Hypercolumns for Object Segmentation and Fine-grained Localization

Bharath Hariharan
University of California
Berkeley
bharath2@eecs.berkeley.edu

Pablo Arbeláez
Universidad de los Andes
Colombia
pa.arbelaez@uniandes.edu.co

Ross Girshick
Microsoft Research
Redmond
rbg@microsoft.com

Jitendra Malik
University of California
Berkeley
malik@eecs.berkeley.edu

Abstract

Recognition algorithms based on convolutional networks (CNNs) typically use the output of the last layer as a feature representation. However, the information in this layer may be too coarse spatially to allow precise localization. On the contrary, earlier layers may be precise in localization but will not capture semantics. To get the best of both worlds, we define the hypercolumn *at a pixel as the vector of activations of all CNN units above that pixel. Using hypercolumns as pixel descriptors, we show results on three fine-grained localization tasks: simultaneous detection and segmentation [22], where we improve state-of-the-art from* 49.7 *mean* AP^r *[22] to* 60.0*, keypoint localization, where we get a 3.3 point boost over [20], and part labeling, where we show a 6.6 point gain over a strong baseline.*

1. Introduction

Features based on convolutional networks (CNNs) [29] have now led to the best results on a range of vision tasks: image classification [28, 36], object segmentation and detection [18, 22], action classification [35], pose estimation [37] and fine-grained category recognition [44, 6]. We have thus moved from the era of HOG and SIFT to the era of convolutional network features. Therefore, understanding these features and how best to exploit them is of wide applicability.

Typically, recognition algorithms use 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 [7].

In this paper, we think of the layers of a convolutional network as a non-linear counterpart of the image 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.) Figure 1 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 [24]. However, our hypercolumn includes not just edge detectors but also more semantic units and is thus a more general notion.

We show the utility of the hypercolumn representation

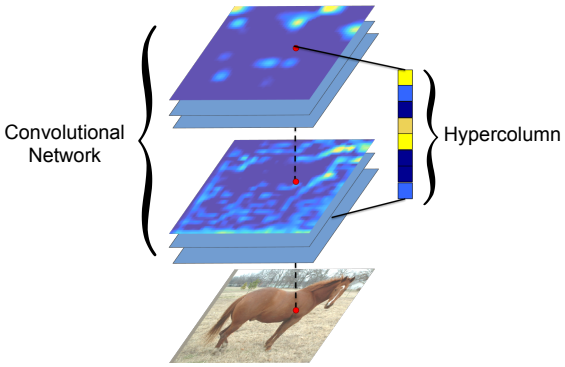

Figure 1. The hypercolumn representation. The bottom image is the input, and above it are the feature maps of different layers in the CNN. The hypercolumn at a pixel is the vector of activations of all units that lie above that pixel.

on two kinds of problems that require precise localization. The first problem is simultaneous detection and segmentation (SDS) [22], where the aim is to both detect and segment every instance of an object category in the image. The second problem deals with detecting an object and localizing its parts. We consider two variants of this: one, locating the keypoints [43], and two, segmenting out each part [41, 40, 3, 31].

We present a general framework for tackling these and other fine-grained localization tasks by framing them as pixel classification and using hypercolumns as pixel descriptors. We formulate our entire system as a neural network, allowing end-to-end training for particular tasks simply by changing the target labels. Our empirical results are:

1. On SDS, the previous state-of-the-art was 49.7 mean AP^r [22]. Substituting hypercolumns into the pipeline of [22] improves this to **52.8**. We also propose a more efficient pipeline that allows us to use a larger network, pushing up the performance to **60.0**.

2. On keypoint prediction, we show that a simple keypoint prediction scheme using hypercolumns achieves a **3.3** point gain in the APK metric [43] over prior approaches working with only the top layer features [20]. While there isn't much prior work on labeling parts of objects, we show that the hypercolumn framework is significantly better (by **6.6** points on average) than a strong baseline based on the top layer features.

2. Related work

Combining features across multiple levels: Burt and Adelson introduced Laplacian pyramids [8], a representation that is widely used in computer vision. Koenderink and van Doorn [27] used "jets", which are sets of partial derivatives of intensity up to a particular order, to estimate edge orientation, curvature, etc. Malik and Perona [32] used the output of a bank of filters as a representation for texture discrimination. This representation also proved useful for optical flow [39] and stereo [26]. 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. [15] combine CNN outputs from multiple scales of an image to do semantic segmentation. Tompson et al. [37] 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. [34] 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. [30] also use multiple layers for their fully convolutional semantic segmentation system.

Detection and segmentation: The task of simultaneous detection and segmentation task, introduced in [22], requires one to detect and segment every instance of a category in the image. SDS differs from classical bounding box detection in its requirement of a segmentation and from classical semantic segmentation in its requirement of separate instances. There has been other prior work on segmenting out instances of a category, mostly starting from bounding box detections. Borenstein and Ullman [4] first suggested the idea of using class-specific knowledge for segmentation. Yang et al. [42] use figure ground masks associated with DPM detectors [16] to segment out detected objects and reason about depth orderings. Parkhi et al. [33] use color models extracted from the detected cat and dog heads to segment them out. Dai and Hoiem [12] generalize this reasoning to all categories. Fidler et al. [17] and Dong et al. [13] combine object detections from DPM [16] with semantic segmentation outputs from O_2P [9] to improve both systems. Current leading methods use CNNs to score bottom-up object proposals, both for object detection [18] and for SDS [22, 11].

Pose estimation and part labeling: Current best performers for pose estimation are based on CNNs. Toshev and Szegedy [38] use a CNN to regress to keypoint locations. Tompson et al. [37] 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 [43] 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. [21] show some results in this setting using HOG-based detectors, but in their later work [20] 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. [41, 40] parse fashion photographs into clothing items. There has also been work on parsing pedestrians [3, 31]. Ionescu et al. [25] 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.

3. Pixel classification using hypercolumns

Problem setting: We assume an object detection system that gives us a set of detections. Each detection comes with a bounding box, a category label and a score (and sometimes an initial segmentation hypothesis). The detections have already been subjected to non-maximum suppression. For every detection, we want to segment out the object, segment its parts or predict its keypoints.

For each task, we expand the bounding box of the detection slightly and predict a heatmap on this expanded box. The type of information encoded by this heatmap depends on the particular task. For segmentation, the heatmap encodes the probability that a particular location is inside the object. For part labeling, we predict a separate heatmap for each part, where each heatmap is the probability a location belongs to that part. For keypoint prediction, again we output a separate heatmap for each keypoint, with each heatmap encoding the probability that the keypoint is at a particular location.

In each case, we predict a 50×50 heatmap that we resize to the size of the expanded bounding box and splat onto the image. Thus, in our framework, these diverse fine-grained localization problems are addressed as the unified task of 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 [18]. 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 1). 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:

$$\mathbf{f}_i = \sum_k \alpha_{ik} \mathbf{F}_k \tag{1}$$

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 [28] would lead to a 4736 dimensional vector.

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 1:

$$h_i(\cdot) = \sum_k \alpha_{ik} g_k(\cdot) \tag{2}$$

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

$$p_i = \sum_k \alpha_{ik} g_k(\mathbf{f}_i) = \sum_k \alpha_{ik} p_{ik} \tag{3}$$

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 [14] 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:

$$\mathbf{w}^T \mathbf{f}_i = \sum_j \mathbf{w}^{(j)T} \mathbf{f}_i^{(j)} \tag{4}$$

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 1:

$$\mathbf{f}_i^{(j)} = \sum_k \alpha_{ik}^{(j)} \mathbf{F}_k^{(j)} \tag{5}$$

$$\mathbf{w}^{(j)T} \mathbf{f}_i^{(j)} = \sum_k \alpha_{ik}^{(j)} \mathbf{w}^{(j)T} \mathbf{F}_k^{(j)} \tag{6}$$

We note that this insight was also used by Barron et al. [2] 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 2. 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 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 grid classifiers together and use classifier in-

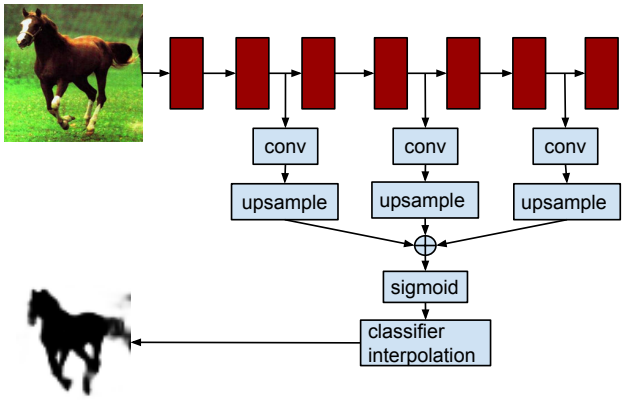

Figure 2. Representing our hypercolumn classifiers as a neural network. Layers of the original classification CNN are shown in red, and layers that we add are in blue.

terpolation during training, instead of training separate grid classifiers independent of each other.

Training classifiers for segmentation and part localization: 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 (SDS, keypoint prediction or part labeling), we then use the labeling of the cropped ground truth instance to label locations in the expanded bounding box as positive or negative. For SDS, locations inside the instance are considered positive, while locations outside are considered 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.

4. Experiments on SDS

Our first testbed is the SDS task. Our baseline for this task is the algorithm presented in [22]. This pipeline scores bottom-up region proposals from [1] using CNN features computed on both the cropped bounding box of the region and the cropped region foreground. The regions are subjected to non-max suppression. Finally, the surviving candidates are refined using figure-ground predictions based on the top layer features.

As our first system for SDS, we use the same pipeline as above, but replace the refinement step with one based on hypercolumns. (We also add a bounding box regression step [18] so as to start from the best available bounding box). We present results with this pipeline in section 4.1, where we show that hypercolumn-based refinement is significantly better than the refinement in [22], and is especially accurate when it comes to capturing fine details of the segmentation. We also evaluate several ablations of our system to unpack this performance gain. For ease of reference, we call this **System 1**.

One issue with this system is its computational cost. Extracting features from region foregrounds is expensive and doubles the time taken. Further, while CNN-based bounding box detection [18] can be speeded up dramatically using approaches such as [23], no such speedups exist for region classification. To address these drawbacks, we propose as our second system the pipeline shown in Figure 3. This pipeline starts with bounding box detections after non-maximum suppression. We expand this set of detections by adding nearby high-scoring boxes that were removed by non-maximum suppression but may be better localized (explained in detail below). 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. We call this system **System 2**.

This pipeline relies crucially on our ability to predict a good segmentation from just bounding boxes. We use hypercolumns to make this prediction. In section 4.2, we show that these predictions are accurate, and significantly better than predictions based on the top layer of the CNN.

Finally, the efficiency of this pipeline also allows us to experiment with larger but more expressive architectures. While [22] used the architecture proposed by Krizhevsky et al. [28] (referred to as "T-Net" henceforth, following [19]) for both the box features and the region features, we show in section 4.2 that the architecture proposed by Simonyan and Zisserman [36] (referred to as "O-Net" henceforth [19]) is significantly better.

4.1. System 1: Refinement using hypercolumns

In our first set of experiments, we compare a hypercolumn-based refinement to that proposed in [22]. We use the ranked hypotheses produced by [22] and refine each hypothesis using hypercolumns. For the CNN, we use the same network that was used for the region classification (described as **C** in [22]). This network consists of two pathways, each based on T-Net. It takes in both the cropped bounding box as well as the cropped foreground. 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

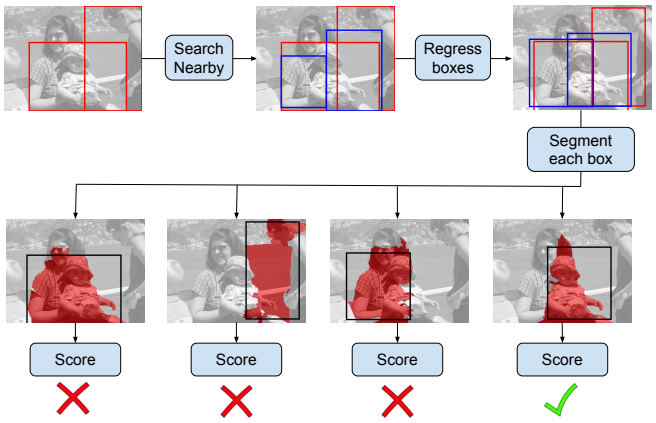

Figure 3. An alternative pipeline for SDS starting from bounding box detections (Section 4)

set of features. In addition, for each location, we add as features a 0 or 1 encoding if the location was inside the original region candidate, and a coarse 10×10 discretization of the original candidate flattened into a 100-dimensional vector. This is to be commensurate with [22] where these features were used in the refinement step. 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 1 shows the results of our experiments. The first two columns show the performance reported in [22] with and without the refinement step. "Hyp" is the result we get using hypercolumns, without bounding box regression or finetuning. Our mean AP^r at 0.5 is **1.5** points higher, and at 0.7 is **6.3** points higher, indicating that our refinement is much better than that of [22] and is a large improvement over the original candidate. Bounding box regression and finetuning the network both provide significant gains, and with both of these, our mean AP^r at 0.5 is **3.1** points higher and at 0.7 is **8.4** points higher than [22].

Table 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 and is thus similar to the refinement step in [22]. We include this baseline to confirm that we can replicate those results.

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 [22] 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.

Finally, following [22], we take our Hyp+FT+bbox-reg system and use the pasting scheme of [9] to obtain a semantic segmentation. We get a mean IU of **54.6** on VOC2012 Segmentation Test, 3 points higher than [22] (51.6 mean IU).

4.2. System 2: SDS from bounding box detections

For our experiments with System 2, we use the detections of R-CNN [18] as the starting point. R-CNN uses CNNs to classify bounding box proposals from selective search. We use the final output after non-max suppression and bounding box regression. However, to allow direct comparison with our previous experiments, we retrained R-CNN to work with box proposals from MCG [1]. 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 T-Net architecture. 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.

We show results in Table 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, our performance is nearly as good as [22] at 0.5 and about 4 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 4 shows example segmentations.

We now replace the T-Net architecture for the O-Net architecture. This architecture is significantly larger, but provides an 8 point gain in detection AP [19]. 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 O-Net architecture is significantly better than the T-Net: 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 about 6.8 and 11.7 points better than the results of [22]. 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 O-Net architecture. This implies that the gain provided by hypercolumns is not specific to a particular network architecture. Figure 4 visualizes our O-Net results.

We now implement the full pipeline proposed in Figure 3. For this, we expand the initial pool of detections as follows. We pick boxes with score higher than a threshold that were suppressed by NMS but that overlap the detections by less than 0.7. We then do a non-max suppression with a lenient threshold of 0.7 to get a pool of candidates to rescore. Starting from 20K initial detections per category across the dataset, our expanded pool is typically less than 50K per category, and less than 600K in total.

Next we segment each candidate using hypercolumns and score it using a CNN trained to classify regions. This network has the same architecture as O-Net. However, instead of a bounding box, this network takes as input the bounding box with the region background masked out. This network is trained as described in [22]. 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, [11] get slightly higher performance at 0.5 but do not report the performance at 0.7; our gains are orthogonal to theirs). Finally, on the semantic segmentation benchmark, we get a mean IU of **62.6**, which is comparable to state-of-the-art.

5. Experiments on part localization

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 [38, 37, 41, 40, 3, 31], which operates on the immediate vicinity of ground-truth instances. We start from the detections of [22]. We use the same features and network as in Section 4.1. As before, we do all training on VOC2012 Train.

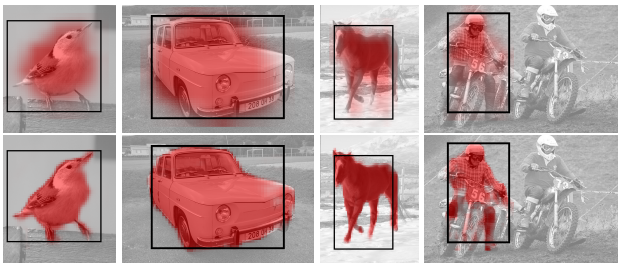

Figure 4. Figure ground segmentations starting from bounding box detections. Top row: baseline using *fc7*, bottom row: Ours.

Metric	T-Net Only *fc7*	T-Net Hyp	O-Net Only *fc7*	O-Net Hyp	O-Net 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 2. Results on SDS on VOC 2012 val using System 2. Our final pipeline is state-of-the-art on SDS. (Section 4.2)

Keypoint prediction We evaluate keypoint prediction on the "person" category using the protocol described in [21]. The test set for evaluating keypoints is the person images in the second half of VOC2009 val. We use the APK metric [43], 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 [22]. We use bounding box regression to start from a better bounding box. As described in Section 3 we train a separate system for each keypoint using the hypercolumn representation. We use keypoint annotations collected by [5]. 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 3. We compare our perfor-

Metric	[22]	[22] refined	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	49.7	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	25.3	31.6	32.4	**33.7**	25.8	30.6	31.2	30.8	28.8	30.2	31.8

Table 1. Results on SDS on VOC2012 val using System 1. Our system (Hyp+FT+bbox-reg) is significantly better than [22] (Section 4.1).

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**
[21]	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*
[20]	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 3. Results on keypoint prediction (APK on the Person subset of VOC2009 val). Our system is 3.3 points better than [20] (Section 5).

Figure 5. 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. Results on part labeling. Our approach (Hyp) is almost uniformly better than using top level features (Section 5).

mance to [20], the previous best on this dataset. Gkioxari et al. [20] 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 [21]. This confirms that the gains we get are from the hypercolumn representation. Figure 5 shows some example predictions.

Finetuning the network as described in Section 3 gives an additional **1.5** point gain, raising mean APK to **18.5**.

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 [10]. 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 4.1 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.

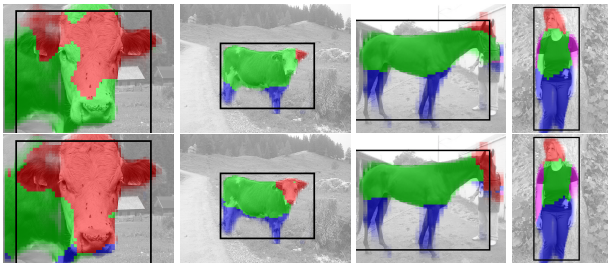

Figure 6. Part labeling. Top: baseline using *fc7*, bottom: ours (hypercolumns). Both rows use the same figure-ground segmentation. Red: head, green: torso, blue: legs, magenta: arms.

For evaluation, we modify the definition of intersection-over-union in the AP^r metric [22]: we count in the intersection only those pixels for which we also get the part label correct. We call this metric AP^r_{part}. As before, we evaluate both our system and a baseline that uses only *fc7* features. Table 4 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 6 shows some example part labelings.

6. Conclusion

We have shown that the hypercolumn representation provides large gains in three different tasks. We also believe that this representation might prove useful for other fine-grained tasks such as attribute or action classification. We leave an investigation of this to future work.

Acknowledgments. This work was supported by ONR MURI N000141010933, a Google Research Grant and a Microsoft Research fellowship. We thank NVIDIA for providing GPUs through their academic program.

References

[1] P. Arbeláez, J. Pont-Tuset, J. Barron, F. Marques, and J. Malik. Multiscale combinatorial grouping. In *CVPR*, 2014.

[2] J. T. Barron, P. Arbeláez, S. V. E. Keränen, M. D. Biggin, D. W. Knowles, and J. Malik. Volumetric semantic segmentation using pyramid context features. *ICCV*, 2013.

[3] Y. Bo and C. C. Fowlkes. Shape-based pedestrian parsing. In *CVPR*, 2011.

[4] E. Borenstein and S. Ullman. Class-specific, top-down segmentation. In *ECCV*. 2002.

[5] L. Bourdev, S. Maji, T. Brox, and J. Malik. Detecting people using mutually consistent poselet activations. In *ECCV*, 2010.

[6] S. Branson, G. Van Horn, P. Perona, and S. Belongie. Improved bird species recognition using pose normalized deep convolutional nets. In *BMVC*, 2014.

[7] T. Brox, A. Bruhn, N. Papenberg, and J. Weickert. High accuracy optical flow estimation based on a theory for warping. In *ECCV*. 2004.

[8] P. J. Burt and E. H. Adelson. The laplacian pyramid as a compact image code. *Communications, IEEE Transactions on*, 31(4), 1983.

[9] J. Carreira, R. Caseiro, J. Batista, and C. Sminchisescu. Semantic segmentation with second-order pooling. In *ECCV*, 2012.

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

[11] J. Dai, K. He, and J. Sun. Convolutional feature masking for joint object and stuff segmentation. In *CVPR*, 2015.

[12] Q. Dai and D. Hoiem. Learning to localize detected objects. In *CVPR*, 2012.

[13] J. Dong, Q. Chen, S. Yan, and A. Yuille. Towards unified object detection and semantic segmentation. In *European Conference on Computer Vision (ECCV)*, 2014.

[14] R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin. Liblinear: A library for large linear classification. *JMLR*, 9, 2008.

[15] C. Farabet, C. Couprie, L. Najman, and Y. LeCun. Learning hierarchical features for scene labeling. *TPAMI*, 35(8), 2013.

[16] P. F. Felzenszwalb, R. B. Girshick, D. McAllester, and D. Ramanan. Object detection with discriminatively trained part-based models. *TPAMI*, 32(9), 2010.

[17] S. Fidler, R. Mottaghi, A. Yuille, and R. Urtasun. Bottom-up segmentation for top-down detection. In *CVPR*, 2013.

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

[19] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. *arXiv preprint arXiv:1409.1556*, 2014.

[20] G. Gkioxari, B. Hariharan, R. Girshick, and J. Malik. R-CNNs for pose estimation and action detection. 2014.

[21] G. Gkioxari, B. Hariharan, R. Girshick, and J. Malik. Using k-poselets for detecting people and localizing their keypoints. In *CVPR*, 2014.

[22] B. Hariharan, P. Arbeláez, R. Girshick, and J. Malik. Simultaneous detection and segmentation. In *ECCV*, 2014.

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

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

[25] C. Ionescu, J. Carreira, and C. Sminchisescu. Iterated second-order label sensitive pooling for 3d human pose estimation. In *CVPR*, 2014.

[26] D. G. Jones and J. Malik. Determining three-dimensional shape from orientation and spatial frequency disparities. In *ECCV*, 1992.

[27] J. J. Koenderink and A. J. van Doorn. Representation of local geometry in the visual system. *Biological cybernetics*, 55(6), 1987.

[28] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, 2012.

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

[30] J. Long, E. Schelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. In *CVPR*, 2015.

[31] P. Luo, X. Wang, and X. Tang. Pedestrian parsing via deep decompositional network. In *ICCV*, 2013.

[32] J. Malik and P. Perona. Preattentive texture discrimination with early vision mechanisms. *Journal of the Optical Society of America A*, 7(5), 1990.

[33] O. M. Parkhi, A. Vedaldi, C. Jawahar, and A. Zisserman. The truth about cats and dogs. In *ICCV*, 2011.

[34] P. Sermanet, K. Kavukcuoglu, S. Chintala, and Y. LeCun. Pedestrian detection with unsupervised multi-stage feature learning. In *CVPR*, 2013.

[35] K. Simonyan and A. Zisserman. Two-stream convolutional networks for action recognition in videos. *arXiv preprint arXiv:1406.2199*, 2014.

[36] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[37] J. Tompson, A. Jain, Y. LeCun, and C. Bregler. Joint training of a convolutional network and a graphical model for human pose estimation. In *NIPS (To appear)*, 2014.

[38] A. Toshev and C. Szegedy. Deeppose: Human pose estimation via deep neural networks. In *CVPR*, 2014.

[39] J. Weber and J. Malik. Robust computation of optical flow in a multi-scale differential framework. *IJCV*, 14(1), 1995.

[40] K. Yamaguchi, M. H. Kiapour, and T. L. Berg. Paper doll parsing: Retrieving similar styles to parse clothing items. In *ICCV*, 2013.

[41] K. Yamaguchi, M. H. Kiapour, L. E. Ortiz, and T. L. Berg. Parsing clothing in fashion photographs. In *CVPR*, 2012.

[42] Y. Yang, S. Hallman, D. Ramanan, and C. C. Fowlkes. Layered object models for image segmentation. *TPAMI*, 34(9), 2012.

[43] Y. Yang and D. Ramanan. Articulated human detection with flexible mixtures of parts. *TPAMI*, 35(12), 2013.

[44] N. Zhang, J. Donahue, R. Girshick, and T. Darrell. Part-based R-CNNs for fine-grained category detection. In *ECCV*, 2014.