanan, 2012; Belhumeur et al., 2013) and compute distances between centres. Inspired by Zhu and Ramanan (2012) one could consider a detection as correct if the distance between its centre and the centre of the closest part ground-truth bounding box is within a 10% of the diagonal of its parent part/object (e.g., for *dog-nose* one would use the length of the diagonal of *dog-head* bounding-box, while for *dog-head* the whole dog bounding-box).

Investigate discriminativeness of semantic parts according to humans. In sec. 6.5 we measured how discriminative a semantic part is for object class detection, in the context of a CNN. We blacked out semantic part regions and looked at how the CNN scores changed. In the future we would like to understand whether CNNs and humans use the same semantic parts for recognition. We could achieve this by comparing the discriminative parts for a CNN with the discriminative parts for a human. In this context, we could explore discriminativeness for humans in two ways. First, we could measure how long it takes for a human to recognize an object with a part blacked out. The longer it takes, the more discriminative the semantic part is. And second, we could count the number of mistakes a human makes when asked to name an object (with a blacked out part) saw for only an instant (e.g., 1 second). The higher the number of mistakes the human makes, the more discriminative the semantic part.

Chapter 7

Conclusions

In this thesis we tackled the problems of detecting object classes (chapters 3, 4) and their semantic parts (chapters 5, 6). We presented four contributions, two for each task. The first two improved existing object class detection techniques by using context and calibration. The other two contributions investigated semantic part detection in weakly-supervised settings.

We explored these two tasks mostly independently, but evidence in this thesis suggests that these two can benefit from each other. Firstly, sec. 5.5.4 shows that part models can help object class detection, even when added to an already trained strong fully supervised object detector. This is mostly due to the fact that detecting parts helps to localize deformed, occluded and truncated object instances. Secondly, sec. 5.4.3 shows how important objects are to part detection. Object bounding-boxes provide a common coordinate frame from which one can infer important information about the parts. For example, objects photographed under the same viewpoint have the same parts visible and these appear in similar spatial arrangements. Furthermore, parts are always within an object instance and the localization of an object can help to reduce false positive part detections outside the object frame.

So, it is natural to think that object classes and their semantic parts should be modeled together. In the next few paragraphs we present some future directions. First, we present some ideas of how to collect valuable samples for training by avoiding expensive manual annotations, leading to a large collection of class detectors. Then, we present some ideas of how to train a model to detect all instances of an object class and its parts simultaneously. Finally, we present some ideas of how to learn finer object details and enrich object representations using part compositions.

Learn classes incrementally from training instances automatically discovered. In this thesis we experimented with PASCAL-Part, which is the only released dataset with semantic parts annotated with bounding-boxes. Unfortunately, as shown in sec. 6.4, the dataset is not complete and it lacks annotations for many semantic parts. Moreover, while there exist several datasets with object instances annotated (e.g., PASCAL VOC, ImageNet, COCO), these only amount to few thousands instances of few hundred object classes. According to Biederman (1987), our complex word can however be described in terms of more than 30000 object classes. Clearly, more training data is necessary to learn rich models that capture the full spectrum of appearance variations of an object class and its semantic parts. In recent years, this data has been manually annotated and object (or part) detectors have been trained on them in supervised settings. Interestingly, each detector is usually trained from scratch, without prior knowledge of either its class or other classes. This is in contrast to how human learn (Elman, 1993; Krueger and Dayan, 2009). Humans are exposed to visual inputs every day of their life and starting from birth, they progressively accumulate knowledge about objects, their interactions, scenes they appear in, etc. Importantly, humans re-use this acquired knowledge to learn about new objects and to enrich models of objects already encountered. In this process, they transfer knowledge about things they know to new instances.

In this spirit, we should learn object and part models progressively, from a variety of available datasets. Instead of manually annotating new instances, we should exploit the ones available and use them to gradually collect more training samples. We propose that one should start learning from datasets with available location supervision (e.g., bounding boxes or segmentation masks), then harvest images from the web (chapter 5) and images from datasets with labels describing the objects present (but not their locations). Finally, one should mine images labeled by natural language captions. At each step, one should employ the learned knowledge to help learning with less supervision. This strategy mirrors learning in humans, which typically follows an easy-to-hard path (Bengio et al., 2009). Progressively, we can learn more and more classes and refine previous models with additional training instances. To achieve this, as described in sec. 5.6, one should leverage both transfer learning techniques (Pan and Yang, 2010; Rohrbach et al., 2010; Tommasi et al., 2010) and humans in the loop (Branson et al., 2010; Russakovsky et al., 2015b; Papadopoulos et al., 2016).

Joint object-semantic parts detector. In this thesis we showed that object and part

detectors can benefit from each other. A natural question that arises from this observation is: *"what would it take to build a system capable of jointly detecting object classes and their semantic parts?"*

We believe that such a model should alternate optimizing the position of the semantic parts of objects, and the position of objects relative to their semantic parts. This should be addressed with Convolutional Neural Networks, which have achieved impressive results on many visual recognition tasks in the last few years (sec. 2.3). These models are flexible and their structure can be easily adapted to different tasks. A good starting point could be the CNN architecture used in the popular Fast R-CNN object class detector (Girshick, 2015). However, as we now aim to detect an object class and its semantic parts jointly, the network should be trained leveraging several losses. Each loss should capture a different aspect of the model. For example, one could have a loss defined for object class detection, one for part detection, one for part visibility prediction and one for number of parts prediction. The convolutional layers of the model would be shared by all these losses, but each loss would have its own series of fully connected layers. The region-of-interest pooling layer should pool features from object bounding-boxes *and* semantic part bounding-boxes. These will then be input to the parallel fully connected layers designed to process different knowledge (i.e., the different losses). Importantly, one fully connected layer (e.g., part visibility) should pool features from both the object and its parts. This will enforce the network to leverage all the available information and learn from both objects and parts.

We expect a network trained in a similar fashion to achieve state-of-the-art performance for both object class detection and part detection. Moreover, one could develop this idea further and train the network to regress to segmentation masks, potentially improving object segmentation performance as well.

Richer representations. Detecting objects and their semantic parts jointly is a short-term research focus. On the long-term, more complex representations should be exploited to better capture the large intra-class variation that exists within an object class. We should go beyond the standard detection procedures employed today and learn the very fine details describing each object instance. In the following paragraphs we present some research directions that have potential for the future:

Object-Parts relations. One should go beyond learning the location relation between an object and its semantic parts (sec. 5), and learn any kind of correlation between an

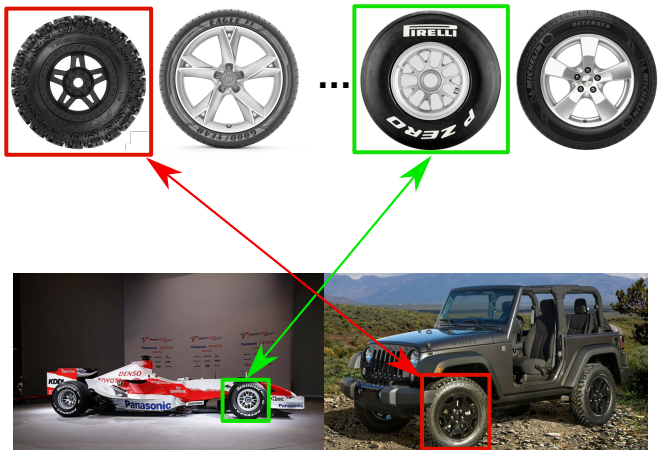


Figure 7.1: *Car-Wheel appearance relation. The top row presents four considerably different wheels, while the bottom row presents two opposite cars. Interestingly, by purely looking at the appearance of these instances, one is able to easily associate what kind of wheel is on each car, and vice-versa.*

object and each of its parts. A simple example includes the relationship between the appearance of an object instance and the appearance of all its semantic parts. These are often correlated; fig. 7.1 shows an example. The top row displays some considerably different car-wheels. These have different thickness, size, color, etc. The bottom row displays two very different cars: a Formula 1 and a Jeep. Interestingly, the association object instance-part instance is very clear: a Formula 1 car usually uses racing wheels, like the Pirelli. Vice-versa, a Pirelli wheel is often on a Formula 1 and not on a Jeep, which instead has off-the-road wheels. With lots of training data available one could learn all these fine-grained appearance relations automatically and use these to better describe objects in images. Instead of today's output: "the image contains a car at location (x,y)", we could be able to produce more complex results like: "the image contains a Formula 1 car at location (x,y). It has Pirelli thick racing tires at location (w,z)". By exploring the relation between an object instance and all its semantic part one can learn to output even richer, more meaningful descriptions.

Photometric consistencies. We should learn photometric object and part consistencies. For example, objects of the class car are usually monochrome, while objects of the class person are very colorful, as people wear different clothes. Moreover, with the helps of semantic parts, we could even learn that a car has a monochromatic body, usually light windows and dark tires. All these information further improve the rich descriptions of an object class that an algorithm can learn to output.

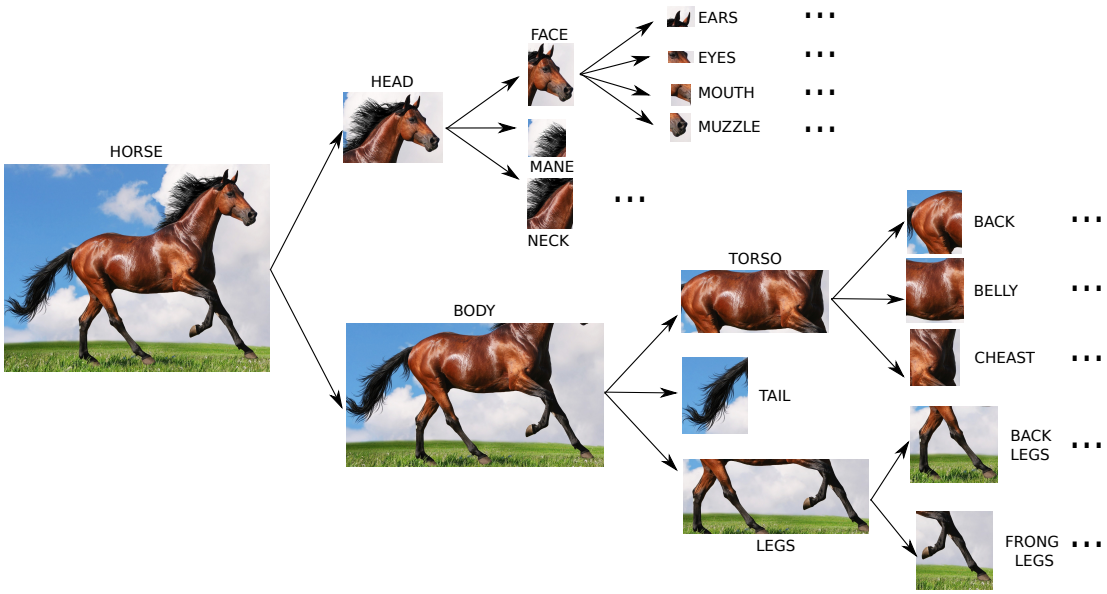


Figure 7.2: *Most current works on semantic parts treat object classes as linear collections of semantic parts. However, these semantic parts are better represented using a hierarchy. This structure is more meaningful and enables richer object representations and easier transfer of information across layers.*

Hierarchy of an object class. In this thesis we described object classes in terms of their semantic parts. One should explore finer details and learn to describe these semantic parts in terms of their sub-parts, their sub-parts in their sub-sub-parts, and so on, leading to a hierarchy of parts for a given object class. Fig. 7.2 shows an example for the object class horse. One should exploit this hierarchy during training and testing, to learn that an eye of a horse is on its face, which usually on the upper part of the body or that a white horse is more likely to have black eyes, rather than blue. Moreover, one could learn to share semantic parts at the same level of the hierarchy across similar object classes. For example, a lynx has ears similar to a cat, eyes similar to a tiger and teeth similar to a panther.

Generate new object instances. Ultimately, one could use the rich body of knowledge learned from objects and parts training instances to generate new object instances. Fig. 7.3 show a very simple example, where we learn some variations of two concepts only: the upper part of a horse and its lower part. On the left we show some representative training samples, while on the right we show some newly generated instances. By leveraging all semantic parts and a rich body of knowledge, one can learn to gener-

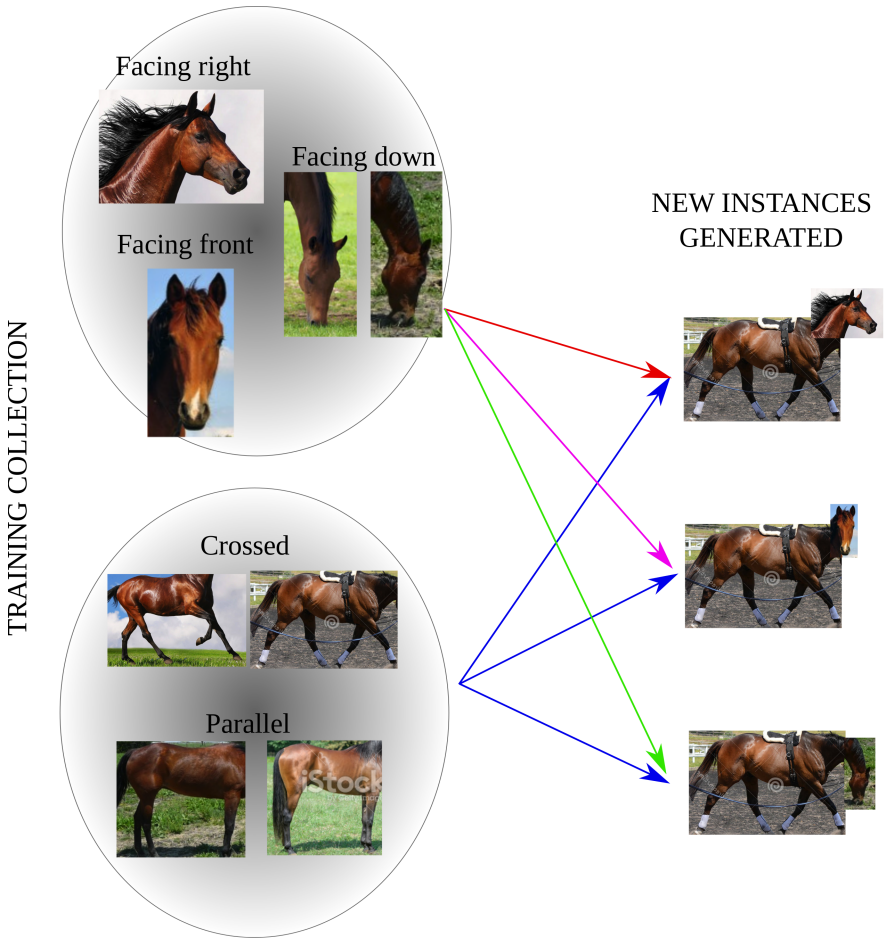


Figure 7.3: *By learning rich object representations and large collections of semantic parts, one can learn to automatically generate new object instances. One could learn that a horse facing right, standing up, and with crossed legs can be associated to a horse with face facing right, facing front and facing down.*

ate accurate object instances. While this example is very easy and straightforward, we believe that in the near future computer vision will be able to progress to a point where a computer will answer any question about an object instance and all its components. It will build a large body of knowledge of the world and it will be able to quickly use this knowledge to automatically transfer information to new unseen instances and to automatically generate realistic new samples.

Bibliography

Agarwal, S., Awan, A., and Roth, D. (2004). Learning to detect objects in images via a sparse, part-based representation. *IEEE Trans. on PAMI*, 20(11):1475–1490.

Aghazadeh, O., Azizpour, H., Sullivan, J., and Carlsson, S. (2012). Mixture component identification and learning for visual recognition. In *ECCV*.

Agrawal, P., Girshick, R., and Malik, J. (2014). Analyzing the performance of multi-layer neural networks for object recognition. In *ECCV*.

Akata, Z., Malinowski, M., Fritz, M., and Schiele, B. (2016). Multi-cue zero-shot learning with strong supervision. In *CVPR*.

Alexe, B., Deselaers, T., and Ferrari, V. (2010). What is an object? In *CVPR*.

Alexe, B., Deselaers, T., and Ferrari, V. (2012). Measuring the objectness of image windows. *IEEE Trans. on PAMI*.

Arbeláez, P., Hariharan, B., Gu, C., Gupta, S., Bourdev, L., and Malik, J. (2012). Semantic segmentation using regions and parts. In *CVPR*.

Aubry, M., Maturana, D., Efros, A., Russell, B., and Sivic, J. (2014a). Seeing 3D chairs: exemplar part-based 2D-3D alignment using a large dataset of cad models. In *CVPR*.

Aubry, M., Russell, B. C., and Sivic, J. (2014b). Painting-to-3D model alignment via discriminative visual elements. *ACM TOG*, 33(2):14.

Aytar, Y. and Zisserman, A. (2012). Enhancing exemplar svms using part level transfer regularization. In *BMVC*.

Azizpour, H. and Laptev, I. (2012). Object detection using strongly-supervised deformable part models. In *ECCV*.

Ballard, D. (1981). Generalizing the hough transform to detect arbitrary shapes. *Pattern recognition*, 13(2):111–122.

Bay, H., Ess, A., Tuytelaars, T., and van Gool, L. (2008). SURF: Speeded up robust features. *CVIU*.

Belhumeur, P., Jacobs, D., Kriegman, D., and Kumar, N. (2013). Localizing parts of faces using a consensus of exemplars. *IEEE Trans. on PAMI*, 35(12):2930–2940.

Belongie, S., Malik, J., and Puzicha, J. (2001). Matching shapes. In *ICCV*, volume 1, pages 454–461.

Bengio, J., Louradour, J., Collobert, R., and Weston, J. (2009). Curriculum learning. In *ICML*.

Berg, A., Berg, T., and Malik, J. (2005). Shape matching and object recognition using low distortion correspondence. In *CVPR*.

Biederman, I. (1987). Recognition-by-components: A theory of human image understanding. *Psychological Review*, 94(2):115–147.

Bilen, H., Pedersoli, M., and Tuytelaars, T. (2014). Weakly supervised object detection with posterior regularization. In *BMVC*.

Bilen, H., Pedersoli, M., and Tuytelaars, T. (2015). Weakly supervised object detection with convex clustering. In *CVPR*.

Branson, S., Wah, C., Schroff, F., Babenko, B., Welinder, P., Perona, P., and Belongie, S. (2010). Visual recognition with humans in the loop. In *ECCV*.

Breiman, L. (2001). Random forests. *ML Journal*, 45(1):5–32.

Caesar, H., Uijlings, J., and Ferrari, V. (2015). Joint calibration for semantic segmentation. In *BMVC*.

Chai, Y., Lempitsky, V., and Zisserman, A. (2013). Symbiotic segmentation and part localization for fine-grained categorization. In *ICCV*.

Chatfield, K., Simonyan, K., Vedaldi, A., and Zisserman, A. (2014). Return of the devil in the details: Delving deep into convolutional networks. In *BMVC*.

Chen, G., Ding, Y., Xiao, J., and Han, T. (2013a). Detection evolution with multi-order contextual co-occurrence. In *CVPR*.

Chen, X. and Gupta, A. (2015). Webly supervised learning of convolutional networks. In *CVPR*.

Chen, X., Mottaghi, R., Liu, X., Fidler, S., Urtasun, R., and Yuille, A. (2014). Detect what you can: Detecting and representing objects using holistic models and body parts. In *CVPR*.

Chen, X., Shrivastava, A., and Gupta, A. (2013b). NEIL: Extracting visual knowledge from web data. In *ICCV*.

Cheng, M.-M., Zhang, Z., L, W. Y., and Torr, P. (2014). Bing: Binarized normed gradients for objectness estimation at 300fps. In *CVPR*.

Choi, M., Lim, J., Torralba, A., and Willsky, A. (2010). Exploiting hierarchical context on a large database of object categories. In *CVPR*.

Choi, M. J., Torralba, A., and Willsky, A. S. (2012). Context models and out-of-context objects. *Patt. Rec. Letters*, 33(7):853–862.

Cinbis, R., Verbeek, J., and Schmid, C. (2013). Segmentation driven object detection with fisher vectors. In *ICCV*.

Cinbis, R., Verbeek, J., and Schmid, C. (2015). Weakly supervised object localization with multi-fold multiple instance learning. *arXiv:1503.00949*.

Crandall, D. J. and Huttenlocher, D. (2006). Weakly supervised learning of part-based spatial models for visual object recognition. In *ECCV*.

Criminisi, A., Shotton, J., and Konukoglu, E. (2011). Decision forests for classification, regression, density estimation, manifold learning and semi-supervised learning. *Microsoft Research Cambridge, Tech. Rep. MSRTR-2011-114*.

Dalal, N. and Triggs, B. (2005a). Histogram of Oriented Gradients for human detection. In *CVPR*.

Dalal, N. and Triggs, B. (2005b). Histogram of Oriented Gradients for Human Detection. In *CVPR*, volume 2, pages 886–893.

Dean, T., Ruzon, M., Segal, M., Shlens, J., Vijayanarasimhan, S., and Yagnik, J. (2013). Fast, accurate detection of 100,000 object classes on a single machine. In *CVPR*.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-fei, L. (2009). ImageNet: A large-scale hierarchical image database. In *CVPR*.

Desai, C., Ramanan, D., and Folkess, C. (2009). Discriminative models for multi-class object layout. In *ICCV*.

Deselaers, T., Alexe, B., and Ferrari, V. (2010). Localizing objects while learning their appearance. In *ECCV*.

Divvala, S., Efros, A., and Hebert, M. (2012). How important are "deformable parts" in the deformable parts model? In *ECCV*.

Divvala, S., Farhadi, A., and Guestrin, C. (2014). Learning everything about anything: Webly-supervised visual concept learning. In *CVPR*.

Dollár, P., Babenko, B., Belongie, S., Perona, P., and Tu, Z. (2008). Multiple component learning for object detection. In *ECCV*.

Donahue, J., Jia, Y., Vinyals, O., Hoffman, J., Zhang, N., Tzeng, E., and Darrell, T. (2013). Decaf: A deep convolutional activation feature for generic visual recognition. *arXiv preprint arXiv:1310.1531*.

Dong, J., Xia, W., Chen, Q., Feng, J., Huang, Z., and Yan, S. (2013). Subcategory-aware object classification. In *CVPR*.

Drayer, B. and Brox, T. (2014). Training deformable object models for human detection based on alignment and clustering. In *ECCV*.

Duembge, L. (2000). Isotonic and concave regression. http://www.imsv.unibe.ch/content/staff/personalhomepages/duembgen/software/isotonicregression/index_eng.html.

Eigen, D., Rolfe, J., Fergus, R., and LeCun, Y. (2013). Understanding deep architectures using a recursive convolutional network. In *ICLR workshop*.

Elman, J. L. (1993). Learning and development in neural networks: The importance of starting small. *Cognition*, 48(1):71–99.

Endres, I., Shih, K., Jiaa, J., and Hoiem, D. (2013). Learning collections of part models for object recognition. In *CVPR*.

Everingham, M., Van Gool, L., Williams, C., Winn, J., and Zisserman, A. (2007). The PASCAL Visual Object Classes Challenge 2007 Results.

Everingham, M., Van Gool, L., Williams, C. K. I., Winn, J., and Zisserman, A. (2008). The PASCAL Visual Object Classes Challenge 2008 (VOC2008) Results. http://host.robots.ox.ac.uk/pascal/VOC/voc2008/workshop/everingham_det.pdf.

Everingham, M., Van Gool, L., Williams, C. K. I., Winn, J., and Zisserman, A. (2010). The PASCAL Visual Object Classes (VOC) Challenge. *IJCV*.

Everingham, M., Van Gool, L., Williams, C. K. I., Winn, J., and Zisserman, A. (2012). The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Results. http://www.pascal-network.org/challenges/VOC/voc2012/workshop/index.html.

Everingham, M., Van Gool, L., Williams, C. K. I., Winn, J., and Zisserman, A. (2013). The PASCAL Visual Object Classes Challenge evaluation server. http://host.robots.ox.ac.uk:8080/leaderboard/main_bootstrap.php.

Farrell, R., Oza, O., Zhang, N., Morariu, V., Darrell, T., Davis, L. S., et al. (2011). Birdlets: Subordinate categorization using volumetric primitives and pose-normalized appearance. In *ICCV*.

Felzenszwalb, P., Girshick, R., and McAllester, D. (2010a). Cascade object detection with deformable part models. In *CVPR*.

Felzenszwalb, P., Girshick, R., McAllester, D., and Ramanan, D. (2010b). Object detection with discriminatively trained part based models. *IEEE Trans. on PAMI*, 32(9).

Felzenszwalb, P. and Huttenlocher, D. (2005). Pictorial structures for object recognition. *IJCV*, 61(1).

Felzenszwalb, P. and McAllester, D. (2011). Object detection grammars. In *ICCV Workshops*.

Felzenszwalb, P., McAllester, D., and Ramanan, D. (2008). A discriminatively trained, multiscale, deformable part model. In *CVPR*.

Felzenszwalb, P. F. and Schwartz, J. D. (2007). Hierarchical matching of deformable shapes. In *CVPR*.

Fergus, R., Fei-Fei, L., Perona, P., and Zisserman, A. (2005). Learning object categories from google's image search. In *ICCV*.

Fergus, R., Perona, P., and Zisserman, A. (2003a). Object class recognition by unsupervised scale-invariant learning. In *CVPR*.

Fergus, R., Perona, P., and Zisserman, A. (2003b). Object class recognition by unsupervised scale-invariant learning. In *CVPR*.

Fergus, R., Perona, P., and Zisserman, A. (2004). A visual category filter for Google images. In *ECCV*. Springer-Verlag.

Ferrari, V., Fevrier, L., Jurie, F., and Schmid, C. (2008). Groups of adjacent contour segments for object detection. *IEEE Trans. on PAMI*, 30(1):36–51.

Ferrari, V., Jurie, F., and Schmid, C. (2010). From images to shape models for object detection. *IJCV*, 87(3).

Ferrari, V., Tuytelaars, T., and van Gool, L. (2006). Object detection by contour segment networks. In *ECCV*.

Gall, J. and Lempitsky, V. (2009). Class-specific hough forests for object detection. In *CVPR*.

Gao, T., Packer, B., and Koller, D. (2011). A segmentation-aware object detection model with occlusion handling. In *CVPR*.

Gavves, E., Fernando, B., Snoek, C., Smeulders, A. W., and Tuytelaars, T. (2013). Fine-grained categorization by alignments. In *ICCV*.

Gidaris, S. and Komodakis, N. (2015). Object detection via a multi-region and semantic segmentation-aware CNN model. In *ECCV*.

Girshick, R. (2015). Fast R-CNN. In *ICCV*.

Girshick, R., Donahue, J., Darrell, T., and Malik, J. (2014). Rich feature hierarchies for accurate object detection and semantic segmentation. In *CVPR*. Software available at https://github.com/rbgirshick/rcnn/.

Girshick, R., Felzenszwalb, P., and McAllester, D. (2012). Discriminatively trained deformable part models, release 5. http://people.cs.uchicago.edu/ rbg/latent-release5/.

Girshick, R. and Malik, J. (2013). Training deformable part models with decorrelated features. In *ICCV*.

Girshick, R. B., Felzenszwalb, P. F., and McAllester, D. (2011). Object detection with grammar models. In *NIPS*.

Gkioxari, G., Girshick, R., and Malik, J. (2015). Actions and attributes from wholes and parts. In *ICCV*.

Goering, C., Rodner, E., Freytag, A., and Denzler, J. (2014). Nonparametric part transfer for fine-grained recognition. In *CVPR*.

Gogate, V. and Dechter, R. (2004). A complete anytime algorithm for treewidth. In *AUAI*.

Gonzalez-Garcia, A., Modolo, D., and Ferrari, V. (2016). Do semantic parts emerge in convolutional neural networks? In *submitted to IJCV*.

Gonzalez-Garcia, A., Vezhnevets, A., and Ferrari, V. (2015). An active search strategy for efficient object class detection. In *CVPR*.

Gronat, P., Obozinski, G., Sivic, J., and Pajdla, T. (2013). Learning and calibrating per-location classifiers for visual place recognition. In *CVPR*.

Gu, C., Arbeláez, P., Lin, Y., Yu, K., and Malik, J. (2012). Multi-component models for object detection. In *ECCV*.

Gu, C. and Ren, X. (2010). Discriminative mixture-of-templates for viewpoint classification. In *ECCV*.

Gupta, S., Girshick, R., Arbeláez, P., and Malik, J. (2014). Learning rich features from RGB-D images for object detection and segmentation. In *ECCV*.

Hariharan, B., Arbeláez, P., Girshick, R., and Malik, J. (2014). Simultaneous detection and segmentation. In *ECCV*.

Hariharan, B., Arbeláez, P., Girshick, R., and Malik, J. (2015). Hypercolumns for object segmentation and fine-grained localization. In *CVPR*.

Hariharan, B., Malik, J., and Ramanan, D. (2012). Discriminative decorrelation for clustering and classification. In *ECCV*.

Harzallah, H., Jurie, F., and Schmid, C. (2009). Combining efficient object localization and image classification. In *ICCV*.

He, K., Zhang, X., Ren, S., and Sun, J. (2014). Spatial pyramid pooling in deep convolutional networks for visual recognition. In *ECCV*.

Heitz, G. and Koller, D. (2008). Learning spatial context: Using stuff to find things. In *ECCV*.

Hoiem, D., Efros, A. A., and Hebert, M. (2008). Putting objects in perspective. *IJCV*, 80(1):3–15.

Hosang, J., Omran, M., Benenson, R., and Schiele, B. (2015). Taking a deeper look at pedestrians. In *CVPR*.

Indyk, P. and Motwani, R. (1998). Approximate nearest neighbors: towards removing the curse of dimensionality. In *ACM Multimedia*.

Jaakkola, T. and Haussler, D. (1999). Exploiting generative models in discriminative classifiers. In *NIPS*.

Jia, Y. (2013). Caffe: An open source convolutional architecture for fast feature embedding. http://caffe.berkeleyvision.org/.

Joon Oh, S., Benenson, R., Fritz, M., and Schiele, B. (2015). Person recognition in personal photo collections. In *ICCV*.

Juneja, M., Vedaldi, A., Jawahar, C., and Zisserman, A. (2013). Blocks that shout: Distinctive parts for scene classification. In *CVPR*.

Kalogeiton, V., Ferrari, V., and Schmid, C. (2016). Analysing domain shift factors between videos and images for object detection. *IEEE Trans. on PAMI*.

Karayev, S., Baumgartner, T., Fritz, M., and Darrell, T. (2012). Timely object recognition. In *NIPS*.

Karayev, S., Fritz, M., and Darrell, T. (2014). Anytime recognition of objects and scenes. In *CVPR*, pages 572–579.

Karpathy, A., Shetty, S., Toderici, G., Leung, T., Sukthankar, R., and Fei-Fei, L. (2014). Large-scale video classification with convolutional neural networks. In *CVPR*.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). ImageNet classification with deep convolutional neural networks. In *NIPS*.

Krueger, K. A. and Dayan, P. (2009). Flexible shaping: How learning in small steps helps. *Cognition*, 110(3):380–394.

Kuettel, D. and Ferrari, V. (2012). Figure-ground segmentation by transferring window masks. In *CVPR*.

Lampert, C. H., Blaschko, M. B., and Hofmann, T. (2008). Beyond sliding windows: Object localization by efficient subwindow search. In *CVPR*.

Lazebnik, S., Schmid, C., and Ponce, J. (2006). Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories. In *CVPR*.

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.

Leibe, B., Leonardis, A., and Schiele, B. (2004). Combined object categorization and segmentation with an implicit shape model. In *Workshop on Statistical Learning in Computer Vision, ECCV*.

Leibe, B. and Schiele, B. (2003). Interleaved object categorization and segmentation. In *BMVC*, volume 2, pages 264–271.

Lenc, K. and Vedaldi, A. (2015). Understanding image representations by measuring their equivariance and equivalence. In *CVPR*.

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

Li, Q., Wu, J., and Tu, Z. (2013). Harvesting mid-level visual concepts from large-scale internet images. In *CVPR*.

Lin, D., Shen, X., Lu, C., and Jia, J. (2015). Deep lac: Deep localization, alignment and classification for fine-grained recognition. In *CVPR*.

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

Liu, C., Yuen, J., and Torralba, A. (2009). Nonparametric scene parsing: Label transfer via dense scene alignment. In *CVPR*.

Liu, J. and Belhumeur, P. N. (2013). Bird part localization using exemplar-based models with enforced pose and subcategory consistency. In *ICCV*.

Liu, J., Kanazawa, A., Jacobs, D., and Belhumeur, P. (2012). Dog breed classification using part localization. In *ECCV*.

Liu, J., Li, Y., and Belhumeur, P. N. (2014). Part-pair representation for part localization. In *ECCV*.

Long, J., Shelhamer, E., and Darrell, T. (2015). Fully convolutional networks for semantic segmentation. In *CVPR*.

Lowe, D. (2004). Distinctive image features from scale-invariant keypoints. *IJCV*, 60(2):91–110.

Mahendran, A. and Vedaldi, A. (2015). Understanding deep image representations by inverting them. In *CVPR*.

Maji, S., Berg, A., and Malik, J. (2008). Classification using intersection kernel support vector machines is efficient. In *CVPR*.

Maji, S., Berg, A. C., and Malik, J. (2013). Efficient classification for additive kernel svms. *IEEE Trans. on PAMI*, 35(1):66–77.

Maji, S. and Malik, J. (2009). Object detection using a max-margin hough tranform. In *CVPR*.

Malinowski, M., Rohrbach, M., and Fritz, M. (2015). Ask your neurons: A neural-based approach to answering questions about images. In *ICCV*.

Malisiewicz, T. (2011). Ensemble of exemplar-svms, implementation. https://github.com/quantombone/exemplarsvm.

Malisiewicz, T., Gupta, A., and Efros, A. (2011). Ensemble of exemplar-svms for object detection and beyond. In *ICCV*.

Manen, S., Guillaumin, M., and Van Gool, L. (2013). Prime object proposals with randomized prim's algorithm. In *ICCV*.

Mikolajczyk, K., Leibe, B., and Schiele, B. (2005). Local features for object class recognition. In *ICCV*.

Mikolajczyk, K. and Schmid, C. (2004). Scale & affine invariant interest point detectors. *IJCV*, 1(60):63–86.

Mitchell, M. (1998). *An introduction to genetic algorithms*. MIT press.

Modolo, D. and Ferrari, V. (2016). Learning semantic part-based models from Google Images. In *submitted to TPAMI*.

Modolo, D., Vezhnevets, A., and Ferrari, V. (2015a). Context forest for object class detection. In *BMVC*.

Modolo, D., Vezhnevets, A., Russakovsky, O., and Ferrari, V. (2015b). Joint calibration of ensemble of exemplar svms. In *CVPR*.

Moosman, F., Triggs, B., and Jurie, F. (2006). Fast discriminative visual codebook using randomized clustering forests. In *NIPS*.

Murphy, K., Torralba, A., and Freeman, W. T. (2003). Using the forest to see the trees: A graphical model relating features, objects, and scenes. In *NIPS*.

Niculescu-Mizil, A. and Caruana, R. (2005). Predicting good probabilities with supervised learning. In *ICML*.

Oliva, A. and Torralba, A. (2001). Modeling the shape of the scene: A holistic representation of the spatial envelope. *IJCV*, 42(3):145–175.

Opelt, A., Pinz, A., and Zisserman, A. (2006). A boundary-fragment-model for object detection. In *ECCV*.

Oquab, M., Bottou, L., Laptev, I., and Sivic, J. (2015). Is object localization for free? Weakly-supervised learning with convolutional neural networks. In *CVPR*.

Ott, P. and Everingham, M. (2011). Shared parts for deformable part-based models. In *CVPR*.

Pan, S. J. and Yang, Q. (2010). A survey on transfer learning. *IEEE Trans. on Knowledge and Data Engineering*, 22:1345–1359.

Pandey, M. and Lazebnik, S. (2011). Scene recognition and weakly supervised object localization with deformable part-based models. In *ICCV*.

Papadopoulos, D., Uijlings, J., Keller, F., and Ferrari, V. (2016). We don't need no bounding-boxes: Training object class detectors using only human verification. In *CVPR*.

Papadopoulos, D. P., Clarke, A. D. F., Keller, F., and Ferrari, V. (2014). Training object class detectors from eye tracking data. In *ECCV*.

Park, D., Ramanan, D., and Fowlkes, C. (2010). Multiresolution models for object detection. In *ECCV*.

Parkhi, O., Vedaldi, A., Jawahar, C. V., and Zisserman, A. (2011). The truth about cats and dogs. In *ICCV*.

Parkhi, O. M., Vedaldi, A., Zisserman, A., and Jawahar, C. (2012). Cats and dogs. In *CVPR*.

Parzen, E. (1962). On the estimation of a probability density function. *Annals of Mathematical Statistics*, 33(3):1065–1076.

Pearson, K. (1895). Note on regression and inheritance in the case of two parents. *Proceedings of the Royal Society of London*, 58:240–242.

Pedersoli, M., Vedaldi, A., and Gonzales, J. (2011). A coarse-to-fine approach for fast deformable object detection. In *CVPR*.

Peng, X. and Schmid, C. (2016). Multi-region two-stream R-CNN for action detection. In *ECCV*.

Pepik, B., Stark, M., Gehler, P., and Schiele, B. (2013). Occlusion patterns for object class detection. In *CVPR*.

Perronnin, F., Sánchez, J., and Mensink, T. (2010). Improving the fisher kernel for large-scale image classification. In *ECCV*, pages 143–156.

Pinheiro, P., Lin, T.-Y., Collobert, R., and Dollár, P. (2016). Learning to refine object segments. In *ECCV*.

Platt, J. (1999). Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods. *Advances in large margin classifiers*.

Prest, A., Leistner, C., Civera, J., Schmid, C., and Ferrari, V. (2012). Learning object class detectors from weakly annotated video. In *CVPR*.

Rabinovich, A., Vedaldi, A., Galleguillos, C., Wiewiora, E., and Belongie, S. (2007). Objects in context. In *ICCV*.

Razavian, A., Azizpour, H., Sullivan, J., and Carlsson, S. (2014). CNN features off-the-shelf: An astounding baseline for recognition. In *DeepVision workshop at CVPR*.

Redmon, J., Divvala, S., Girshick, R., and Farhadi, A. (2016). You only look once: Unified, real-time object detection. In *CVPR*.

Rematas, K., Ritschel, T., Fritz, M., and Tuytelaars, T. (2014). Image-based synthesis and re-synthesis of viewpoints guided by 3D models. In *CVPR*.

Ren, S., He, K., Girshick, R., and Sun, J. (2015). Faster R-CNN: Towards real-time object detection with region proposal networks. In *NIPS*.

Rohrbach, M., Stark, M., Szarvas, G., Gurevych, I., and Schiele, B. (2010). What helps where? And why? Semantic relatedness for knowledge transfer. In *CVPR*.

Rosenfeld, A. and Weinshall, D. (2011). Extracting foreground masks towards object recognition. In *ICCV*.

Rother, C., Kolmogorov, V., and Blake, A. (2004). Grabcut: interactive foreground extraction using iterated graph cuts. *SIGGRAPH*, 23(3):309–314.

Rubinstein, M., Joulin, A., Kopf, J., and Liu, C. (2013). Unsupervised joint object discovery and segmentation in internet images. In *CVPR*.

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A., and Fei-Fei, L. (2015a). ImageNet large scale visual recognition challenge. *IJCV*.

Russakovsky, O., Li, L.-J., and Fei-Fei, L. (2015b). Best of both worlds: human-machine collaboration for object annotation. In *CVPR*.

Russakovsky, O., Lin, Y., Yu, K., and Fei-Fei, L. (2012). Object-centric spatial pooling for image classification. In *ECCV*.

Russel, B. and Torralba, A. (2008). LabelMe: a database and web-based tool for image annotation. *IJCV*, 77(1-3):157–173.

Russell, B., Torralba, A., Liu, C., Ferugs, R., and Freeman, W. (2007). Object recognition by scene alignment. In *NIPS*.

Schroff, F., Criminisi, A., and Zisserman, A. (2011). Harvesting image databases from the web. *IEEE Trans. on PAMI*, 33(4):754–766.

Sermanet, P., Eigen, D., Zhang, X., Mathieu, M., Fergus, R., and LeCun, Y. (2014). Overfeat: Integrated recognition, localization and detection using convolutional networks. In *ICLR*.

Shih, K. J., Mallya, A., Singh, S., and Hoiem, D. (2015). Part localization using multi-proposal consensus for fine-grained categorization. In *BMVC*.

Shotton, J., Blake, A., and Cipolla, R. (2005). Contour-based learning for object detection. In *ICCV*.

Shrivastava, A. and Gupta, A. (2016). Contextual priming & feedback for faster R-CNN. In *ECCV*.

Shrivastava, A., Gupta, A., and Girshick, R. (2016). Training region-based object detectors with online hard example mining. In *CVPR*.

Shrivastava, A., Malisiewicz, T., Gupta, A., and Efros, A. (2011). Data-driven visual similarity for cross-domain image matching. In *SIGGRAPH*.

Shu, G., Dehghan, A., Oreifej, O., Hand, E., and Shah, M. (2012). Part-based multiple-person tracking with partial occlusion handling. In *CVPR*.

Simon, M. and Rodner, E. (2015). Neural activation constellations: Unsupervised part model discovery with convolutional networks. In *ICCV*.

Simon, M., Rodner, E., and Denzler, J. (2014). Part detector discovery in deep convolutional neural networks. In *ACCV*.

Simonyan, K., Vedaldi, A., and Zisserman, A. (2014). Deep inside convolutional networks: Visualising image classification models and saliency maps. In *ICLR workshop*.

Simonyan, K. and Zisserman, A. (2015). Very deep convolutional networks for large-scale image recognition. In *ICLR*.

Singh, S., Gupta, A., and Efros, A. (2012). Unsupervised discovery of mid-level discriminative patches. In *ECCV*.

Siva, P. and Xiang, T. (2011). Weakly supervised object detector learning with model drift detection. In *ICCV*.

Sivic, J., Russell, B., Efros, A., Zisserman, A., and Freeman, W. (2005). Discovering objects and their location in images. In *ICCV*.

Song, H., Girshick, R., Jegelka, S., Mairal, J., Harchaoui, Z., and Darell, T. (2014a). On learning to localize objects with minimal supervision. In *ICML*.

Song, H., Lee, Y., Jegelka, S., and Darell, T. (2014b). Weakly-supervised discovery of visual pattern configurations. In *NIPS*.

Song, H., Zickler, S., Althoff, T., Girshick, R., Fritz, M., Geyer, C., Felzenszwalb, P., and Darrell, T. (2012). Sparselet models for efficient multiclass object detection. In *ECCV*.

Song, S. and Xiao, J. (2014). Sliding shapes for 3D object detection in depth images. In *ECCV*.

Song, Z., Chen, Q., Huang, Z., Hua, Y., and Yan, S. (2011). Contextualizing object detection and classification. In *CVPR*.

Sun, M. and Savarese, S. (2011). Articulated part-based model for joint object detection and pose estimation. In *ICCV*.

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

Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., and Fergus, R. (2014). Intriguing properties of neural networks. In *ICLR*.

Taigman, Y., Yang, M., Ranzato, M., and Wolf, L. (2014). Deepface: Closing the gap to human-level performance in face verification. In *CVPR*.

Tian, Y., Luo, P., Wang, X., and Tang, X. (2015). Pedestrian detection aided by deep learning semantic tasks. In *CVPR*.

Tighe, J. and Lazebnik, S. (2010). Superparsing: Scalable nonparametric image parsing with superpixels. In *ECCV*.

Tighe, J. and Lazebnik, S. (2013). Finding things: Image parsing with regions and per-exemplar detectors. In *CVPR*.

Tommasi, T., Orabona, F., and Caputo, B. (2010). Safety in numbers: Learning categories from few examples with multi model knowledge transfer. In *CVPR*.

Torralba, A. (2003). Contextual priming for object detection. *IJCV*, 53(2):153–167.

Torralba, A. and Efros, A. (2011). An unbiased look on dataset bias. In *CVPR*.

Toshev, A. and Szegedy, C. (2014). DeepPose: Human pose estimation via deep neural networks. In *CVPR*.

Tsai, D., Jing, Y., Liu, Y., Rowley, H., Ioffe, S., and Rehg, J. (2011). Large-scale image annotation using visual synset. In *ICCV*.

Tsai, Y.-H., Hamsici, O. C., and Yang, M.-H. (2015). Adaptive region pooling for object detection. In *CVPR*.

Uijlings, J. R. R., van de Sande, K. E. A., Gevers, T., and Smeulders, A. W. M. (2013). Selective search for object recognition. *IJCV*.

Ukita, N. (2012). Articulated pose estimation with parts connectivity using discriminative local oriented contours. In *CVPR*.

Van de Sande, K., Uijlings, J., Gevers, T., and Smeulders, A. (2011). Segmentation as selective search for object recognition. In *ICCV*.

Van De Sande, K. E. A., Fontijne, D., Stokman, H., Snoek, C., and Smeulders, A. (2013). University of Amsterdam and Euvision technologies at ilsvrc2013 (ILSVRC). www.image-net.org/challenges/LSVRC/2013/slides/ILSVRC2013-UvA-Euvision-web.pdf.

Van De Sande, K. E. A., Gevers, T., and Snoek, C. G. M. (2010). Evaluating color descriptors for object and scene recognition. *IEEE Trans. on PAMI*, 32(9):1582–1596.

Vedaldi, A., Gulshan, V., Varma, M., and Zisserman, A. (2009). Multiple kernels for object detection. In *ICCV*.

Vedaldi, A., Mahendran, S., Tsogkas, S., Maji, S., Girshick, R., Kannala, J., Rahtu, E., Kokkinos, I., Blaschko, M. B., Weiss, D., et al. (2014). Understanding objects in detail with fine-grained attributes. In *CVPR*.

Vedaldi, A. and Zisserman, A. (2009). Structured output regression for detection with partial truncation. In *NIPS*.

Vedaldi, A. and Zisserman, A. (2010). Efficient additive kernels via explicit feature maps. In *CVPR*.

Vezhnevets, A. and Ferrari, V. (2014). Associative embeddings for large-scale knowledge transfer with self-assessment. In *CVPR*.

Vijayanarasimhan, S. and Grauman, K. (2008). Keywords to visual categories: Multiple-instance learning for weakly supervised object categorization. In *CVPR*.

Vinyals, O., Toshev, A., Bengio, S., and Erhan, D. (2015). Show and tell: A neural image caption generator. In *CVPR*.

Viola, P. and Jones, M. (2001). Rapid object detection using a boosted cascade of simple features. In *CVPR*, pages 511–518.

Viola, P. and Jones, M. (2004). Robust real-time face detection. *IJCV*, 57(2):137–154.

Wah, C., Branson, S., Perona, P., and Belongie, S. (2011). Multiclass recognition and part localization with humans in the loop. In *ICCV*.

Wang, C., Ren, W., Zhang, J., Huang, K., and Maybank, S. (2015). Large-scale weakly supervised object localization via latent category learning. *IEEE Transactions on Image Processing*, 24(4):1371–1385.

Wang, J., Markert, K., and Everingham, M. (2009a). Learning models for object recognition from natural language descriptions. In *BMVC*.

Wang, J. and Yuille, A. (2015). Semantic part segmentation using compositional model combining shape and appearance. In *CVPR*.

Wang, T., He, X., and Barnes, N. (2013a). Learning structured hough voting for joint object detection and occlusion reasoning. In *CVPR*.

Wang, X., Han, T. X., and Yan, S. (2009b). An HOG-LBP human detector with partial occlusion handling. In *ICCV*.

Wang, X., Yang, M., Zhu, S., and Lin, Y. (2013b). Regionlets for generic object detection. In *ICCV*, pages 17–24. IEEE.

Wei, S.-E., Ramakrishna, V., Kanade, T., and Sheikh, Y. (2016). Convolutional pose machines. In *CVPR*.

Xiao, J., Hays, J., Ehinger, K., Oliva, A., and Torralba, A. (2010). SUN database: Large-scale scene recognition from Abbey to Zoo. In *CVPR*.

Xiao, T., Xu, Y., Yang, K., Zhang, J., Peng, Y., and Zhang, Z. (2015). The application of two-level attention models in deep convolutional neural network for fine-grained image classification. In *CVPR*.

Xu, K., Ba, J., Kiros, R., Cho, K., Courville, A., Salakhutdinov, R., Zemel, R., and Bengio, Y. (2015). Show, attend and tell: Neural image caption generation with visual attention. In *ICML*.

Yan, J., Lei, Z., Wen, L., and Li, S. (2014). The fastest deformable part model for object detection. In *CVPR*.

Yang, J., Price, B., Cohen, S., and Yang, M.-H. (2014). Context driven scene parsing with attention to rare classes. In *CVPR*.

Yosinski, J., Clune, J., Bengio, Y., and Lipson, H. (2014). How transferable are features in deep neural networks? In *NIPS*.

Zadrozny, B. and Elkan, C. (2001a). Learning and making decisions when costs and probabilities are both unknown. In *ACM SIGKDD*.

Zadrozny, B. and Elkan, C. (2001b). Obtaining calibrated probability estimates from decision trees and naive bayesian classifiers. In *ICML*.

Zadrozny, B. and Elkan, C. (2002). Transforming classifier scores into accurate multi-class probability estimates. In *ACM SIGKDD*.

Zeiler, M. D. and Fergus, R. (2014). Visualizing and understanding convolutional networks. In *ECCV*.

Zepeda, J. and Perez, P. (2015). Exemplar svms as visual feature encoders. In *CVPR*.

Zhang, J., Huang, K., Yu, Y., and Tan, T. (2011). Boosted local structured HOG-LBP for object localization. In *CVPR*.

Zhang, J., Marszalek, M., Lazebnik, S., and Schmid, C. (2007). Local features and kernels for classification of texture and object categories: a comprehensive study. *IJCV*, 73(2):213–238.

Zhang, L., Lin, L., Liang, X., and He, K. (2016). Is faster R-CNN doing well for pedestrian detection? In *ECCV*.

Zhang, N., Donahue, J., Girshick, R., and Darrell, T. (2014a). Part-based R-CNNs for fine-grained category detection. In *ECCV*.

Zhang, N., Farrell, R., and Darrell, T. (2012). Pose pooling kernels for sub-category recognition. In *CVPR*.

Zhang, N., Farrell, R., Iandola, F., and Darrell, T. (2013). Deformable part descriptors for fine-grained recognition and attribute prediction. In *ICCV*.

Zhang, N., Paluri, M., Ranzato, M., Darrell, T., and Bourdev, L. (2014b). Panda: Pose aligned networks for deep attribute modeling. In *CVPR*.

Zhang, Y., Sohn, K., Villegas, R., Pan, G., and Lee, H. (2015). Improving object detection with deep convolutional networks via bayesian optimization and structured prediction. In *CVPR*.

Zhou, B., Khosla, A., Lapedriza, A., Oliva, A., and Torralba, A. (2015). Object detectors emerge in deep scene CNNs. In *ICLR*.

Zhou, B., Lapedriza, A., Xiao, J., Torralba, A., and Oliva, A. (2014). Learning deep features for scene recognition using places database. In *NIPS*.

Zhu, L., Chen, Y., Yuille, A., and Freeman, W. (2010). Latent hierarchical structural learning for object detection. In *CVPR*.

Zhu, M., Atanasov, N., Pappas, G., and Daniilidis, K. (2014). Active aeformable part models inference. In *ECCV*.

Zhu, X. and Ramanan, D. (2012). Face detection, pose estimation, and landmark localization in the wild. In *CVPR*.

Zhu, X., Vondrick, C., Ramanan, D., and Fowlkes, C. (2012). Do we need more training data or better models for object detection? In *BMVC*.

Zilberstein, S. (1996). Using anytime algorithms in intelligent systems. *AI magazine*, 17(3):73.

Zitnick, C. L. and Dollár, P. (2014). Edge boxes: Locating object proposals from edges. In *ECCV*.