Beyond Bounding Boxes: Precise Localization of Objects in Images

by

Bharath Hariharan

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Doctor of Philosophy

in

Computer Science

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor Jitendra Malik, Chair
Professor Trevor Darrell
Professor Bruno Olshausen

Summer 2015

Beyond Bounding Boxes: Precise Localization of Objects in Images

Abstract

Beyond Bounding Boxes: Precise Localization of Objects in Images

by

Bharath Hariharan

Doctor of Philosophy in Computer Science

University of California, Berkeley

Professor Jitendra Malik, Chair

Object recognition in computer vision comes in many flavors, two of the most popular being object detection and semantic segmentation. Object detection systems detect every instance of a category in an image, and coarsely localize each with a bounding box. Semantic segmentation systems assign category labels to pixels, thus providing pixel-precise localization but failing to resolve individual instances of the category. We argue for a richer output: recognition systems should detect individual instances of a category and provide pixel precise segmentations for each, a task we call Simultaneous Detection and Segmentation or SDS. We describe approaches to this task that leverage convolutional neural networks for precise localization. We also show that the techniques we develop are also effective for other tasks such as segmenting the parts of a detected object or localizing its keypoints. These are our first steps towards a recognition system that goes beyond category labels and coarse bounding boxes to precise, detailed descriptions of objects in images.

To my grandfather.

Contents

List of Figures

List of Tables

Acknowledgments

Thanks to Jitendra Malik, not only for mentoring me and showing me how to do research, but also for giving me the space and freedom to try out my own ideas and at the same time being there when I needed guidance. The relaxed, supportive environment fostered by Jitendra brought out the best in me. Thanks also to Ross Girshick and Pablo Arbeláez for showing me how to be a good empirical scientist: the idea of evaluating one's ideas in the harsh light of benchmarks and metrics, and the notion of framing careful experiments that tease out what is happening is something I learned from them. Thanks to Trevor and Alyosha for providing the counter-weight to Jitendra and allowing me to learn from three computer vision behemoths instead of one.

I want to thank all of Malik-group: Subhransu, Lubomir, Chunhui, Jon, Georgia, Saurabh, Pulkit, Abhishek, Panna, Shubham, Ross, Pablo, Katerina and Phillip, for all the insightful discussions in the bay: my understanding and opinions have been framed mostly by these discussions, and I have learned unmeasurably from them.

I want to thank the awesome circle of friends I had: Neeraja, Vivek, Saurabh, Radhika, Gautam, TD, Nitesh,Shubham, Anurag, Nikunj, Shromona, Sreeta, Aishwarya, and the much wider, yet still close net of friends all over the world: Divya, Ekta, Swati, Navneet, Pragun and many many others, for making life bearable when research was hard. Finally, I am glad to have had a most supportive and loving family who were always there for me in times of need.

Chapter 1

Introduction

In the abstract, the goal of computer vision is to enable computers to “understand” images. But what does it mean to understand an image? We can take human-level understanding as the gold standard. As humans, with a brief look at the image in Figure 1.1, we know that we are looking at a coffee-shop scene. We notice several coffee cups on the table, one of them full; we notice keys, a fork on a plate with breadcrumbs, spoons and used sugar packets. We can not only identify the objects in the scene, but also locate each object precisely: a server who chanced upon this scene would have no trouble in picking up the empty cups and saucers and cleaning up the table.

Suppose we want to create a robotic server that would clear the table. Let us assume that this robot is equipped with a vision system. What should this vision system be able to do? It should definitely be able to see that it is standing in front of a table that has cups, saucers, spoons, forks and plates on it. This is the task of image classification: labeling an image with all the object categories that are present in the image. This is a very well studied problem that has seen tremendous progress

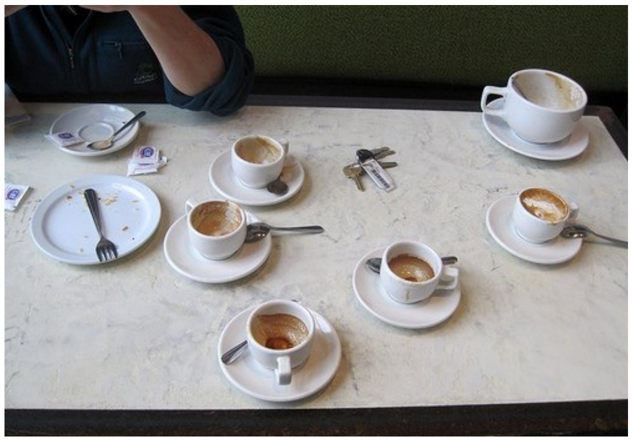

Figure 1.1: Understanding an image involves not only identifying the presence or absence of object categories, but also precisely localizing each instance of each category, identifying its pose and so on.

in recent years [41, 54].

But knowing that there are cups and saucers is not enough; our robotic server needs to know where these cups and saucers are. Such need for the localization of objects has prompted research in the task of object detection: the goal is to mark out bounding boxes around each object of a particular category in the image. A detection system would find each cup and saucer and localize it with a bounding box. Again, we have seen large strides in object detection performance [24] on well established benchmarks [18].

However, the bounding boxes output by an object detection system are very coarse. To accurately grasp a cup, the robot needs to know the precise boundaries of the cup. We thus want the vision system to go beyond bounding boxes and segment out each detected cup precisely. For this, one might turn to a semantic segmentation system. Semantic segmentation is also a well studied problem, and has as its goal the assignment of category labels to each pixel in the image. Such a system would thus mark a bunch of pixels as cup pixels, some others as table pixels, still others as plate pixels and so on. However, this still doesn't solve our problem, because now we have lost all notion of instances: the server now does not know how many cups there are, let alone the boundaries of each individual cup.

What we want instead is an algorithm that will detect each instance of the "cup" category, and precisely localize each detected instance with a segmentation. We call this task "Simultaneous Detection and Segmentation" or SDS. Note that robotic servers are not the only potential consumers of such technology. As a very different example, consider a graphics application where one wants to automatically crop objects out of one scene and paste them into another scene. Here again, knowing the boundaries of the object is crucial. For the most part, this thesis presents algorithms for and results on SDS.

Of course, it would be wrong to say that SDS is all that we need. Even the precise segmentation may not be enough for our robotic server: it may want to grasp the cup by its handle, which requires knowing where the handle of the cup is. A shopping application that looks at an image of a person and automatically figures out the shirt he is wearing will also need to segment out not just the person but also the shirt. Thus we may want the segmentation to be finer-grained, requiring the algorithm to label each part of the object. A related task is one of estimating the pose of the object by locating some keypoints or joint locations, which can be useful for understanding what a person is doing. Alternatively, we may be interested in segmentation, but we may care specifically of the boundaries of the object. This is especially true of graphics applications, where cropping operations that do not respect boundaries can be visually jarring. In the last part of this thesis, we explore how ideas from our SDS algorithm can be carried over to these other tasks.

What all these tasks have in common is the need to go beyond category labels and coarse bounding boxes towards richer, more useful descriptions of the detected objects. Our work is a first step towards this ambitious goal.

Most of the work in this thesis has been published before in two papers [31, 30]. This thesis combines the two and extends them with additional results.

1.1 Prior work on localization

Bringing instances into segmentation

Several researchers have incorporated instance-level information, often from object detectors, to guide the segmentation process. Yang et al.[61] use object detections from the deformable parts model [21] to segment the image, pasting figure-ground masks and reasoning about their relative depth ordering. Arbeláez et al.[2] use poselet detections [7] as features to score region candidates, in addition to appearance-based cues. Ladicky et al.[42] use object detections as higher order potentials in a CRF-based segmentation system: all pixels in the foreground of a detected object are encouraged to share the category label of the detection. In addition, their system is allowed to switch off these potentials by assigning a true/false label to each detection. This system was extended by Boix et al.[5] who added a global, image-level node in the CRF to reason about the categories present in the image, and by Kim et al.[38] who added relationships between objects. In more recent work, Tighe et al.[55] use exemplar object detectors to segment out the scene as well as individual instances.

Improving localization in object detection

There has also been work on localizing detections better using segmentation. Parkhi et al.use color models from predefined rectangles on cat and dog faces to do GrabCut [50] and improve the predicted bounding box [49]. Dai and Hoiem generalize this to all categories and use instance and category appearance models to improve detection [14]. These approaches do well when the objects are coherent in color or texture. This is not true of many categories such as people, where each object can be made of multiple regions of different appearance.

An alternative to doing segmentation *post facto* is to use segmentation to generate object proposals which are then classified. The proposals may be used as just bounding boxes [52] or as region proposals [11, 1]. These proposals incorporate both the consistency of appearance in an object as well as the possibility of having multiple disparate regions for each object. State-of-the-art detection systems [24] and segmentation systems [10] are now based on these methods.

While many of these approaches do produce segmentation hypotheses, detection systems typically discard the segmentation and evaluate on bounding box-based metrics [24], and semantic segmentation systems marginalize out instance information to produce pixel labeling [10]. A notable exception to this is the work of Tighe et al. [55] who do produce segmentation hypotheses for each detected object, and evaluate performance on the same metric that we use in this thesis.

In many of these approaches, segmentation is used only to localize the detections better. Other authors have explored using segmentation as a stronger cue. Fidler et al.[22] use the output of the state-of-the-art semantic segmentation approach of [10] to score detections better. Mottaghi [48] uses detectors based on non-rectangular patches to both detect and segment objects.

Labeling parts and localizing keypoints

A highly active area of research in computer vision is the task of labeling keypoint locations of people. A long-standing line of work in this area is based on the notion of part detectors connected by springs [62]. Early work used simple features such as Histograms of Oriented Gradients [15, 62, 17, 26]. However current best performers for pose estimation are based on CNNs [43]. Toshev and Szegedy [57] use a CNN to regress to keypoint locations. Tompson et al. [56] show large improvements over state-of-the-art by predicting a heatmap for each keypoint, where the value of the heatmap at a location is the probability of the keypoint at that location. These algorithms show results in the setting where the rough location of the person is known. Yang and Ramanan [62] propose a more realistic setting where the location of the person is not known and one has to both detect the person and identify his/her keypoints. Gkioxari et al. [28] show some results in this setting using HOG-based detectors, but in their later work [27] show large gains using CNNs.

Related to pose estimation is the task of segmenting out the different parts of a person, a task typically called “object parsing”. Yamaguchi et al. [60, 59] parse fashion photographs into clothing items. There has also been work on parsing pedestrians [4, 45]. Ionescu et al. [35] jointly infer part segmentations and pose. However, the setting is typically tightly cropped bounding boxes of pedestrians, while we are interested in the completely unconstrained case.

1.2 Evaluation metrics

A key requirement for progress in an empirical field such as computer vision is the availability of standard evaluation metrics and benchmarks to evaluate progress. Keeping this in mind, here we standardize the evaluation metrics for the tasks that we are interested in.

Our evaluation metrics are modifications of the standard object detection metric, which we describe next. In object detection, for each image, the algorithm produces a set of bounding boxes together with category labels and scores. Each such bounding box is a *detection*. We also have ground truth bounding boxes with category labels annotated on the test set. We rank the detections in descending order of scores. Then, we go down this list of detections and for each predicted bounding box, we compute the intersection-over-union (or IoU) of this box with all ground truth boxes of this category. If this IoU is greater than an overlap threshold (typically 0.5), and if this ground truth object has not been covered by a higher-scoring detection, we mark this detection as correct and the corresponding ground truth box as covered. Otherwise, the detection is marked as incorrect. With this labeling of all the detections, one can compute a precision-recall or PR curve. The area under this curve is reported as Average Precision or AP.

In the SDS task, we expect the algorithm to also produce a segmentation hypothesis for each detection. We assume, for evaluation, that the test set is annotated with the ground truth segmentation for each object. We go down the ranked list of detections as before, but now we compute the intersection-over-union of the predicted *segmentation* with the *segmentation* of each ground truth instance of this category. We then check if this overlap is greater than a threshold and label the detections as correct or incorrect as before. Finally we report the area under the precision recall

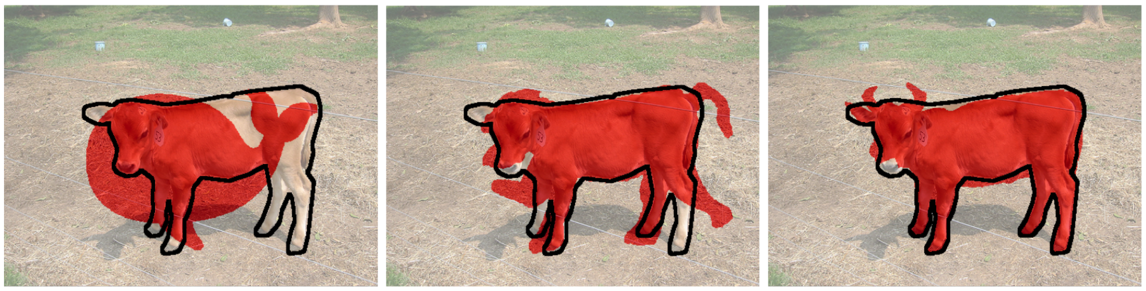

Figure 1.2: Three different detections (in red) overlapping a given ground truth (outlined in black). The segment IoU's are (from left to right): 0.51, 0.72, 0.91. The box IoU's are 1.00, 0.78, 0.91.

curve. We call this metric AP^r where the r stands for *region*

Figure 1.2 visualizes what segment IoU means. Each image shows a candidate detection (in red) and the ground truth (outlined in black). The leftmost detection gets the location of the object correct, but the predicted shape is very wrong. It achieves an IoU of slightly more than 0.5. The second detection gets both the shape and location reasonably correct and gets a segment IoU of 0.72. The last detection is correct to a very high degree of accuracy and gets a segment IoU of 0.91. However, at this level of IoU we are approaching human-level accuracy, and the error in the ground truth annotations might be of this order. We therefore use 0.7 as the overlap threshold for computing AP^r in all experiments in this thesis. We will, however, also report AP^r at 0.5.

Note that the corresponding bounding box overlaps for the three detections in Figure 1.2 are 1.00, 0.78 and 0.91 respectively. The worst segmentation thus gets the best bounding box overlap. This again indicates that one can estimate the bounding box of an object very accurately and still be quite wrong about its precise extent.

We can extend this metric for the other tasks that we are interested in. For part labeling,we assume that together with the segmentation hypothesis, the algorithm also produces a part label for each pixel predicted to be in the foreground. We also assume that the ground truth is also labeled with part labels. We then modify the definition of IoU so that a pixel counts in the intersection only if the predicted part label is also correct. We then compute the area under the PR curve as before. We call this AP$^r_{part}$.

For keypoint localization, we use the APK metric proposed by [62]. Again, this is a modification of the object detection metric. Each keypoint is evaluated independently. Each detection comes with a keypoint prediction and a score for the keypoint, which combines the score of the detection and the confidence that the keypoint is visible. The detections are ranked in descending order of scores, and a detection is considered correct if the keypoint is visible and the predicted location lies within a threshold distance (expressed in units of the torso height) of the true keypoint location. As always the metric reported is the area under the PR curve.

If we are interested in precise boundaries, then the AP^r metric may not be satisfactory because it gives undue importance to pixels in the interior of the object. As an alternative, we compute a bipartite matching between the boundaries of a predicted segmentation and a ground truth segmentation, as in [47]. Using this matching, we can compute an intersection over union, where the matched contour pixels are considered as belonging in the intersection, and all other pixels are outside the intersection. We can then count the detection as correct if this intersection over union exceeds a threshold. As before, we compute a PR curve and compute the area under the curve, which we call AP^{bdry}.

Chapter 2

Classifying Bottom-up Region Proposals

In this chapter we present a system for SDS that first uses bottom-up contour and texture cues to come up with a few thousand candidate segmentations per image and then uses powerful classifiers to classify them. We describe each step of the pipeline below, and then evaluate it experimentally.

2.1 Proposing an initial pool of candidates

The space of possible outputs for an object detection system is huge: every possible box in an image is a potential detection. For the SDS task, this space is the space of possible segments and is orders of magnitude larger. Exhaustively sifting through each possibility and evaluating it with a heavyweight classifier is extremely expensive. Instead, most current detection and segmentation approaches rely on first using simple bottom-up cues of color and texture coherence to come up with a small number (a few thousand per image) of candidate boxes or segments that can then be scored by a classifier.

Methods of producing such bottom-up proposals vary depending on whether they produce bounding boxes or regions, and on the kind of cues they use. Selective Search [52] computes segmentation hierarchies in multiple color spaces and uses the resulting segments as object proposals. MCG [1] creates such a segmentation hierarchy at multiple scales and additionally combinatorially combines pairs and triplets of adjacent regions. CPMC [11] and GOP [40] produce region hypotheses by using graph cuts with multiple foreground and background seeds. Edge boxes [63] scores box proposals by computing contour strength straddling the box boundaries.

Since we are interested in the AP^r metric, we care about segments, and not just boxes. Keeping our task in mind, we use candidates from MCG [1] for this paper. This approach significantly outperforms all competing approaches on the object level Jaccard index metric, which measures the average best overlap achieved by a candidate for a ground truth object. In our experiments we find that simply switching to MCG from Selective Search [52] improves detection AP slightly (by 0.7 points), justifying this choice.

We use the proposals from MCG as is. MCG starts by computing a segmentation hierarchy at multiple image resolutions, which are then fused into a single multiscale hierarchy at the finest scale. Then candidates are produced by combinatorially grouping regions from all the single scale hierarchies and from the multiscale hierarchy. The candidates are ranked based on simple features such as size and location, shape and contour strength.

2.2 Extracting features from candidates and classifying them

Once we have a pool of candidate segments, the next step is to classify them. In recent years, convolutional neural networks or CNNs have come to be regarded as the classifier of choice. CNNs are multilayer perceptrons which contain convolution and pooling layers. A convolution layer can be either seen as a layer convolving an input feature map using a set of filters, or as a layer of locally connected neurons sharing their weights. A pooling layer simply subsamples its input by taking the max, average or some other statistic over a neighborhood. The basic idea of a convolutional network was proposed by Fukushima [23], and they were first trained with backpropagation by, among others, Rumelhart et al. [51] and LeCun et al. [43]. Krizhevsky et al. [41] showed impressive results on the ImageNet image classification challenge [16] using these models, following which several researchers showed impressive performance on a wide range of tasks using CNNs.

Of particular interest is the work of Girshick et al.[24], who showed large gains in object detection by using CNN-derived features to classify object proposals. In our work we take their object detection system, called R-CNN, and adapt it to the SDS task.

R-CNN takes a pool of bounding box proposals, crops and warps them to a fixed size, and passes each through the convolutional network. Features from one of the top layers are then extracted out and fed into a linear SVM. Girshick et al. train the CNN on ImageNet Classification and then finetune the network on the PASCAL detection set. For finetuning they took bounding boxes from Selective Search, padded them, cropped them and warped them to a square and fed them to the network. Bounding boxes that overlap with the ground truth by more than 50% were taken as positives and other boxes as negatives. The class label for each positive box was taken to be the class of the ground truth box that overlaps the most with the box. The network thus learned to predict if the bounding box overlaps highly with a ground truth bounding box. We are working with MCG instead of Selective Search, so we train a similar object detection network, finetuned using bounding boxes of MCG regions instead of Selective Search boxes.

For the SDS task, we can now use this network finetuned for detection to extract feature vectors from MCG bounding boxes (for this work,we use features from the penultimate fully connected layer, or *fc7*). However these feature vectors do not contain any information about the actual region foreground, and so will be ill-equipped to decide if the region overlaps highly with a ground truth segmentation or not. To get around this, we start with the idea used by Girshick et al.for their experiment on semantic segmentation: we extract a second set of features from the region by feeding it the cropped, warped box, but with the background of the region masked out (with the mean image.) Concatenating these two feature vectors together gives us the feature vector we use. (In their experiments Girshick et al.found both sets of features to be useful). This method of extracting features out of the region is the simplest way of extending the object detection system to the SDS task and forms our baseline. We call this feature extractor **A**.

The network we are using above has been finetuned to classify bounding boxes, so its use in extracting features from the region foreground is suboptimal. Several neurons in the network may be focussing on context in the background, which will be unavailable when the network is fed the region foreground. This suggests that we should use a different network to extract the second set of features: one that is finetuned on the kinds of inputs that it is going to see. We therefore finetune another network (starting again from the net trained on ImageNet) which is fed as input cropped, padded bounding boxes of MCG regions with the background masked out. Because this region sees the actual foreground, we can actually train it to predict region overlap instead, which is what we care about. Therefore we change the labeling of the MCG regions to be based on segmentation overlap of the region with a ground truth region (instead of overlap with bounding box). We call this feature extractor **B**.

The previous strategy is still suboptimal, because the two networks have been trained in isolation, while at test time the two feature sets are going to be combined and fed to the classifier. This suggests that one should train the networks jointly. We formalize this intuition as follows. We create a neural network with the architecture shown in Figure 2.1. This architecture is a single network with two pathways. The first pathway operates on the cropped bounding box of the region (the "box" pathway) while the second pathway operates on the cropped bounding box with the background masked (the "region" pathway). The two pathways are disjoint except at the very final classifier layer, which concatenates the features from both pathways. Both these pathways individually have the same architecture as that of Krizhevsky et al.Note that both A and B can be seen as instantiations of this architecture, but with different sets of weights. A uses the same network parameters for both pathways. For B, the box pathway gets its weights from a network finetuned separately using bounding box overlap, while the region pathway gets its parameters from a network finetuned separately using region overlap.

Instead of using the same network in both pathways or training the two pathways in isolation, we now propose to train it as a whole directly. We use segmentation overlap as above. We initialize the box pathway with the network finetuned on boxes and the region pathway with the network finetuned on regions, and then finetune the entire network. At test time, we discard the final classification layer and use the output of the penultimate layer, which concatenates the features from the two pathways. We call this feature extractor **C**.

Region classification

We use the features from the previous step to train a linear SVM. We first train an initial SVM using ground truth as positives and regions overlapping ground truth by less than 20% as negative. Then we re-estimate the positive set: for each ground truth we pick the highest scoring MCG candidate that overlaps by more than 50%. Ground truth regions for which no such candidate exists (very few in number) are discarded. We then retrain the classifier using this new positive set. This training procedure corresponds to a multiple instance learning problem where each ground truth defines a positive bag of regions that overlap with it by more than 50%, and each negative region is its own bag. We found this training to work better than using just the ground truth as positives.

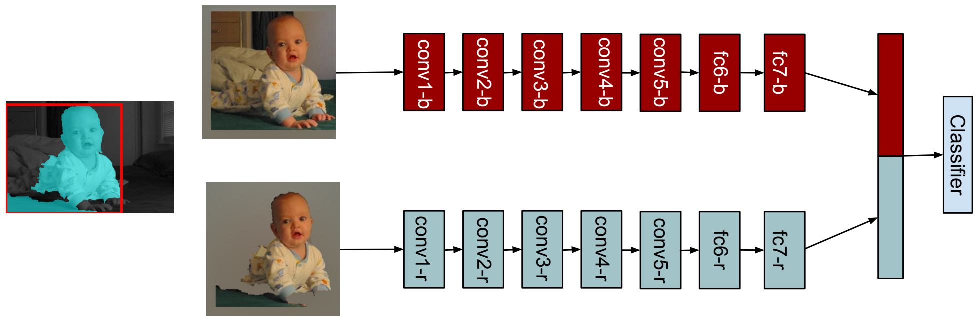

Figure 2.1: Left: The region with its bounding box. Right: The architecture that we train for **C**. The top pathway operates on cropped boxes, and the bottom operates on region foregrounds.

At test time we use the region classifiers to score each region. Because there may be multiple overlapping regions, we do a strict non-max suppression using a region overlap threshold of 0. This is because while the bounding box of two objects can in fact overlap, their pixel support in the image typically shouldn't. Post NMS, we work with only the top 20,000 detections for each category (over the whole dataset) and discard the rest for computational reasons. We confirmed that this reduction in detections has no effect on the AP^r metric.

2.3 Experiments and results

We use the segmentation annotations from SBD [29] to train and evaluate. We train all systems on PASCAL VOC 2012 train. For all training and finetuning of the network we use the recently released Caffe framework [36].

Results on AP^r

Table 2.3 shows results on the AP^r metric for an overlap threshold of 0.5 and 0.7 respectively on PASCAL VOC 2012 val (ground truth segmentations are not available for test).

1. **O_2P** uses features and regions from Carreira et al.[10], which is the state-of-the-art in semantic segmentation. We train region classifiers on these features and do NMS to get detections. This baseline gets a mean AP^r of 25.2% at 0.5 overlap and 11.6% at 0.7 overlap%.

2. **A** is our most naive feature extractor. It uses MCG candidates and features from the bounding box and region foreground, using a single CNN finetuned using box overlaps. It achieves a mean AP^r of 42.9% at 0.5 overlap and 18.0% at 0.7, a large jump over O_2P. This mirrors gains in object detection observed by Girshick et al.[24], although since O_2P is not designed for this task the comparison is somewhat unfair.

3. **B** is the result of finetuning a separate network exclusively on region foregrounds with labels defined by region overlap. This gives a large jump of the AP^r metric (of about 4 percentage points at 0.5 overlap and 3.9 percentage points at 0.7 overlap).

4. **C** is the result of training a single large network with two pathways. There is a clear gain over using two isolated networks: on both metrics we gain about 1 percentage point.

A paired sample t-test indicates that each of the above improvements are statistically significant at the 0.05 significance level.

The left part of Figure 2.3 plots the improvement in mean AP^r over **A** as we vary the threshold at which a detection is considered correct. Each of our improvements increases AP^r across all thresholds, indicating that we haven't overfit to a particular regime.

Clearly we get significant gains over both our naive baseline as well as O2P. However, prior approaches that reason about segmentation together with detection might do better on the AP^r metric. To see if this is the case, we compare to the SegDPM work of Fidler et al.[22]. SegDPM combined DPMs [21] with O_2P [10] and achieved a 9 point boost over DPMs in classical object detection. For this method, only the bounding boxes are available publicly, and for some boxes the algorithm may choose not to have associated segments. We therefore compute an upper bound of its performance by taking each detection, considering all MCG regions whose bounding box overlaps with the detection by more than 70%, and selecting the region which best overlaps a ground truth.

Since SegDPM detections are only available on PASCAL VOC2010 val, we restrict our evaluations only to this set. Our upper bound on SegDPM has a mean AP^r of **31.3**, whereas **C** achieves a mean AP^r of **48.3**.

We visualize some example detections in Figure 2.2. Our system is able to detect and segment individual instances of objects and seems to be able to handle occlusion and pose variation. We also visualize some false positives. All three false positives are mislocalizations: the system has detected the object, but it has chosen a region that does not overlap very well.

We investigate the error modes more deeply using a method similar to that proposed by Hoiem et al. [33]: we divide the error modes into 3 different kinds and measure the impact of each:

1. *Mislocalization*: We compute the AP^r for an overlap threshold of 0.1 and an overlap threshold of 0.5 (or 0.7). Detections marked true positive under the more lenient overlap threshold and false positive under the harsher threshold are considered mislocalized. We consider two oracles that can deal with mislocalized detections: the first oracle suppresses these detections, and the second oracle corrects them. To get the AP^r for the first oracle, we can simply drop the mislocalized detections. The AP^r for the second oracle is simply the AP^r obtained at the overlap threshold of 0.1. The difference between the performance of these oracles and the AP^r we actually get at 0.5 overlap is a measure of how much mislocalization hurts our system.

2. *Confusion with similar categories*: To evaluate the impact of false positives coming from objects from other similar categories, we group the categories into supergroups corresponding

to animals, transport categories and indoor categories. We then suppress false positive detections that overlap by more than 0.1 with instances from other categories of the supergroup. The difference between the performance of this oracle, and the true performance, gives us a measure of the impact of confusion with similar categories.

3. *Confusion with background*: All other false positives arise from firings on the background. The AP^r when these false positives are suppressed gives us an indication of the impact of false positives on generic background.

The right half of Figure 2.3 plots the impact of these three error modes as described above (averaged over all categories). For localization, the dark shaded bar indicates the impact measured by the oracle that suppresses the mislocalized detections, while the light shaded bar uses the oracle that corrects mislocalization.

As can be seen, the impacts of confusion with similar categories or background are relatively small. This means that our region classification system is quite good at distinguishing between the category of interest and similar categories or background. Mislocalization is, however, a much larger problem: mislocalization errors set us back by about 15 percentage points. Correcting mislocalization will therefore likely provide a huge boost in performance.

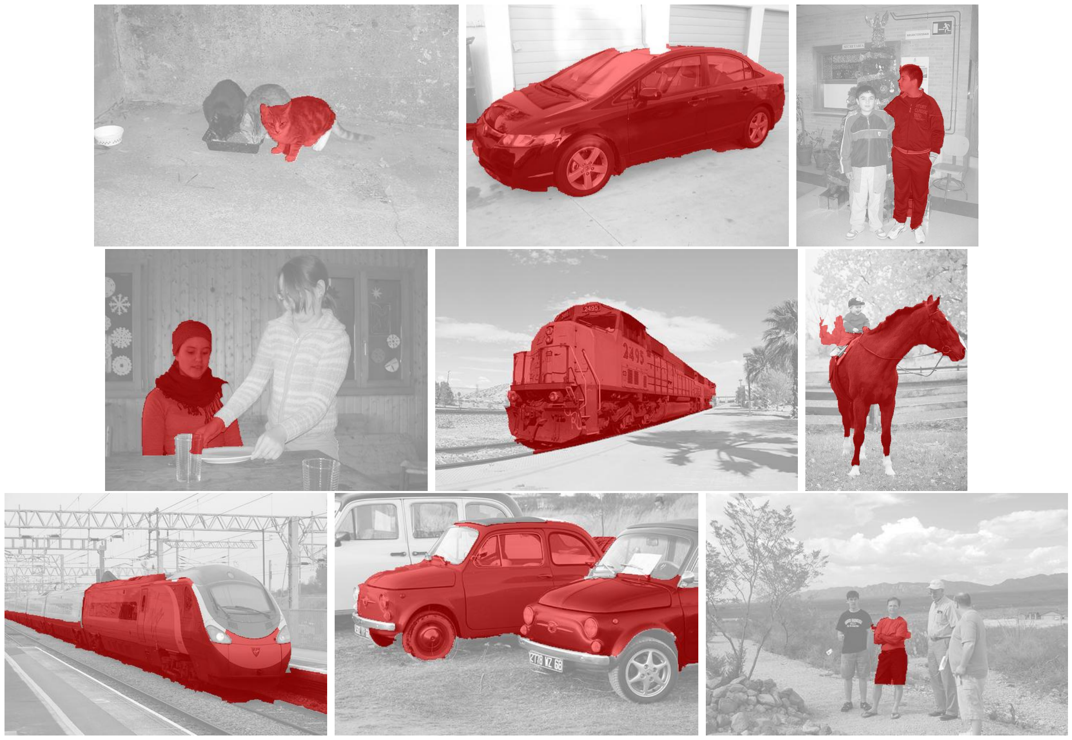

Figure 2.2: Example detections. Our classifier is able to pick out objects from clutter, separating out individual instances and can handle occlusion and large pose variations. The last row shows false positives. Predicted category (in order): cat, car, person, person, train, horse, train,car,person.

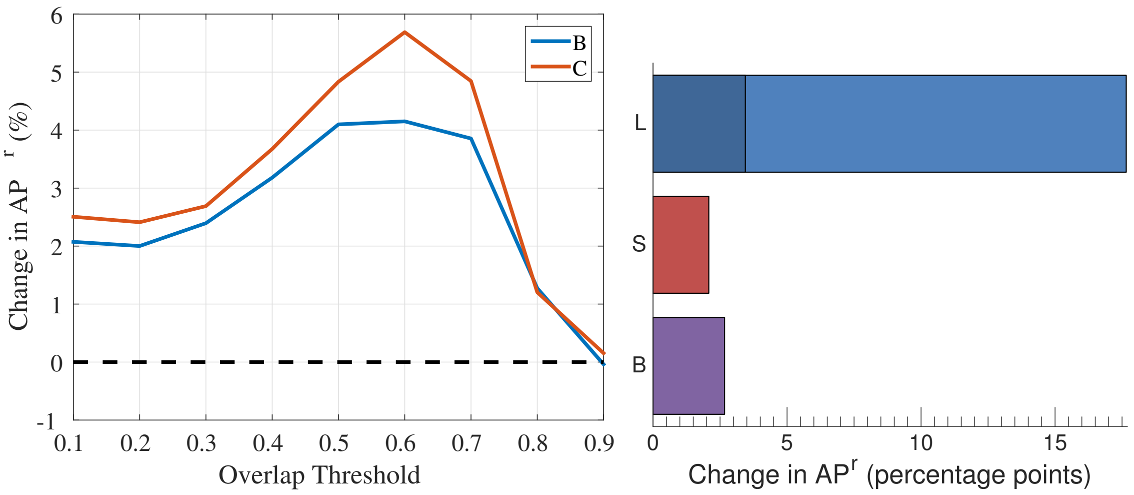

Figure 2.3: Left: The improvement offered by **B** and **C** over a range of overlap thresholds. Right: The impact on AP^r of mislocalization, confusion with similar categories and confusion with background.

	AP^r at 0.5				AP^r at 0.7			
	O2P	**A**	**B**	**C**	O2P	**A**	**B**	**C**
aero	56.5	61.8	65.7	**67.4**	**34.0**	19.3	24.1	31.0
bike	19.0	43.4	**49.6**	49.6	7.0	11.1	13.1	**14.0**
bird	23.0	46.6	47.2	**49.1**	10.8	22.6	26.0	**27.3**
boat	12.2	27.2	**30.0**	29.9	3.8	7.4	8.1	**8.1**
bottle	11.0	28.9	31.7	**32.0**	5.4	**16.9**	16.0	16.7
bus	48.8	61.7	**66.9**	65.9	30.4	31.5	**42.4**	41.1
car	26.0	46.9	50.9	**51.4**	12.6	21.4	26.7	**27.3**
cat	43.3	58.4	69.2	**70.6**	23.1	25.9	36.9	**40.1**
chair	4.7	17.8	19.6	**20.2**	0.6	5.5	**6.0**	5.5
cow	15.6	38.8	42.7	**42.7**	8.0	20.9	**25.0**	24.2
table	7.8	18.6	22.8	**22.9**	1.3	7.1	6.7	**7.8**
dog	24.2	52.6	56.2	**58.7**	9.8	23.8	**28.5**	27.8
horse	27.5	44.3	51.9	**54.4**	9.8	9.3	14.8	**17.4**
mbike	32.3	50.2	52.6	**53.5**	9.4	9.6	13.0	**13.7**
person	23.5	48.2	52.6	**54.4**	6.4	16.1	20.1	**21.7**
plant	4.6	23.8	**25.7**	24.9	0.8	8.2	8.6	**8.8**
sheep	32.3	**54.2**	54.2	54.1	18.0	25.6	**30.8**	30.8
sofa	20.7	26.0	**32.2**	31.4	7.2	14.2	16.3	**16.8**
train	38.8	53.2	59.2	**62.2**	21.4	23.4	30.9	**34.0**
tv	32.3	55.3	58.7	**59.3**	11.9	40.2	**43.0**	42.6
mean	25.2	42.9	47.0	**47.7**	11.6	18.0	21.9	**22.8**

Table 2.1: Category-wise AP^r

Chapter 3

Predicting Figure-ground

As discussed in the previous chapter, one of the major error modes of the system we have described is mislocalization. For localization, we rely principally on the bottom-up proposal method to give us a good candidate. However, proposal methods are typically based on low-level color and texture cues. For images such as the dog in Figure 3.1, bottom-up signals can be deceptive: low contrast object boundaries may be missed (such as that between the torso of the dog and the wall) and high contrast internal contours (such as that between the snout and the torso) might be mistaken as object boundaries.

However, once a recognition system has figured out that this is a dog, top-down understanding can be brought to bear: the algorithm can reason that this is a dog, dogs have torsos, and so the pixels below the dog's snout are probably part of the torso, even though they share the wall's color. In this chapter we build exactly such a top-down figure-ground prediction system. Its output for this object is shown in the third image in Figure 3.1

As for region classification, we want to use features from convolutional networks or CNNs for making this prediction. In the region classification step, we used the output of the last layer of the CNN. This makes sense when the task is assigning category labels to images or bounding boxes: the last layer is the most sensitive to category-level semantic information and the most invariant to "nuisance" variables such as pose, illumination, articulation, precise location and so on. However, when the task we are interested in is finer-grained, such as one of segmenting the detected object or estimating its pose, these nuisance variables are precisely what we are interested in. For such applications, the top layer is thus *not* the optimal representation.

The information that is generalized over in the top layer is present in intermediate layers, but intermediate layers are also much less sensitive to semantics. For instance, bar detectors in early layers might localize bars precisely, but cannot discriminate between bars that are horse legs and bars that are tree trunks. This observation suggests that reasoning at multiple levels of abstraction and scale is necessary, mirroring other problems in computer vision where reasoning across multiple levels has proven beneficial. For example, in optical flow, coarse levels of the image pyramid are good for correspondence, but finer levels are needed for accurate measurement, and a multiscale strategy is used to get the best of both worlds [8].

We think of the layers of a convolutional network as a non-linear counterpart of the image

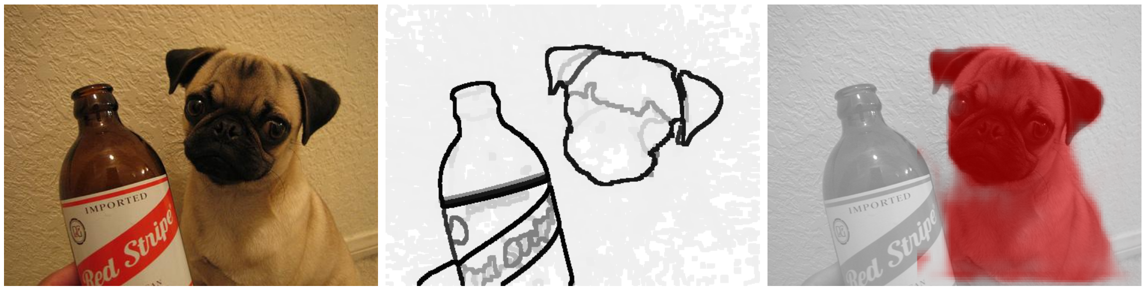

Figure 3.1: The image of a dog (left), its bottom-up contours (middle) and the segmentation predicted using top-down knowledge (right). The bottom-up contours miss the boundary between the dog body and the wall, and fire erroneously on the contour between the dog face and its torso. On the other hand, once we detect the dog, our top-down figure-ground prediction system does a much better job of segmenting out the dog.

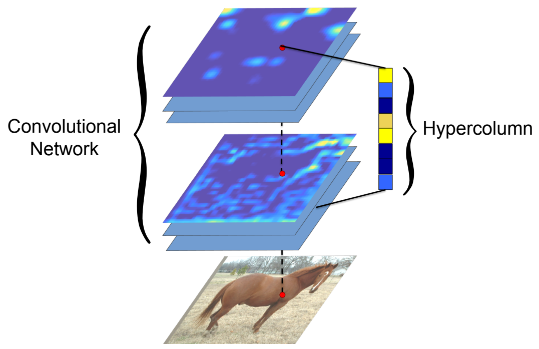

Figure 3.2: The hypercolumn representation. The bottom image is the input, and above it are the feature maps of different layers in the CNN. The hypercolumn at a pixel is the vector of activations of all units that lie above that pixel.

pyramids used in optical flow and other vision tasks. Our hypothesis is that the information of interest is distributed over *all* levels of the CNN and should be exploited in this way. We define the "hypercolumn" at a given input location as the outputs of all units above that location at all layers of the CNN, stacked into one vector. (Because adjacent layers are strongly correlated, in practice we need not consider all layers but can simply sample a few.) The last image in Figure 3.2 shows a visualization of the idea. We borrow the term "hypercolumn" from neuroscience, where it is used to describe a set of V1 neurons sensitive to edges at multiple orientations and multiple frequencies arranged in a columnar structure [34]. However, our hypercolumn includes not just edge detectors but also more semantic units and is thus a more general notion.

3.1 Connections to prior work

The idea of using top-down knowledge to localize objects better is an old one. Borenstein and Ullman [6] first suggested the idea of using class-specific knowledge for segmentation. Yang et al. [61] use figure ground masks associated with DPM detectors [21] to segment out detected objects and reason about depth orderings. Parkhi et al. [49] use color models extracted from the detected cat and dog heads to segment them out. Dai and Hoiem [14] generalize this reasoning to all categories.

Combining features across multiple scales is also a classic idea in computer vision. Burt and Adelson introduced Laplacian pyramids [9], a representation that is widely used in computer vision. Koenderink and van Doorn [39] used “jets”, which are sets of partial derivatives of intensity up to a particular order, to estimate edge orientation, curvature, etc. Malik and Perona [46] used the output of a bank of filters as a representation for texture discrimination. This representation also proved useful for optical flow [58] and stereo [37]. While the filter banks in these works cover multiple scales, they are still restricted to simple linear filters, whereas many of the features in the hypercolumn representation are highly non-linear functions of the image.

There has also been work in convolutional networks that combines multiple levels of abstraction and scale. Farabet et al. [20] combine CNN outputs from multiple scales of an image to do semantic segmentation. Tompson et al. [56] use a similar idea for detecting parts and estimating pose. However, the features being combined still come from the same level of the CNN and hence have similar invariance. Sermanet et al. [53] combine subsampled intermediate layers with the top layer for pedestrian detection. In contrast, since we aim for precise localization, we maintain the high resolution of the lower layers and upsample the higher layers instead. In contemporary work, Long et al. [44] also use multiple layers for their fully convolutional semantic segmentation system.

3.2 Using hypercolumns for figure-ground prediction

Figure-ground prediction as pixel classification: Let us assume that the object has been detected and coarsely localized (as in the output of an object detection system). This means that we have a category label and a bounding box. We may also have an initial segmentation hypothesis (if we start from the detections output by the region classifier described in the previous chapter). We now want to segment this object out.

We expand the bounding box slightly and predict a heatmap on this expanded box. This heatmap encodes the probability that a particular location is inside the object. We predict a 50×50 heatmap that we resize to the size of the expanded bounding box and splat onto the image. Figure ground prediction thus reduces to assigning a probability to each of the 50×50 locations or, in other words, of classifying each location. We solve this classification problem using the hypercolumn representation as described in detail below.

Computing the hypercolumn representation: We take the cropped bounding box, resize it to a fixed size and feed it into a CNN as in [24]. For each location, we extract features from a set of layers by taking the outputs of the units that are "above" the location (as shown in Figure 3.2). All the intermediate outputs in a CNN are feature maps (the output of a fully connected layer can be seen as a 1×1 feature map). However, because of subsampling and pooling operations in the CNN, these feature maps need not be at the same resolution as the input or the target output size. So which unit lies above a particular location is ambiguous. We get around this by simply resizing each feature map to the size we want with bilinear interpolation. If we denote the feature map by $\mathbf{F}$ and the upsampled feature map by $\mathbf{f}$, then the feature vector for the ith location has the form:

$$\mathbf{f}_i = \sum_k \alpha_{ik} \mathbf{F}_k \tag{3.1}$$

α_{ik} depends on the position of i and k in the box and feature map respectively.

We concatenate features from some or all of the feature maps in the network into one long vector for every location which we call the hypercolumn at that location. As an example, using *pool2* (256 channels), *conv4* (384 channels) and *fc7* (4096 channels) from the architecture of [41] would lead to a 4736 dimensional vector.

Using an initial segmentation: The features we use for the classification need not be restricted to CNN outputs. For instance, an initial segmentation might be available (as when we are using high ranking regions from our region classifier). This initial segmentation captures bottom-up color and texture coherence and is therefore valuable signal. We simply append this initial mask as additional features (which are 1 for locations inside this mask and 0 otherwise). To capture the global shape indicated by this initial mask, we discretize the mask using a 10×10 grid, and use these 100 features as 1×1 feature maps (as with the output of fully connected layers). In other words, these 100 features are shared by all locations.

Interpolating into a grid of classifiers: Because these feature maps are the result of convolutions and poolings, they do not encode any information about where in the bounding box a given pixel lies. However, location can be an important feature. For instance, in a person bounding box, the head is more likely to be at the top of the bounding box than at the bottom. Thus a pixel that looks like a nose should be considered as part of the person if it occurs at the top of the box and should be classified as background otherwise. The reasoning should be the opposite for a foot-like pixel. This is a highly non-linear effect of location, and such reasoning cannot be achieved simply by a location-specific bias. (Indeed, our classifiers include (x, y) as features but assign negligible weight to them). Such reasoning requires different classifiers for each location.

Location is also needed to make better use of the features from the fully connected layers at the top. Since these features are shared by all the locations in the bounding box, they can at best contribute a global instance-specific bias. However, with a different classifier at each location, we can have a separate instance-specific bias for each location. Thus location-specific classifiers in conjunction with the global, instance-level features from the fully connected layer produce an instance-specific prior.

The simplest way to get a location-specific classifier is to train separate classifiers for each of the 50×50 locations. However, doing so has three problems. One, it dramatically reduces the amount of data each classifier sees during training. In our training sets, some categories may have only a few hundred instances, while the dimensionality of the feature vector is of the order of several thousand. Thus, having fewer parameters and more sharing of data is necessary to prevent overfitting. Two, training this many classifiers is computationally expensive, since we will have to train 2500 classifiers for 20 categories. Three, while we do want the classifier to vary with location, the classifier should change slowly: two adjacent pixels that are similar to each other in appearance should also be classified similarly.

Our solution is to train a coarse $K \times K$ grid of classifiers and interpolate between them. In our experiments we use $K = 5$ or 10. For the interpolation, we use an extension of bilinear interpolation where we interpolate a grid of *functions* instead of a grid of *values*. Concretely, each classifier in the grid is a function $g_k(\cdot)$ that takes in a feature vector and outputs a probability between 0 and 1. We use this coarse grid of functions to define the function h_i at each pixel i as a linear combination of the nearby grid functions, analogous to Equation 3.1:

$$h_i(\cdot) = \sum_k \alpha_{ik} g_k(\cdot) \tag{3.2}$$

If the feature vector at the ith pixel is $\mathbf{f}_i$, then the score of the ith pixel is:

$$p_i = \sum_k \alpha_{ik} g_k(\mathbf{f}_i) = \sum_k \alpha_{ik} p_{ik} \tag{3.3}$$

where p_{ik} is the probability output by the kth classifier for the ith pixel. Thus, at test time we run all our K^2 classifiers on all the pixels. Then, at each pixel, we linearly combine the outputs of all classifiers at that pixel using the above equation to produce the final prediction. Note that the coefficients of the linear combination depend on the location.

Training this interpolated classifier is a hard optimization problem. We use a simple heuristic and ignore the interpolation at train time, using it only at test time.We divide each training bounding box into a $K \times K$ grid. The training data for the kth classifier consists only of pixels from the kth grid cell across all training instances. Each classifier is trained using logistic regression. This training methodology does not directly optimize the loss we would encounter at test time, but allows us to use off-the-shelf code such as liblinear [19] to train the logistic regressor.

Efficient classification using convolutions and upsampling: Our system requires us to resize every feature map to 50×50 and then classify each location. But resizing feature maps with hundreds of channels can be expensive. However, we know we are going to run several linear classifiers on top of the hypercolumn features and we can use this knowledge to save computation as follows: each feature map with c channels will give rise to a c-dimensional block of features in the hypercolumn representation of a location, and this block will have a corresponding block of weights in the classifiers. Thus if $\mathbf{f}_i$ is the feature vector at location i, then $\mathbf{f}_i$ will be composed of blocks $\mathbf{f}_i^{(j)}$ corresponding to the jth feature map. A linear classifier $\mathbf{w}$ will decompose similarly.

The dot product between $\mathbf{w}$ and $\mathbf{f}_i$ can then be written as:

$$\mathbf{w}^T\mathbf{f}_i = \sum_j \mathbf{w}^{(j)T}\mathbf{f}_i^{(j)} \tag{3.4}$$

The jth term in the decomposition corresponds to a linear classifier on top of the upsampled jth feature map. However, since the upsampling is a linear operation, we can first apply the classifier and then upsample using Equation 3.1:

$$\mathbf{f}_i^{(j)} = \sum_k \alpha_{ik}^{(j)}\mathbf{F}_k^{(j)} \tag{3.5}$$

$$\mathbf{w}^{(j)T}\mathbf{f}_i^{(j)} = \sum_k \alpha_{ik}^{(j)}\mathbf{w}^{(j)T}\mathbf{F}_k^{(j)} \tag{3.6}$$

We note that this insight was also used by Barron et al. [3] in their volumetric semantic segmentation system.

Observe that applying a classifier to each location in a feature map is the same as a 1×1 convolution. Thus, to run a linear classifier on top of hypercolumn features, we break it into blocks corresponding to each feature map, run 1×1 convolutions on each feature map to produce score maps, upsample all score maps to the target resolution, and sum.

We consider a further modification to this pipeline where we replace the 1×1 convolution with a general $n \times n$ convolution. This corresponds to looking not only at the unit directly above a pixel but also the neighborhood of the unit. This captures the pattern of activations of a whole neighborhood, which can be more informative than a single unit, especially in the lower layers of the network.

Representation as a neural network: We can write our final hypercolumn classifier using additional layers grafted onto the original CNN as shown in Figure 3.3. For each feature map, we stack on an additional convolutional layer. Each such convolutional layer has K^2 channels, corresponding to the K^2 classifiers we want to train. We can choose any kernel size for the convolutions as described above, although for fully connected layers that produce 1×1 feature maps, we are restricted to 1×1 convolutions. We take the outputs of all these layers, upsample them using bilinear interpolation and sum them. Finally, we pass these outputs through a sigmoid, and combine the K^2 heatmaps using equation 3.3 to give our final output. Each operation is differentiable and can be back-propagated over.

Representing our pipeline as a neural network allows us to train the whole network (including the CNN from which we extract features) for this task. For such training, we feed in the target 50×50 heatmap as a label. The loss is the sum of logistic losses (or equivalently, the sum of the negative log likelihoods) over all the 50×50 locations. We found that treating the sigmoids, the linear combination and the log likelihood as a single composite function and computing the gradient with respect to that led to simpler, more numerically stable expressions. Instead of training the network from scratch, we use a pretrained network and finetune, i.e., do backpropagation with a small learning rate. Finally, this representation as a neural network also allows us to train the

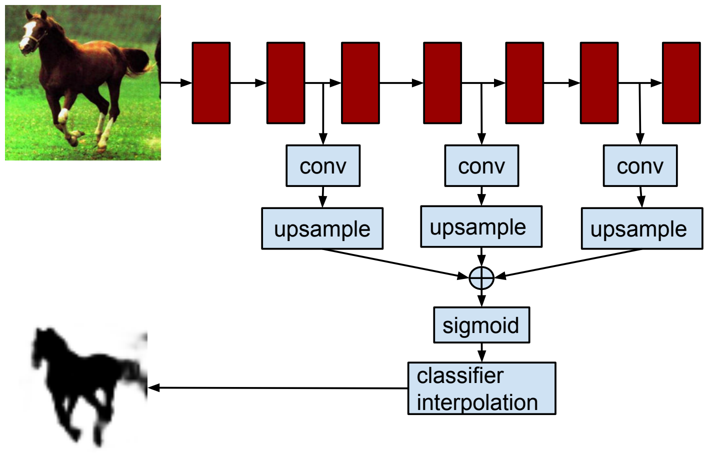

Figure 3.3: Representing our hypercolumn classifiers as a neural network. Layers of the original classification CNN are shown in red, and layers that we add are in blue.

grid classifiers together and use classifier interpolation during training, instead of training separate grid classifiers independent of each other.

Training setup: For each category we take bottom-up MCG candidates [1] that overlap a ground truth instance by 70% or more. For each such candidate, we find the ground truth instance it overlaps most with, and crop that ground truth instance to the expanded bounding box of the candidate. We then use the labeling of the cropped ground truth instance to label locations inside the instance as positive and locations outside as negative. We use images from VOC2012 Train as the training set.

Superpixel projection: At test time, after we predict the figure-ground mask, we project the mask to superpixels by assigning to each superpixel the average value of the mask in that superpixel. This serves to smooth away some of the noise in the masks and also makes sure that the segmentation adheres to image boundaries. As a superpixel representation, we simply take the union of the boundaries of all the MCG proposals.

3.3 Refining SDS detections

We start from SDS detections from our region classifier. These detections come with an initial segmentation, which is the segmentation output by the bottom-up proposer (MCG). As described above, we use the initial segmentation to provide additional features. For the CNN, we use **C**. For the hypercolumn representation we use the top-level *fc7* features, the *conv4* features from both pathways using a 1×1 neighborhood, and the *pool2* features from the box pathway with a 3×3 neighborhood. We choose these layers because they are spread out evenly in the network and capture a diverse set of features. We use a 10×10 grid of classifiers. As a last step, we project our predictions to superpixels by averaging the prediction over each superpixel. We train on VOC2012 Train and evaluate on VOC2012 Val.

Table 3.1 shows the results of our experiments. The first column shows the performance of the basic region classification system. "Hyp" is the result we get by making a figure-ground prediction using hypercolumns. We improve mean AP^r at 0.5 by **3.5** points, and at 0.7 by **8.8** points, indicating a large improvement over the original candidate. We can improve the prediction by first doing bounding box regression [24] and using the regressed boxes as the domain for our pixel classification. As expected, better bounding boxes lead to better segmentations: bounding box regression adds a small but statistically significant 0.7 points at 0.5 overlap and 0.8 points at 0.7 overlap. Finetuning the entire network for this prediction improves this even further, pushing AP^r at 0.7 by another 1.3 points[1].

Table 3.1 also shows the results of several ablations of our model (all without bounding box regression or finetuning):

1. *Only fc7* uses only *fc7* features, i.e features from the topmost layer of the CNN. The poor performance of this baseline, especially at 0.7 overlap, confirms our intuitions that the top layer of the network has lost localization information and one needs to look at lower layers.

2. *fc7+pool2, fc7+conv4* and *pool2+conv4* are refinement systems that use hypercolumns but leave out features from *conv4*, *pool2* and *fc7* respectively. Each of these baselines performs worse than our full system. In each case the difference is statistically significant at a confidence threshold of 0.05, computed using paired sample permutation tests.

3. The 1×1, 2×2 and 5×5 models use different grid resolutions, with the 1×1 grid amounting to a single classifier. There is a significant loss in performance (2.4 points at 0.7 overlap) when using a 1×1 grid. However this baseline still outperforms a 10×10 grid of classifiers operating on the topmost layer of the network, indicating that even without our grid classifiers (and without *fc7*, since the global *fc7* features are ineffectual without the grid), the hypercolumn representation by itself is quite powerful. A 5×5 grid is enough to recover full performance.

Figure 3.4 shows some example figure-ground predictions made using hypercolumns and using the baseline system that uses the topmost layer. Hypercolumn-based predictions are sharper and capture fine details such as thin structures, even for novel poses.

3.4 Segmenting out bounding box detections

We now investigate if we can segment bounding box detections from an off-the-shelf object detector. This setting is important because it makes our approach much more generically applicable to

[1]For computational reasons, we used a 5×5 grid for the finetuned result

Metric	Region Classification	Hyp	Hyp +bbox-reg	Hyp+FT +bbox-reg	Only *fc7*	*fc7*+ *pool2*	*fc7*+ *conv4*	*pool2*+ *conv4*	1×1 grid	2×2 grid	5×5 grid
mean AP^r at 0.5	47.7	51.2	51.9	**52.8**	49.7	50.5	51.0	50.7	50.3	51.2	51.3
mean AP^r at 0.7	22.8	31.6	32.4	**33.7**	25.8	30.6	31.2	30.8	28.8	30.2	31.8

Table 3.1: Results on SDS on VOC2012 val. Our final figure-ground prediction (Hyp+FT+bbox-reg) is significantly better than the output of our basic region classifier.

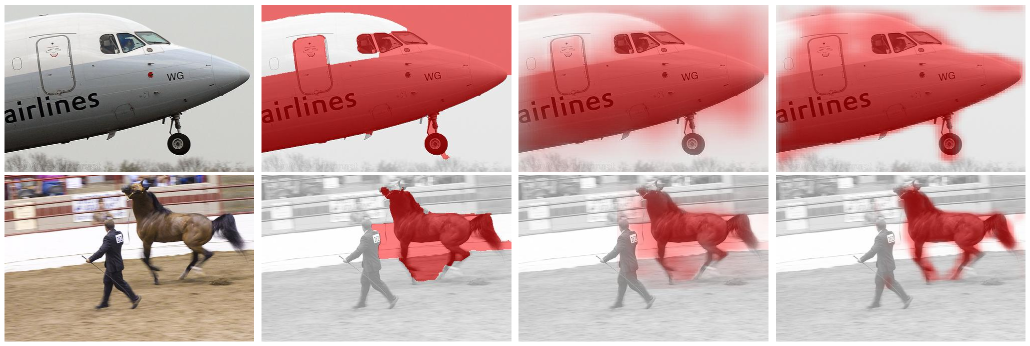
Figure 3.4: Comparing the hypercolumn representation to the topmost layer of the CNN. Each row shows the image, the original proposal ranked high by the classifier, the predicted figure-ground predicted using the topmost layer (*fc7*) and the predicted figure-ground predicted using the hypercolumn representation. The hypercolumn-based prediction is sharper and captures thin structures such as legs, even in fairly novel poses. The results shown are before superpixel projection.

any object detection approach, and allows us to easily take advantage of ongoing rapid progress in object detection.

Another rationale for starting from bounding box object detections is efficiency: scoring region proposals by masking out cropped image patches is quite slow (however see [13] for contemporary work on speeding it up). On the contrary, several recent papers have explored methods of speeding up bounding box object detection [32, 25]. Segmenting the small number of detections output by a fast object detection system is thus an efficient solution for SDS.

We use the detections of R-CNN [24] as the starting point. We use a version of R-CNN retrained with MCG boxes. We use the final output after non-max suppression and bounding box regression. We do all training on VOC2012 Train.

We first evaluate our segmentation predictions. As before, we use the same network as the detector to compute the hypercolumn transform features. We first experiment with the Alexnet architecture [41]: this is also the architecture of each stream of our region classification network described earlier. We use the layers *fc7*, *conv4* with a neighborhood of 1, and *pool2* with a neighborhood of 3. For computational reasons we do not do any finetuning. We use superpixel projection as before.

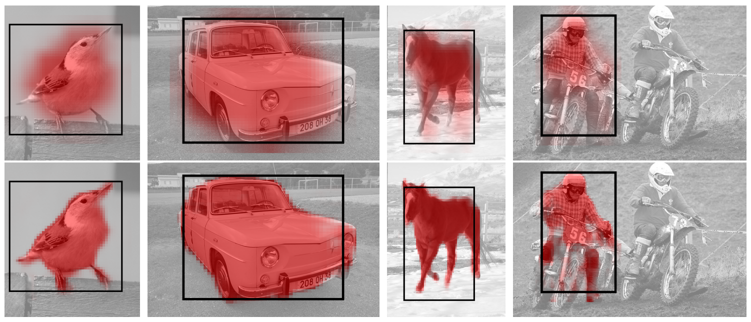
Figure 3.5: Figure ground segmentations starting from bounding box detections. Top row: baseline using *fc7*, bottom row: Ours. The results shown are before superpixel projection.

We show results in Table 3.2. Since we use only one network operating on bounding boxes instead of two working on both the box and the region, we expect a drop in performance. We find that this is the case, but the loss is small: we get a mean AP^r of 49.1 at 0.5 and 29.1 at 0.7, compared to 51.9 and 32.4 when we have the region features. In fact, compared to the simple region classification baseline, our performance is about 1.5 points better at 0.5 and about 6.3 points better at 0.7, and we get this accuracy starting from just the bounding box.

To see how much of this performance is coming from the hypercolumn representation, we also run a baseline using just *fc7* features. As expected, this baseline is only able to output a fuzzy segmentation, compared to the sharp delineation we get using hypercolumns. It performs considerably worse, losing 5 points at 0.5 overlap and almost 13 points at 0.7 overlap. Figure 3.5 shows example segmentations.

Girshick et al. [24] also experimented with deeper architectures for R-CNN. In particular, they explored the VGG architecture proposed by Simonyan et al. [54], and found that it gave large gains in detection performance (about 8 point gain in AP). We therefore now switch to the VGG architecture. We again retrain the R-CNN system using this architecture on MCG bounding box proposals. Again, for the hypercolumn representation we use the same network as the detector. We use the layers *fc7*, *conv4* with a neighborhood of 1 and *pool3* with a neighborhood of 3. (We use *pool3* instead of *pool2* because the *pool3* feature map has about half the resolution and is thus easier to work with.)

We observe that the VGG architecture is significantly better than AlexNet: we get a boost of 7.5 points at the 0.5 overlap threshold and 8 points at the 0.7 threshold. We also find that this architecture gives us the best performance on the SDS task so far: with simple bounding box detection followed by our hypercolumn-based mask prediction, we achieve a mean AP^r of 56.5 at an overlap threshold of 0.5 and a mean AP^r of 37.0 at an overlap threshold of 0.7. These numbers are a huge improvement over the region classification baseline, and are significantly better than the best system we have obtained till now. Last but not the least, we observe that the large gap between our hypercolumn system and the only-*fc7* baseline persists, and is equally large for the VGG architecture. This implies that the gain provided by hypercolumns is not specific to a particular network architecture. Figure 3.5 visualizes our VGG results.

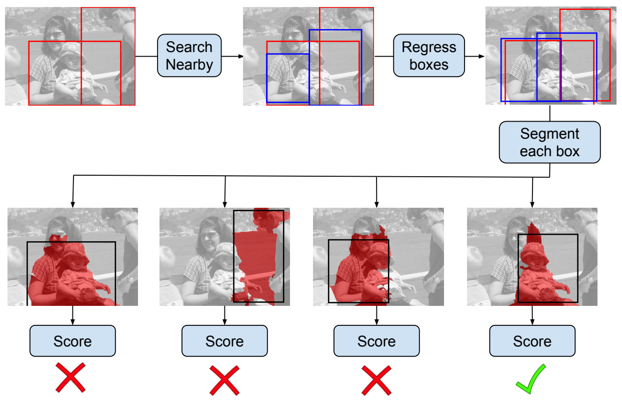

Figure 3.6: An alternative pipeline for SDS starting from bounding box detections

While these results are impressive, note that since we are now directly segmenting bounding box detections, we lose two important cues. One, since the region proposal no longer comes into play, the bottom-up color and texture coherence cues it encoded are no longer available. Two, the segmentation is never used in the detection scoring step.

We can correct the second problem by using the predicted segmentation for rescoring the detections. To do this, we start with bounding box detections after non-maximum suppression. To avoid discarding potentially better localized boxes, we expand this set of detections by adding to the pool boxes with score higher than a threshold that were suppressed by NMS but that overlap the detections by less than 0.7. This expanded set is only twice as large as the original set, and about two orders of magnitude smaller than the full set of bottom-up proposals. For each candidate in this set, we predict a segmentation, and score this candidate using CNN features computed on the segmentation. Because region-based features are computed only on a small set, the pipeline is much more efficient than classifying every region proposal. This pipeline is summarized in Figure 3.6.

For the segment-based rescoring, we use the VGG architecture. Instead of a bounding box, we take as input the bounding box with the region background masked out. This network is trained in the same way as described in the previous chapter. We use features from the topmost layer of this network and concatenate them with the features from the top layer of the detection network, and feed these into an SVM. For training data, we use our expanded pool of candidates on the training set, and take all candidates for which segmentation predictions overlap groundtruth by more than 70% as positive and those with overlap less than 50% as negative. After rescoring, we do a non-max suppression using region overlap to get the final set of detections (we use an overlap threshold of 0.3).

We get **60.0** mean AP^r at 0.5, and **40.4** mean AP^r at 0.7. These numbers are state-of-the-art on the SDS benchmark (in contemporary work, [13] get slightly higher performance at 0.5 but do not report the performance at 0.7; our gains are orthogonal to theirs).

Metric	AlexNet Only *fc7*	AlexNet Hyp	VGG Only *fc7*	VGG Hyp	VGG Hyp+ Rescore
mAP^r at 0.5	44.0	49.1	52.6	56.5	**60.0**
mAP^r at 0.7	16.3	29.1	22.4	37.0	**40.4**

Table 3.2: Results on SDS on VOC 2012 val using the detect-segment-rescore pipeline.Our final pipeline is state-of-the-art on SDS.

3.5 Efficient SDS

A lot of recent research on object detection has aimed at speeding up R-CNN [32, 25]. The key idea behind these approaches is that instead of warping and computing features on each box proposal separately, some computation can be shared. In particular, the convolutional layers are agnostic to the spatial extent of their input, and so they can be run once on the full image. Then for any box proposal, one can simply crop out the corresponding box out of the resulting feature map to get a feature map for that proposal. This feature map can then be fed into subsequent fully connected layers. However, fully connected layers accept a fixed sized input, whereas cropped feature maps will have varied sizes depending on the size of the box. He et al. [32] got around this by using a spatial pyramid to pool features before passing them into fully connected layers. A spatial pyramid grid divides the feature map into a fixed number of bins. Features are then pooled over each bin, and the pooled features from all bins are concatenated into one vector and passed forward. He et al.called this a spatial pyramid pooling layer or SPP. Girshick [25] extended this idea into an end-to-end trainable system for object detection, called Fast R-CNN.

It is straightforward to extend the hypercolumn idea to such architectures. Intuitively, to get the hypercolumn representation for a given pixel in a particular bounding box, one possibility is to crop out the box from each feature map, and then interpolate into these cropped feature maps. As before, cropping is also a linear operation. So instead of first cropping and upsampling and then convolving with a filter (corresponding to a block of weights in a classifier), we can convolve the entire image-level feature map with the filter, and then crop and upsample the filter response. (Note that this modification is not needed for the fully connected layers, since the fully connected layers are still evaluated separately for each box as before.) Figure 3.7 shows how this classification is implemented in the network.

The shared computation drastically reduces the time taken, as observed by [32, 25]. This is true even when working with a few hundred detections per image: the fast hypercolumn prediction is able to segment upto 400 boxes in about 300 milliseconds using the VGG system, while passing an equal number of cropped and warped image patches through VGG takes several seconds.

Table 3.3 shows experimental results using the fast figure-ground prediction. We evaluate three alternatives:

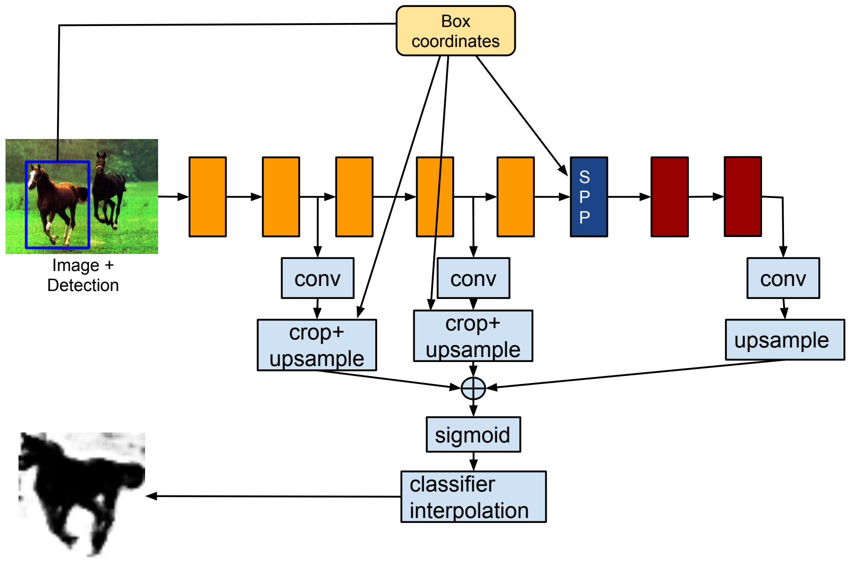

Figure 3.7: Fast hypercolumn prediction using the SPP idea. Convolutional layers of the CNN (shown in orange) are run just once on the entire image. Then for each box, the SPP layer uses a spatial pyramid grid to compute a fixed length vector that is then passed to the fully connected layers (in red). Similarly for the hypercolumn based figure-ground prediction, the image-level convolutional feature maps are convolved with filters once, and for each box the filter responses are cropped, upsampled and added before classifier interpolation. The fully connected layer features are still separately computed per box.

1. To make an apples-to-apples comparison between fast and slow figure-ground prediction, we run our fast figure-ground predictor on (slow) R-CNN detections and compare it with numbers reported in the previous section. Note that the fast figure-ground predictor uses a VGG network trained as in Fast R-CNN [25] as the substrate on which to compute features.

2. Next we take detections from the Fast R-CNN system [25] which uses the SPP trick to speed up object detection, and then segment each detection out using our fast figure-ground predictor. Fast R-CNN takes about 500 milliseconds per image to detect all the objects, and the tens or hundreds of detections that are produced are segmented by our system in a few hundred milliseconds. The total time taken per image is thus less than a second.

3. Finally we implement the detect-segment-rescore pipeline described in the previous section, with Fast R-CNN for the detection step and fast figure-ground prediction for the segmentation step. The rescore step remains the same as before.[2]

[2]One can also speed up the rescoring step using similar ideas; see for example [13]. However, we do not explore this in this thesis.

	Slow Seg		Fast Seg		
Metric	Slow Det	Slow Det +Rescore	Slow Det	Fast Det	Fast Det +Rescore
mAP^r at 0.5	56.5	60.0	56.6	58.6	**62.4**
mAP^r at 0.7	36.5	**40.4**	35.3	36.4	39.4

Table 3.3: Speeding up SDS. The first two columns show results from the slow system with and without rescoring. The other columns show the results of our fast figure-ground prediction system operating on slow R-CNN or Fast R-CNN, and the result with Fast R-CNN after rescoring.

We observe that in the apples-to-apples comparison when operating on the same set of detections, the slow figure-ground predictor is about 1.2 points better at 0.7 overlap, and about the same at 0.5 overlap. However this difference in performance disappears when we shift to Fast R-CNN detections. Indeed, our fast system when applied to Fast R-CNN detections is better by about 2 percentage points at 0.5 overlap. Girshick [25] showed that Fast R-CNN is in fact better than vanilla R-CNN owing to the joint training of the bounding box regressor and the detector. This improvement in detection thus carries over to SDS. Finally, rescoring the segmented objects as before pushes performance up to 62.4 for 0.5 overlap, 2 points higher than the slow system. At 0.7 overlap, it is about 1 point below the slow system. This might be an acceptable loss in performance considering the large gain in speed offered.

3.6 Semantic Segmentation

All of our SDS results can be converted into a class label for each pixel, i.e. a semantic segmentation output. We do this using the pasting scheme of Carreira et al. [10]: detections that score over a threshold are pasted into the image in increasing order of scores. Our slow detect-segment-rescore system achieves a mean IU of 62.6, and our fast version achieves 62.0. This is comparable to contemporary semantic segmentation approaches such as [44]. However note that our output is much richer than standard semantic segmentation approaches since it produces separate instances.

Chapter 4

Extension to other tasks

As described in Chapter 1, downstream applications may want to know more about the objects in a scene than their segmentations, such as the segmentation of the object into parts or the location of various keypoints. Even if we are interested in segmentation, one may also want to place emphasis on getting the boundaries right: for instance, graphics applications are especially sensitive to errors on the boundaries, which can be visually jarring. In this chapter, we show how the techniques we have developed can be used to tackle these other tasks and present experimental results.

The key idea is that all these tasks can be cast in the framework of classifying pixels inside the bounding box of the object. The target for this classification varies depending on the task. For the task of part labeling, we ask whether a particular pixel belongs to a part. For keypoint localization, we ask whether the pixel lies on the keypoint. For boundary prediction, we ask whether a pixel lies on the boundary of the object. We also predict the polarity of the boundary: which side of the boundary the object lies on.

4.1 Part Labeling and Keypoint Localization

Training setup For each category we take bottom-up MCG candidates [1] that overlap a ground truth instance by 70% or more. For each such candidate, we find the ground truth instance it overlaps most with, and crop that ground truth instance to the expanded bounding box of the candidate. Depending on the task we are interested in (keypoint prediction or boundary prediction), we then use the labeling of the cropped ground truth instance to label locations in the expanded bounding box as positive or negative. For part labeling, locations inside a part are positive and all other locations are negative. For keypoint prediction, the true keypoint location is positive and locations outside a certain radius (we use 10% of the bounding box diagonal) of the true location are labeled negative. We use the same features and network as in Section 3.3. As before, we do all training on VOC2012 Train.

We evaluate part localization in the unconstrained detection setting, where the task is to both detect the object and label its keypoints/segment its parts. This is different from most prior work on these problems [57, 56, 60, 59, 4, 45], which operates on the immediate vicinity of ground-truth

Method	L.S	L.E	L.W	R.S	R.E	R.W	L.H	L.K	L.A	R.H	R.K	R.A	N	**Mean**
[28]	27.3	11.8	2.8	27.1	12.2	3.4	**11.4**	4.9	3.2	**10.6**	4.4	3.8	42.9	*12.8*
[27]	32.1	14.6	5.6	32.5	**16.6**	5.9	10.8	4.8	4.8	9.7	4.0	4.6	52.0	*15.2*
Only *fc7*	22.7	9.9	2.8	25.5	10.0	2.6	6.6	3.5	5.2	7.7	3.4	4.2	34.0	*10.6*
Hyp	32.2	16.5	11.5	31.2	16.6	9.3	9.6	7.1	**9.1**	8.0	4.2	**8.2**	57.5	*17.0*
Hyp+FT	**33.7**	**21.9**	**12.3**	**35.2**	**20.9**	**15.3**	6.7	**7.7**	8.1	**9.1**	**5.6**	6.1	**58.4**	***18.5***

Table 4.1: Results on keypoint prediction (APK on the Person subset of VOC2009 val). Our system is 3.3 points better than [27] (Section 4.1).

instances.

Keypoint prediction We evaluate keypoint prediction on the "person" category using the protocol described in [28]. The test set for evaluating keypoints is the person images in the second half of VOC2009 val. As described in Section 1.2, we use the APK metric [62], which evaluates keypoint predictions in a detection setting. Each detection comes with a keypoint prediction and a score. A predicted keypoint within a threshold distance (0.2 of the torso height) of the ground-truth keypoint is a true positive, and is a false positive otherwise. The area under the PR curve gives the APK for that keypoint.

We start from the person detections of our SDS region classification system (Chapter 2.1). We use bounding box regression to start from a better bounding box. As described in Section 3.2 we train a separate system for each keypoint using the hypercolumn representation. We use keypoint annotations collected by [7]. We produce a heatmap for each keypoint and then take the highest scoring location of the heatmap as the keypoint prediction.

The APK metric requires us to attach a score with each keypoint prediction. This score must combine the confidence in the person detection and the confidence in the keypoint prediction, since predicting a keypoint when the keypoint is invisible counts as a false positive. For this score we multiply the value of the keypoint heatmap at the predicted location with the score output by the person detector (which we pass through a sigmoid).

Results are shown in Table 4.1. We compare our performance to [27], the previous best on this dataset. Gkioxari et al. [27] finetuned a network for pose, person detection and action classification, and then trained an SVM to assign a score to the keypoint predictions. Without any finetuning for pose, our system achieves a 1.8 point boost. A baseline system trained using our pipeline but with just the *fc7* features performs significantly worse than our system, and is even worse than a HOG-based method [28]. This confirms that the gains we get are from the hypercolumn representation. Figure 4.1 shows some example predictions.

Finetuning the network as described in Section 3.2 gives an additional **1.5** point gain, raising mean APK to **18.5**.

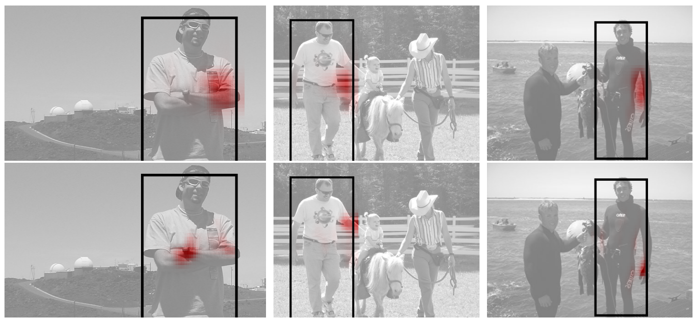

Figure 4.1: Keypoint prediction (left wrist). Top row: baseline using *fc7*, bottom row: ours (hypercolumns without finetuning). In black is the bounding box and the predicted heatmap is in red. We normalize each heatmap so that the maximum value is 1.

AP^r_{part} at 0.5	Person	Horse	Cow	Sheep	Cat	Dog	Bird
Only *fc7*	21.9	16.6	14.5	38.9	19.2	8.5	**15.4**
Hyp	**28.5**	**27.8**	**21.5**	**44.9**	**30.3**	**14.2**	14.2

Table 4.2: Results on part labeling. Our approach (Hyp) is almost uniformly better than using top level features (Section 4.1).

Part labeling We evaluate part labeling on the articulated object categories in PASCAL VOC: person, horse, cow, sheep, cat, dog, bird. We use the part annotations provided by [12]. We group the parts into top-level parts: head, torso, arms and legs for person, head, torso, legs and tail for the four-legged animals and head, torso, legs, wings, tail for the bird. We train separate classifiers for each part. At test time, we use the Hyp+bbox-reg+FT system from Section 3.3 to predict a figure-ground mask for each detection, and to every pixel in the figure-ground mask, we assign the part with the highest score at that pixel.

For evaluation, we use the AP^r_{part} metric described in Section 1.2. As before, we evaluate both our system and a baseline that uses only *fc7* features. Table 4.2 shows our results. We get a large gain in almost all categories by using hypercolumns. Note that this gain is entirely due to improvements in the part labeling, since both methods use the same figure-ground mask. Figure 4.2 shows some example part labelings.

4.2 Boundary Prediction

For boundary prediction, the setup is more involved. We do a three way classification: no boundary, boundary with object on one “side”, or boundary with object on the other “side”. Because there is no universal, unambiguous definition of a “side” of a boundary, we divide boundary pixels into 5 different bins based on orientation, and train different classifiers for each orientation bin. In each

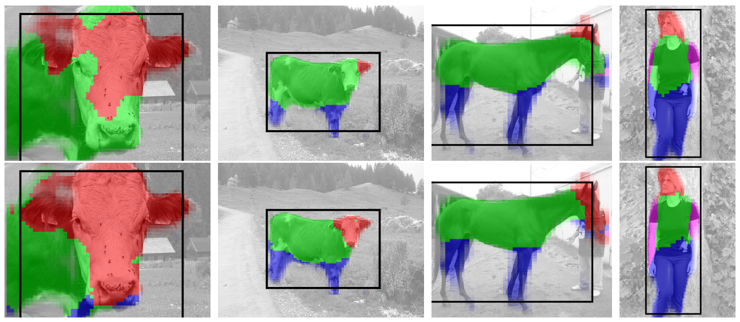

Figure 4.2: Part labeling. Top: baseline using *fc7*, bottom: ours (hypercolumns). Both rows use the same figure-ground segmentation. Red: head, green: torso, blue: legs, magenta: arms.

orientation bin, we arbitrarily define one side of the boundary as label 1 and the other side as label 2. Non-boundary pixels get a label of 0.

Another difference in relation to other tasks is that for boundary prediction, not all pixels in the detection window are a priori equally likely to be on the boundaries. Indeed, we now have bottom-up contour prediction algorithms such as UCM [1] which give highly accurate contours. Therefore we take the UCM output, threshold it at a low value (we use 0.1) and then only evaluate our classifiers on this sparse set of pixels. Further, we can also get the orientation of the contours from the contour map. We bin this orientation into the 5 bins and use the classifier corresponding to the bin in which the contour pixel falls.

During training, we collect separate datasets for each orientation bin. To get ground truth labels, we match pixels on UCM contours with ground truth using the bipartite matching algorithm proposed in [47]. For UCM contours that match, we record which side of the contour the object lies in, use that to label the contour, and add it to the dataset of the appropriate orientation bin. UCM contours that don't match are added to the dataset of the appropriate orientation bin with a label of 0.

Using boundary predictions to improve segmentation

It is intuitively clear that boundaries and segmentations are two sides of the same coin and should not be predicted completely independently. We want a segmentation that is consistent with our boundary prediction. In ensuring this consistency, we will also likely improve the segmentation.

We combine our boundary prediction with the predicted segmentation as follows. We operate on superpixels. The figure-ground prediction is projected to superpixels by averaging the prediction in each superpixel, as described in the previous chapter. This gives a probability p_i for every superpixel i. In addition, every pair of adjacent superpixels (i, j) is connected by contour pixels. For each of these contour pixels c, we have assigned probabilities to three possibilities: there is no boundary on this contour pixel, there is a boundary, and the object is on the side of i, and there is a boundary and the object is on the side of j. The last two probabilities give us two scores: $b^c_{i>j}$ indicating that i is part of the object and not j, and $b^c_{j>i}$ indicating the converse.

Metric	Without boundary	With boundary
mAP^r at 0.5	**60.0**	59.8
mAP^r at 0.7	40.4	**41.6**
mAP^{bdry} at 0.5	38.0	**42.3**
mAP^{bdry} at 0.7	15.7	**19.5**

Table 4.3: Boundary prediction in the detect-segment-rescore pipeline. Using boundary predictions improves AP^r at 0.7 slightly, but offers much larger gains in the boundary-based metric.

We can convert our boundary predictions into superpixel scores as follows. Consider a labeling of the superpixels $\mathbf{y}$, with $y_i \in \{0,1\}$ being the label for the i-th superpixel. Then, we can measure how much $\mathbf{y}$ respects the boundary prediction by evaluating:

$$f(\mathbf{y},\mathbf{b}) = \sum_{(i,j)\in\mathcal{E}} \sum_{c\in e(i,j)} (y_i - y_j) b^c_{i>j} + (y_j - y_i) b^c_{j>i} \tag{4.1}$$

$$= \sum_i y_i \sum_{j\in\mathcal{N}(i)} \sum_{c\in e(i,j)} (b^c_{i>j} - b^c_{j>i}) \tag{4.2}$$

$$= \sum_i y_i s_i^{(b)} \tag{4.3}$$

where $s_i^{(b)} = \sum_{j\in\mathcal{N}(i)} \sum_{c\in e(i,j)} (b^c_{i>j} - b^c_{j>i})$. Here $\mathcal{E}$ is the set of all pairs of superpixels that adjoin each other, $\mathcal{N}(i)$ is the set of all superpixels adjoining i, and $e(i,j)$ is the set of contour pixels separating i and j. Thus the pairwise boundary predictions can be combined into a single score for each superpixel $s_i^{(b)}$, which we can convert into a probability by passing through a sigmoid: $p_i^{(b)} = \sigma(s_i^{(b)})$. In practice, we also divide by the square root of the area of the superpixel before passing through a sigmoid, to avoid large superpixels getting large scores.

Finally, we simply average p_i and $p_i^{(b)}$ to get a probability for each prediction. Thresholding this probability gives us a new figure-ground mask.

We perform this boundary prediction using the VGG network on top of the expanded pool of bounding box detections that we get in the (slow) detect-segment-rescore pipeline described in Section 3.4. We then compare the result when using just the figure-ground predictor scores p_i, and when using both p_i and $p_i^{(b)}$. We evaluate the two alternatives using the AP^r metric. However, the AP^r metric places more emphasis on getting the interior of the object right. We therefore also evaluate on the AP^{bdry} metric proposed in Section 1.2.

The numbers are shown in Table 4.3. We observe that the boundary prediction does offer slight improvements even on the AP^r metric at an overlap threshold of 0.7. However, when compared on the boundary prediction metrics AP^{bdry}, the gap is much larger: upto 4 points at both 0.5 overlap and 0.7 overlap. Figure 4.3 visualizes cases where using the boundary prediction improves the segmentation.

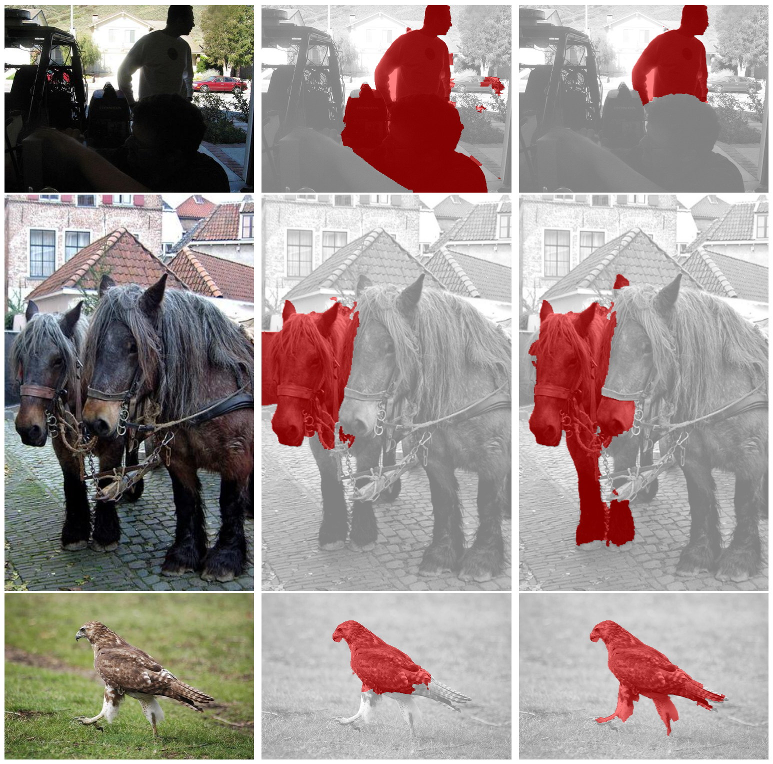
Figure 4.3: Examples of cases where boundary prediction helps. Each row shows in order the image, the segmentation without using the boundary prediction, and the segmentation with the boundary prediction.

Chapter 5

Conclusion

In this thesis we have presented algorithms that go beyond the bounding boxes or pixel labels output by typical object detection or semantic segmentation systems and produce a much richer output: a *segmentation* for each object of every category in the image. We have explored ways of using top-down knowledge to predict precise segmentations, as well as ways of using predicted segmentations to classify objects better. We have also explored how the techniques we have developed can also be used to segment out the parts of the object, locate its keypoints or precisely mark its boundaries.

However, these are first steps, and there is a lot of distance to go. Each object hypothesis is currently handled independently, and figure-ground prediction only happens within the confines of its bounding box. This means that multiple hypotheses might claim the same object and end up dividing the object in two, as in the left part of Figure 5.1. The figure-ground prediction at each pixel is done independently, which can lead to predictions that are mutually inconsistent, as exemplified by the two front wheels of the motorbike in the right half of Figure 5.1. Correcting such errors will be the target of future work.

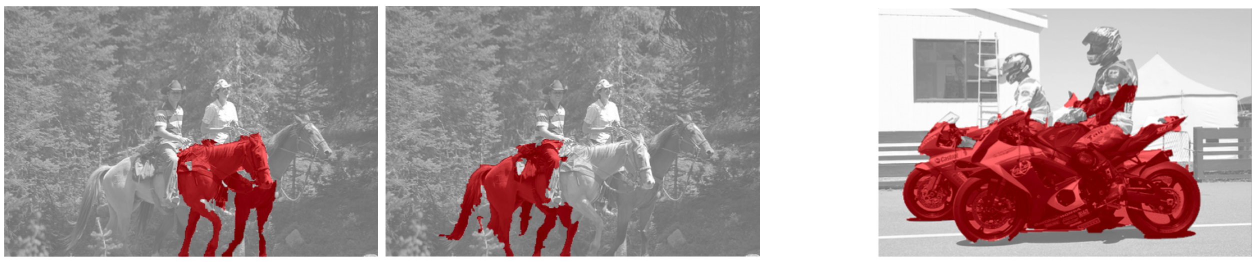

Figure 5.1: Surviving error modes. Our system sometimes breaks objects into two, or merges multiple objects.

Bibliography

[1] P. Arbeláez, J. Pont-Tuset, J. Barron, F. Marques, and J. Malik. “Multiscale Combinatorial Grouping”. In: *CVPR*. 2014.

[2] Pablo Arbeláez, Bharath Hariharan, Chunhui Gu, Saurabh Gupta, and Jitendra Malik. “Semantic Segmentation using Regions and Parts”. In: *CVPR*. 2012.

[3] Jonathan T. Barron, Pablo Arbeláez, Soile V. E. Keränen, Mark D. Biggin, David W. Knowles, and Jitendra Malik. “Volumetric Semantic Segmentation using Pyramid Context Features”. In: *ICCV* (2013).

[4] Yihang Bo and Charless C Fowlkes. “Shape-based pedestrian parsing”. In: *CVPR*. 2011.

[5] Xavier Boix, Josep M Gonfaus, Joost van de Weijer, Andrew D Bagdanov, Joan Serrat, and Jordi Gonzàlez. “Harmony potentials”. In: *IJCV* 96.1 (2012).

[6] Eran Borenstein and Shimon Ullman. “Class-specific, top-down segmentation”. In: *ECCV*. 2002.

[7] Lubomir Bourdev, Subhransu Maji, Thomas Brox, and Jitendra Malik. “Detecting people using mutually consistent poselet activations”. In: *ECCV*. 2010.

[8] Thomas Brox, Andrés Bruhn, Nils Papenberg, and Joachim Weickert. “High accuracy optical flow estimation based on a theory for warping”. In: *ECCV*. 2004.

[9] Peter J Burt and Edward H Adelson. “The Laplacian pyramid as a compact image code”. In: *Communications, IEEE Transactions on* 31.4 (1983).

[10] Joao Carreira, Rui Caseiro, Jorge Batista, and Cristian Sminchisescu. “Semantic segmentation with second-order pooling”. In: *ECCV*. 2012.

[11] Joao Carreira and Cristian Sminchisescu. “Constrained parametric min-cuts for automatic object segmentation”. In: *CVPR*. 2010.

[12] Xianjie Chen, Roozbeh Mottaghi, Xiaobai Liu, Sanja Fidler, Raquel Urtasun, and Alan Yuille. “Detect What You Can: Detecting and Representing Objects using Holistic Models and Body Parts”. In: *CVPR*. 2014.

[13] Jifeng Dai, Kaiming He, and Jian Sun. “Convolutional Feature Masking for Joint Object and Stuff Segmentation”. In: *CVPR*. 2015.

[14] Qieyun Dai and Derek Hoiem. “Learning to localize detected objects”. In: *CVPR*. 2012.

[15] Navneet Dalal and Bill Triggs. “Histograms of oriented gradients for human detection”. In: *CVPR*. 2005.

[16] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. “ImageNet: A Large-Scale Hierarchical Image Database”. In: *CVPR*. 2009.

[17] Chaitanya Desai and Deva Ramanan. “Detecting actions, poses, and objects with relational phraselets”. In: *ECCV*. 2012.

[18] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. “The Pascal Visual Object Classes (VOC) Challenge”. In: *IJCV* 88.2 (2010).

[19] Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin. “LIBLINEAR: A library for large linear classification”. In: *JMLR* 9 (2008).

[20] Clément Farabet, Camille Couprie, Laurent Najman, and Yann LeCun. “Learning hierarchical features for scene labeling”. In: *TPAMI* 35.8 (2013).

[21] Pedro F Felzenszwalb, Ross B Girshick, David McAllester, and Deva Ramanan. “Object detection with discriminatively trained part-based models”. In: *TPAMI* 32.9 (2010).

[22] Sanja Fidler, Roozbeh Mottaghi, Alan Yuille, and Raquel Urtasun. “Bottom-up segmentation for top-down detection”. In: *CVPR*. 2013.

[23] Kunihiko Fukushima. “Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position”. In: *Biological cybernetics* 36.4 (1980).

[24] R. Girshick, J. Donahue, T. Darrell, and J. Malik. “Rich feature hierarchies for accurate object detection and semantic segmentation”. In: *CVPR*. 2014.

[25] Ross Girshick. “Fast R-CNN”. In: *arXiv preprint arXiv:1504.08083* (2015).

[26] Georgia Gkioxari, Pablo Arbeláez, Lubomir Bourdev, and Jitendra Malik. “Articulated Pose Estimation using Discriminative Armlet Classifiers”. In: *CVPR*. 2013.

[27] Georgia Gkioxari, Bharath Hariharan, Ross Girshick, and Jitendra Malik. “R-CNNs for Pose Estimation and Action Detection”. In: (2014). arXiv: `1406.5212 [cs.CV]`.

[28] Georgia Gkioxari, Bharath Hariharan, Ross Girshick, and Jitendra Malik. “Using k-poselets for detecting people and localizing their keypoints”. In: *CVPR*. 2014.

[29] Bharath Hariharan, Pablo Arbeláez, Lubomir Bourdev, Subhransu Maji, and Jitendra Malik. “Semantic Contours from Inverse Detectors”. In: *ICCV*. 2011.

[30] Bharath Hariharan, Pablo Arbeláez, Ross Girshick, and Jitendra Malik. “Hypercolumns for Object Segmentation and Fine-grained Localization”. In: *CVPR*. 2015.

[31] Bharath Hariharan, Pablo Arbeláez, Ross Girshick, and Jitendra Malik. “Simultaneous Detection and Segmentation”. In: *ECCV*. 2014.

[32] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. “Spatial pyramid pooling in deep convolutional networks for visual recognition”. In: *ECCV*. 2014.

[33] Derek Hoiem, Yodsawalai Chodpathumwan, and Qieyun Dai. “Diagnosing error in object detectors”. In: *ECCV*. 2012.

[34] David H Hubel and Torsten N Wiesel. “Receptive fields, binocular interaction and functional architecture in the cat’s visual cortex”. In: *The Journal of physiology* 160.1 (1962).

[35] Catalin Ionescu, Joao Carreira, and Cristian Sminchisescu. “Iterated Second-Order Label Sensitive Pooling for 3D Human Pose Estimation”. In: *CVPR*. 2014.

[36] Yangqing Jia. *Caffe: An Open Source Convolutional Architecture for Fast Feature Embedding*. `http://caffe.berkeleyvision.org/`. 2013.

[37] David G Jones and Jitendra Malik. “Determining three-dimensional shape from orientation and spatial frequency disparities”. In: *ECCV*. 1992.

[38] B. Kim, M. Sun, P. Kohli, and Silvio Savarese. “Relating Things and Stuff by High-Order Potential Modeling”. In: *ECCV’12 Workshop on Higher-Order Models and Global Constraints in Computer Vision*. 2012.

[39] Jan J Koenderink and Andrea J van Doorn. “Representation of local geometry in the visual system”. In: *Biological cybernetics* 55.6 (1987).

[40] Philipp Krähenbühl and Vladlen Koltun. “Geodesic object proposals”. In: *ECCV*. 2014.

[41] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. “ImageNet Classification with Deep Convolutional Neural Networks.” In: *NIPS*. 2012.

[42] L’ubor Ladickỳ, Paul Sturgess, Karteek Alahari, Chris Russell, and Philip HS Torr. “What, where and how many? combining object detectors and crfs”. In: *ECCV*. 2010.

[43] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. “Backpropagation applied to handwritten zip code recognition”. In: *Neural computation* 1.4 (1989).

[44] Jonathan Long, Evan Schelhamer, and Trevor Darrell. “Fully Convolutional Networks for Semantic Segmentation”. In: *CVPR*. 2015.

[45] Ping Luo, Xiaogang Wang, and Xiaoou Tang. “Pedestrian parsing via deep decompositional network”. In: *ICCV*. 2013.

[46] Jitendra Malik and Pietro Perona. “Preattentive texture discrimination with early vision mechanisms”. In: *Journal of the Optical Society of America A* 7.5 (1990).

[47] David R Martin, Charless C Fowlkes, and Jitendra Malik. “Learning to detect natural image boundaries using local brightness, color, and texture cues”. In: *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 26.5 (2004), pp. 530–549.

[48] Roozbeh Mottaghi. “Augmenting deformable part models with irregular-shaped object patches”. In: *CVPR*. 2012.

[49] Omkar M Parkhi, Andrea Vedaldi, CV Jawahar, and Andrew Zisserman. “The truth about cats and dogs”. In: *ICCV*. 2011.

[50] Carsten Rother, Vladimir Kolmogorov, and Andrew Blake. "Grabcut: Interactive foreground extraction using iterated graph cuts". In: *ACM Transactions on Graphics (TOG)* 23.3 (2004), pp. 309–314.

[51] D.E. Rumelhart, G.E. Hinton, and R.J. Williams. "Learning Internal Representations by Error Propagation". In: *Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Volume 1: Foundations* (1986).

[52] Koen EA van de Sande, Jasper RR Uijlings, Theo Gevers, and Arnold WM Smeulders. "Segmentation as selective search for object recognition". In: *Computer Vision (ICCV), 2011 IEEE International Conference on*. 2011.

[53] Pierre Sermanet, Koray Kavukcuoglu, Soumith Chintala, and Yann LeCun. "Pedestrian detection with unsupervised multi-stage feature learning". In: *CVPR*. 2013.

[54] Karen Simonyan and Andrew Zisserman. "Very Deep Convolutional Networks for Large-Scale Image Recognition". In: (2015).

[55] Joseph Tighe, Marc Niethammer, and Svetlana Lazebnik. "Scene Parsing with Object instances and Occlusion Handling". In: *ECCV*. 2010.

[56] Jonathan Tompson, Arjun Jain, Yann LeCun, and Christoph Bregler. "Joint Training of a Convolutional Network and a Graphical Model for Human Pose Estimation". In: *NIPS (To appear)*. 2014.

[57] Alexander Toshev and Christian Szegedy. "Deeppose: Human pose estimation via deep neural networks". In: *CVPR*. 2014.

[58] Joseph Weber and Jitendra Malik. "Robust computation of optical flow in a multi-scale differential framework". In: *IJCV* 14.1 (1995).

[59] Kota Yamaguchi, M Hadi Kiapour, and Tamara L Berg. "Paper doll parsing: Retrieving similar styles to parse clothing items". In: *ICCV*. 2013.

[60] Kota Yamaguchi, M Hadi Kiapour, Luis E Ortiz, and Tamara L Berg. "Parsing clothing in fashion photographs". In: *CVPR*. 2012.

[61] Yi Yang, Sam Hallman, Deva Ramanan, and Charless C Fowlkes. "Layered object models for image segmentation". In: *TPAMI* 34.9 (2012).

[62] Yi Yang and Deva Ramanan. "Articulated human detection with flexible mixtures of parts". In: *TPAMI* 35.12 (2013).

[63] C Lawrence Zitnick and Piotr Dollár. "Edge boxes: Locating object proposals from edges". In: *ECCV*. 2014.