Int J Comput Vis (2017) 124:115–131
DOI 10.1007/s11263-017-1006-x

DeepProposals: Hunting Objects and Actions by Cascading Deep Convolutional Layers

Amir Ghodrati[1] · **Ali Diba**[2] · **Marco Pedersoli**[3] · **Tinne Tuytelaars**[2] · **Luc Van Gool**[2]

Received: 3 May 2016 / Accepted: 6 March 2017 / Published online: 15 March 2017

Abstract In this paper, a new method for generating object and action proposals in images and videos is proposed. It builds on activations of different convolutional layers of a pretrained CNN, combining the localization accuracy of the early layers with the high informativeness (and hence recall) of the later layers. To this end, we build an inverse cascade that, going backward from the later to the earlier convolutional layers of the CNN, selects the most promising locations and refines them in a coarse-to-fine manner. The method is efficient, because (i) it re-uses the same features extracted for detection, (ii) it aggregates features using integral images, and (iii) it avoids a dense evaluation of the proposals thanks to the use of the inverse coarse-to-fine cascade. The method is also accurate. We show that DeepProposals outperform most of the previous object proposal and action proposal approaches and, when plugged into a CNN-based object detector, produce state-of-the-art detection performance.

Keywords Object Proposals · Action proposals · Object detection · Action localization

1 Introduction

In recent years, the paradigm of generating a reduced set of window candidates to be evaluated with a powerful classifier has become very popular in object detection. Indeed, most of the recent state-of-the-art detection methods (Cinbis et al. 2013; Ren et al. 2015; He et al. 2015; Wang et al. 2013) are based on such proposals. Furthermore, generating a limited number of proposals also helps weakly supervised object detection, which consists of learning to localize objects without any bounding box annotations (Deselaers et al. 2010; Song et al. 2014; Bilen et al. 2015).

Detection methods based on proposals can be seen as a two-stage cascade: first, a reduced set of promising and class-independent hypotheses, the *proposals*, are selected; and second, each hypothesis is classified in a class-specific manner. Similarly to sliding windows, this pipeline casts the detection problem to a classification problem. However, in contrast to sliding windows, more powerful and time consuming detectors can be employed at the class-specific stage, as the number of candidate windows is reduced.

Methods proposed in the literature for the generation of window candidates are based on two different approaches. The first approach uses bottom-up cues like image segmentation (Arbelaez et al. 2014; Van de Sande et al. 2011), object edges and contours (Zitnick and Dollár 2014) for window generation. The second approach is based on top-down cues which learn to separate correct object hypotheses from other possible window locations (Alexe et al. 2010; Cheng et al. 2014). So far, the latter strategy seems to have inferior performance. In this paper we show that, with the proper features, accurate and fast top-down window proposals can be generated.

We consider for this task the convolutional neural network (CNN) "feature maps" extracted from the intermediate layers of pretrained Alexnet-like (Krizhevsky et al. 2012) networks. We observe that classifiers trained on deeper convolutional layers, which form a more semantic representation, perform very well in recalling the objects with a reduced set

Communicated by Antonio Torralba.

✉ Amir Ghodrati
a.ghodrati@uva.nl

[1] QUVA Lab, University of Amsterdam, Amsterdam, Netherlands

[2] ESAT-PSI, iMinds, KU Leuven, Leuven, Belgium

[3] École de technologie supérieure, Montreal, Canada

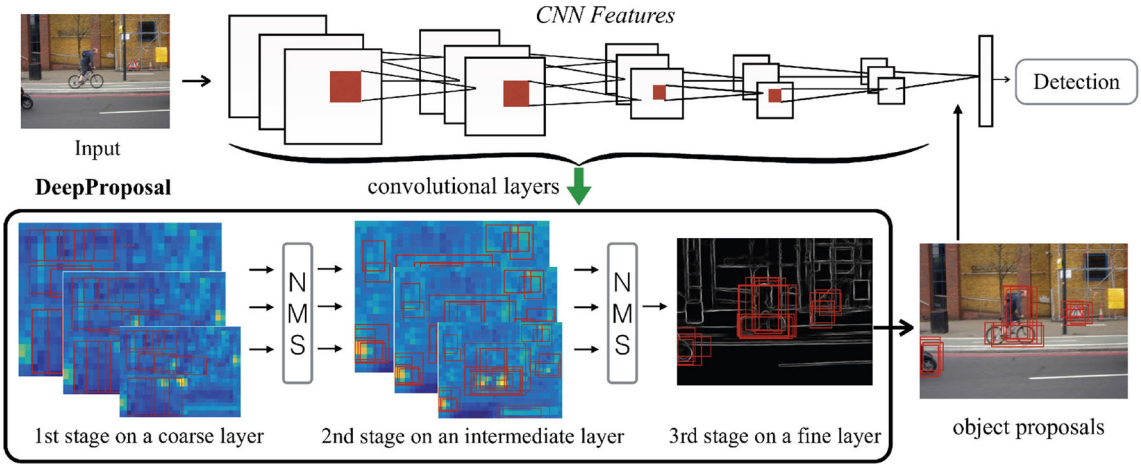

Fig. 1 DeepProposals pipeline. Our method uses the activation layers of a deep convolutional neural network in a coarse-to-fine inverse cascading to obtain proposals for object detection. Starting from a dense proposal sampling in the last convolutional layer, we gradually filter out irrelevant boxes until reaching the initial layers of the net. In the last stage we use contours extracted from an earlier layer with fine feature maps, to refine the proposals. Finally the generated boxes can be used within an object detection pipeline

of hypotheses. Unfortunately, as noticed also for other tasks (Hariharan et al. 2014), these layers provide a poor localization of the object due to their coarseness. In contrast, earlier layers are better in accurately localizing the object of interest, but their recall is reduced as they do not contain strong object cues. Here we propose a method for generating object proposals based itself on a cascade, starting from the last convolutional layer and going down with subsequent refinements to the initial layers of the net. As the flow of the cascade is inverse to the flow of the feature computation we call this an *inverse cascade*. Also, as we start from a coarse spatial window resolution, and throughout the layers we select and spatially refine the window hypotheses until we obtain a reduced and spatially well localized set of hypotheses, it is a *coarse-to-fine inverse cascade*. An overview of our approach, which we coined *DeepProposals*, is illustrated in Fig. 1.

More specifically, similarly to BING (Cheng et al. 2014), we select a reduced set of window sizes and aspect ratios and slide them on each possible location of the feature map generated by the last convolutional layer of a CNN. The relevance (or objectness) of the windows is evaluated using a linear classifier. As the proposal generation procedure should be fast, we base the feature aggregation for each candidate window on average pooling, which can be computed in constant time using integral images (Viola and Jones 2004). We filter out boxes that are less likely to contain an object and propagate the remaining ones to the next stage. In the second stage, we move to an earlier convolutional layer—in particular, the earliest layer that has the same feature map size as used in the first stage, and further filter the set of candidate boxes. As the number of boxes to evaluate has been reduced, we can afford adding more geometry in the representation by encoding each window with a pyramid representation. We rerank the boxes and select the best N of them to pass to the last stage. Finally, in the third stage, we refine the localization obtained from the previous stage of the cascade, using an edgemap obtained from the second layer of the CNN.

We use the above framework not just for static images, but extend it to video, generating *action proposals*. To this end, we first apply the coarse-to-fine inverse cascade on each frame of a video. Then, we group the proposals into tubes, by imposing time continuity constraints, based on the assumption that the object of interest has a limited speed. We show that such proposals can provide excellent results for action localization.

We evaluate the performance of the DeepProposals in terms of recall versus number of proposals as well as in terms of recall versus object (action) overlap. We show that in both evaluations the method is better than the current state of the art, and computationally very efficient. However, the biggest gains are achieved when using the method as part of a CNN-based detector like the Fast R-CNN proposed by Girshick (2015). The features extracted from the generation of proposals will then be also the same features to be used for object detection. Thus, we can execute the full detection pipeline at a very low computational cost.

This paper is an extension of our earlier work (Ghodrati et al. 2015). In this version we include some additional related work; we apply the same framework to deeper network architectures; we evaluate it on more data sets and compare the method against some recent approaches. In addition, we extend DeepProposals for generating action proposals in videos and compare it with other state-of-the-art methods on two action datasets.

In the next section, we describe the most related work. Next, in Sect. 3, we describe our proposed inverse coarse-to-fine cascade, followed by an detailed description of its components in Sect. 4. In Sect. 5, the extension of our method to action proposals is explained. Section 6 covers experiments for object proposals. It consists of an in-depth analysis of different components of the cascade and quantitative as well as qualitative comparison of our method with the state-of-the-art. In Sect. 7, experiments for action proposal generation are described. Section 8 concludes the paper.

2 Related Work

2.1 Object Proposal Methods

Object proposal generators aim at obtaining an accurate object localization with few object window hypotheses. These proposals can help object detection in two ways: searching objects in fewer locations to reduce the detector running time and/or using more sophisticated and expensive models to achieve better performance.

A first set of approaches measures the objectness of densely sampled windows (i.e. how likely is it for an image window to represent an object) (Alexe et al. 2010; Cheng et al. 2014; Zitnick and Dollár 2014). Alexe et al. (2010) propose a measure based on image saliency and other cues like color and edges to discriminate object windows from background. BING (Cheng et al. 2014) is a very fast proposal generator, obtained by training a classifier on edge features, but it suffers from low localization accuracy. Moreover, Zhao et al. (2014) have shown that the BING classifier has minimal impact on locating objects and without looking at the actual image a similar performance can be obtained. Edge-boxes (Zitnick and Dollár 2014) uses the structural edges of (Dollár and Zitnick 2013), a state-of-the-art contour detector, to compute proposal scores in a sliding window fashion without any parameter learning. For a better localization they use a final window refinement step. Like these methods, our approach densely samples hypotheses in a sliding window fashion. However, in contrast to them, we use a hierarchy of high-to-low level features extracted from a deep CNN which has proven to be effective for object detection (Girshick et al. 2014; Wang et al. 2013).

An alternative approach to sliding-window methods are the segmentation-based algorithms. This approach extracts from the image multiple levels of bottom-up segmentation and then merges the generated segments in order to generate object proposals (Arbelaez et al. 2014; Carreira and Sminchisescu 2012; Manen et al. 2013; Van de Sande et al. 2011). The first and most widely used segmentation-based algorithm is selective search (Van de Sande et al. 2011). It hierarchically aggregates multiple segmentations in a bottom-up greedy manner without involving any learning procedure, but based on low level cues, such as color and texture. Multiscale combinatorial grouping (MCG) (Arbelaez et al. 2014) extracts multiscale segmentations and merges them by using the edge strength in order to generate object hypotheses. Carreira and Sminchisescu (2012) propose to segment the object of interest based on graph-cut. It produces segments from randomly generated seeds. As in selective search, each segment represents a proposal bounding box. Randomized Prim's (Manen et al. 2013) uses the same segmentation strategy as selective search. However, instead of merging the segments in a greedy manner it learns the probabilities for merging, and uses those to speed up the procedure. Geodesic object proposals (Krähenbühl and Koltun 2014) are based on classifiers that place seeds for a geodesic distance transform on an over-segmented image.

Recently, following the great success of CNN in different computer vision tasks, CNN-based methods have been used to either generate proposals or directly regress the coordinates of the object bounding box. MultiBox (Erhan et al. 2014) proposes a network which directly regresses the coordinates of all object bounding boxes (without a sliding window fashion approach) and assigns a confidence score for each of them in the image. However, MultiBox is not translation invariant and it does not share features between the proposal and detection networks i.e. it dedicates a network just for generating proposals. DeepMask (Pinheiro et al. 2015) and its next generation, SharpMask (Pinheiro et al. 2016), learn segmentation proposals by training a network to predict a class-agnostic mask for each image patch and an associated score. Similar to us, SharpMask uses a top-down refinement approach, utilizing features at lower layers to refine and generate segmentation masks with double the spatial resolution. But again, they do not share features between the proposal generation and detection. Moreover, they need segmentation annotations to train their network. OverFeat (Sermanet et al. 2013) is a method where proposal generation and detection are combined in one-stage. In OverFeat, region-wise features are extracted from a sliding window and are used to simultaneously determine the location and category of the objects. In contrast to it, our goal is to predict class-agnostic proposals which can be used in a second stage for class-specific detections.

Probably, the most similar to our work is the concurrent work of region proposal networks (RPN) proposed by Ren et al. (2015). RPN is a convolutional network that simultaneously predicts object bounds and objectness scores at each position. To generate region proposals, they slide a small network over the convolutional feature map. At each sliding-window location, they define k reference boxes (anchors) at different scales and aspect ratios and predict multiple region proposals parameterized relative to the anchors. Similarly to us, RPN builds on the convolutional features of a detec-

tion network. However, we leverage low-level features in early layers of the network to improve the localization quality of proposals. Both methods are based on a strategy to avoid a dense and expensive scan of all image windows. RPN regresses boxes from a pre-defined set of achor boxes. In our case instead, we avoid computation by using a cascade on some of the network convolutional layers.

2.2 Action Proposal Methods

Action proposals are 3D boxes or temporal tubes extracted from videos that can be used for action localization, i.e. predicting the action label in a video and spatially localizing it. Also in this case, the main advantage of using proposals is to reduce the computational cost of the task and therefore make the method faster or allow for the use of more powerful classification approaches. The action proposal methods proposed in the literature to date mainly extend ideas originally developed for 2D object proposals in static images to 3D space. (Jain et al. 2014) is an extension of selective search (Van de Sande et al. 2011) to video. It extracts super-voxels instead of super-pixels from a video and by hierarchical grouping it produces spatio-temporal tubes. Bergh et al. (2013) is an action proposal method inspired by the objectness method (Alexe et al. 2010), while a spatio-temporal variant of randomized Prim's (Manen et al. 2013) is proposed in Oneata et al. (2014). Since most of those methods are based on a super-pixel segmentation approach as a pre-processing step, they are computationally very expensive. To avoid such computationally demanding pre-processing, van Gemert et al. (2015) proposed action localization proposals (APT) which use the same features used in detection to generate action proposals.

Several action localization methods use 2D proposals in each frame without generating intermediate action proposals at video-level. Typically, they leverage 2D object proposals that are generated separately for each frame in order to find the most probable path of bounding boxes across time for each action class separately (Gkioxari and Malik 2015; Tran et al. 2014; Weinzaepfel et al. 2015; Yu and Yuan 2015). Our method is similar to these works in spirit. However, these methods use class-specific detectors for action localization while we propose a class-agnostic method to generate a reduced set of action proposals. The idea of using class-agnostic proposals allows us to filter out many negative tubes with a reduced computational time which enables the use of more powerful classifiers in the final stage.

3 Overview of the Method

An overview of our inverse coarse-to-fine cascade is illustrated in Fig. 1. We first explain our method for static images, then later extend it to video in Sect. 5.

We start the search for object proposals in the top convolutional layer of the net. The feature maps at this layer have features well adapted to recognize high-level concepts like objects and actions, but have limited resolution. This coarseness leads to an efficient sliding window, yet results in poor localization results. We then gradually move to the lower layers, that use simpler features but have a much finer spatial representation of the image. As we go from a coarse to a fine representation of the image and we follow a flow that is exactly the opposite of how those features are computed we call this approach *coarse-to-fine inverse cascade*. We found that a cascade with 3 layers is an optimal trade-off between complexity of the method and gain obtained from the cascading strategy. In the rest of this section we describe in detail the three stages of the cascade.

3.1 Stage 1: Dense Sliding Window on a Coarse Layer

The first stage of the cascade uses the activation map of a one of the last convolutional layers of the network. This implies a high semantic representation, but also coarseness due to the multiple max pooling steps used in the network. As the feature representation is coarse, we can afford a dense sliding window approach with 50 different window sizes that best cover the training data in terms of size and aspect ratio. For details, see Sect. 4 (sliding window). The base descriptor of a given window is the sum pooling of the map activations that fall inside the window in the spatial dimension. We augment this descriptor with some information about the size and the aspect ratio of the window as detailed in Sect. 4 (bias on size and aspect ratio). The computation of the descriptor is carried out in a fast and size-independent way using an integral image representation. For details, see Sect. 4 (pooling). The scores of each window are the dot product of the window descriptor and the learned weights of a linear SVM classifier trained for discriminating between object and background, see Sect. 4 (classifier). We linearly re-scale the window scores to [0, 1] such that the lowest and highest scores are mapped to 0 and 1 respectively. Afterwards we select the best $N_1 = 4000$ windows obtained from a non-maximum suppression algorithm [see Sect. 4 (non-maximum suppression)] before propagating them to the next stage.

3.2 Stage 2: Re-scoring Windows on an Intermediate Layer

At this stage, as we use a reduced set of windows, we can afford to spend more computation time per window. Therefore we add more geometry in the representation by encoding each window with a pyramid representation composed of two levels: 1×1 and 2×2, as described in Sect. 4 (pyramid). As in the first stage, we train a linear classifier with the aim to classify object versus background. The proposal scores from

this layer are again mapped to [0, 1]. The final score for each proposal is obtained by multiplying the scores of both stages. Afterwards we apply a non-maximum suppression with overlap threshold $\beta + 0.05$ and select the 3000 best candidates. At the end of this stage, we aggregate the boxes from different scales using non-maximum suppression with threshold β and select the $N_{desired} = 1000$ best for refinement.

3.3 Stage 3: Local Refinement on a Fine Layer

The main objective of this final stage is to refine the localization obtained from the previous stage of the cascade. For this stage we need higher resolution convolutional features in order to grasp the low-level information which is suitable for the refinement task. Specifically, we refine the $N_{desired}$ windows received from the previous stage using the procedure explained in Zitnick and Dollár (2014) [see Sect. 4 (refinement)]. To this end, we train a structured random forest (Dollár and Zitnick 2013) on the second layer of the convolutional features to estimate contours similarly to DeepContour (Xinggang et al. 2015). After computing the edge map, a greedy iterative search tries to maximize the score of a proposal over different locations and aspect ratios. It is worth mentioning that since our contour detector is based on the CNN-features, we again do not need to extract any extra features for this step.

4 Components of the Inverse Coarse-To-Fine Cascade

4.1 Sliding Window

Computing all possible boxes in a feature map of size $N \times N$ is in the order of $O(N^4)$ and therefore computationally unfeasible. Hence, similarly to Cheng et al. (2014), we select a set of window sizes that best cover the training data in terms of size and aspect ratio and use them in a sliding window fashion over the selected CNN layer. This approach is much faster than evaluating all possible windows and avoids to select windows with sizes or aspect ratios different from the training data and therefore probably false positives.

For the selection of the window sizes, we start with a pool of windows W in different sizes and aspect ratios $W : \{w | w \in \mathbb{Z}^2, \mathbb{Z} = [1 \dots 20]\}$. It is important to select a set of window sizes that gives high recall and at the same time produces well localized proposals. To this end, for each window size w, we compute its recall, R_α^w with different Intersection over Union (IoU) thresholds α and greedily pick one window size at a time. Specifically, we first compute recall of each window size w on multiple IoU of $\{0.5, 0.6, 0.7, 0.8, 0.9\}$ on 300 randomly selected images from the training set and then select the best N window sizes using the method described in Algorithm 1. Using this procedure, $N = 50$ window sizes are selected for the sliding window procedure. In Fig. 4 (middle) we show the maximum recall that can be obtained with the selected window sizes, which is an upper bound of the achievable recall of our method.

Algorithm 1 Window size selection

Input: R_α^w recall of window size w over all objects at threshold α, and N number of best selected window sizes (set to 50), and W all possible window sizes.
Output: S^* best selected window sizes.
initialize: $S^* = \{\}$
$w^* \leftarrow \text{argmax}_w \sum_\alpha R_\alpha^w$
$S^* = \{w^*\}$
$W \leftarrow W \setminus \{w^*\}$
for $i = 2 \cdots N$ **do**
  for each $w_j \in W$ **do**
    $\mathbf{L}^{w_j} = \sum_\alpha R_\alpha^{S^* \cup w_j}$
  end for
  $w^* \leftarrow \text{argmax}_w \mathbf{L}^{\mathbf{w}}$
  $S^* \leftarrow S^* \cup w^*$
  $W \leftarrow W \setminus \{w^*\}$
end for

4.2 Multiple Scales

Even though it is possible to cover all possible objects using a sliding window at a single scale of a feature map, it is inefficient since by using a single scale the stride is fixed and defined by the feature map resolution. For an efficient sliding window, the window stride should be proportional to the window size. Therefore, in all the experiments we evaluate our set of windows at multiple scales. For each scale, we resize the image such that $min(w, h) = s$ where $s \in \{227, 300, 400, 600\}$. Note that the first scale is the network original input size.

4.3 Pooling

As the approach should be very fast we represent a window by average pooling of the convolutional features that are inside the window. As averaging is a linear operation, after computing the integral image, the features of any proposal window can be extracted in a constant time. Let $f(x, y)$ be the specific channel of the feature map from a certain CNN layer and $F(x, y)$ its integral image. Then, average pooling Avg of a box defined by the top left corner $a = (a_x, a_y)$ and the bottom right corner $b = (b_x, b_y)$ is obtained as:

$$Avg(a, b) = \frac{F(b_x, b_y) - F(a_x, b_y) - F(b_x, a_y) + F(a_x, a_y)}{(b_x - a_x)(b_y - a_y)}. \quad (1)$$

Thus, after computing the integral image, the average pooling of any box is obtained in a constant time that cor-

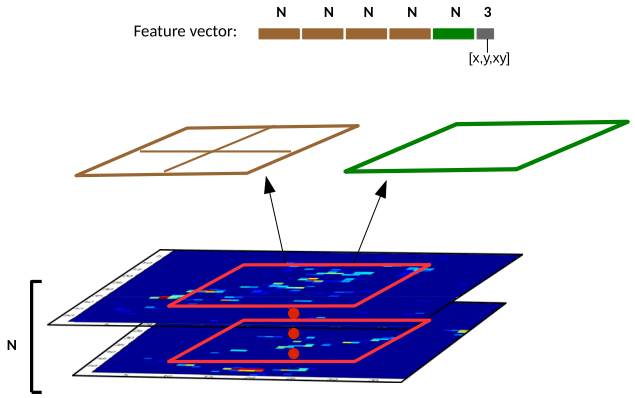

Fig. 2 A spatial pyramid representation used for the second stage of our method

responds to summing 4 integral values and dividing by the area of the box. We compute integral images for all feature maps of a particular layer, therefore the dimensionality of the feature vector of a window is equal to the number of channels (feature maps).

4.4 Pyramid

One of the main cues used to detect general objects is the object boundaries. Using an approach based on average pooling can dilute the importance of the object boundaries because it discards any geometrical information among features. Therefore, to introduce more geometry into the description of a window we consider a spatial pyramid representation (Lazebnik et al. 2006). An illustration is shown in Fig. 2. It consists of one full proposal window and a set of sub-windows. The sub-windows are generated by dividing the proposal window into a number of same size sub-windows (e.g. 2×2). Finally, each of them is represented and l_2 normalized separately.

4.5 Bias on Size and Aspect Ratio

Objects tend to appear at specific sizes and aspect ratios. Therefore, we add to the feature representation three additional elements: $(w, h, w \times h)$, where w and h are the width and height of a window. This can be considered as an explicit kernel which lets the SVM learn which object sizes can be covered at a specific scale. For the final descriptor, we normalize each pooled feature and size-related feature separately with the l_2 norm.

4.6 Classifier

We train linear classifiers shared between all window sizes but for each scale and for each layer separately. For a specific scale and layer, we randomly select at most 10 windows per object that overlap the annotation bounding boxes more than 70%, as positive training data and 50 windows per image that overlap less than 30% with ground-truth objects as negative data. In all experiments we use a linear SVM (Fan et al. 2008) because of its simplicity and fast training. We did not test non-linear classifiers since they would be too slow for our approach.

4.7 Non-maximum Suppression

The ranked window proposals at each scale are finally reduced through a non-maximum suppression step. A window is removed if its IoU with a higher scored window is more than a threshold α, which defines the trade-off between recall and accurate localization. So, this threshold is directly related to the IoU criteria that is used for evaluation (see Sect. 6.2). By tuning α, it is possible to maximize the recall at an arbitrary IoU of β. Particularly, in this work we define two variants of our DeepProposals, namely DeepProposals50 and DeepProposals70, that maximize recall at IoU of $\beta = 0.5$ and $\beta = 0.7$ respectively. To this end, we fix α to $\beta + 0.05$, as suggested in Zitnick and Dollár (2014). In addition, to aggregate boxes from different scales, we use another non-maximum suppression, fixing $\alpha = \beta$.

4.8 Refinement

The refinement is based on an edge-based scoring function introduced in Zitnick and Dollár (2014). The score is computed as difference between the contours wholly enclosed in a candidate bounding box and those that are not. A contour is wholly enclosed by a box if all edge pixels belonging to the contour lie within the interior of the box. Therefore, to compute the scoring function, we need to compute an edge response for each pixel in a given image. Edge responses are found by training 6 trees using a structured learning framework applied to random decision forests (Dollár and Zitnick 2013). As features, feature maps extracted from the fine layer of the CNN are fed to the structural random forest, considering each feature map as a channel for the algorithm. Given the edge map, the refinement is performed using a greedy iterative search to maximize the scoring function over position, scale and aspect ratio. At each iteration, the search step is reduced by half and the scores for new boxes are calculated. The search is halted once the translational step size is less than 2 pixels. Once the search is finished, the refined proposals are considered as our final proposals.

5 Proposals in Videos

Given a video sequence of length T, the goal is to generate a set of action proposals (tubes). Each proposal $P =$

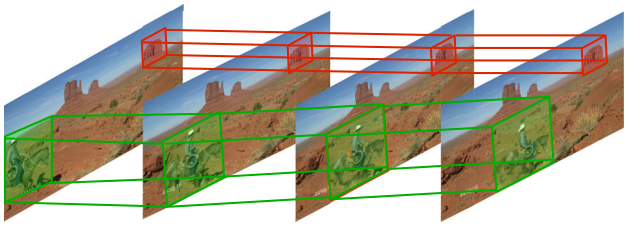

Fig. 3 Two sample action proposals start from the first frame and end in the last frame. The *green one* is a correctly recalled action proposal while the *red one* is a false positive (Color figure online)

$\{R_1, \ldots, R_t, \ldots, R_T\}$ corresponds to a path from the box R_1 in the first frame to the box R_T in the last frame, and it spatially localizes the action (see Fig. 3). When the goal is to find proposals in videos, we need (a) to capture the motion information that a video naturally provides, and (b) to satisfy time continuity constraints.

One advantage of DeepProposals is that it can be set up on top of any fine-to-coarse convolutional network regardless of its input/output (and possibly its architecture). To benefit from both appearance and motion cues in a given video, we use two networks for the task of action proposal generation. The first network takes as input the RGB frames of a video, and is based on an Alexnet-like architecture, fine-tuned on VOC2007 for the task of object detection. The second network takes as input the optical flow of each frame extracted from the video. We use the motion-CNN network of Gkioxari and Malik (2015) trained on UCF101 (split1) (Soomro et al. 2012). The architecture of this network is identical to that of the first network.

To generate a set of proposals in each frame, in the first and second stage of DeepProposals, we use an early fusion strategy, concatenating the feature maps generated by both networks and treating them as a single set of feature maps. For the last stage, since it is an alignment process, we only use the feature map of the appearance network.

So far the output is a set of proposals in each frame. In order to make the proposals temporally coherent, we follow the procedure of Gkioxari and Malik (2015) and link the proposals of each single frame over time into tubes. We define the linking scoring function between every two consecutive boxes R_t and R_{t+1} as follows:

$$S(R_t, R_{t+1}) = C(R_t) + C(R_{t+1}) + O(R_t, Rt + 1)$$

where $C(\cdot)$ is the confidence score for a box and $O(\cdot)$ is the intersection over union value if the overlap of the two boxes is more than 0.5, otherwise it is $-Inf$. Intuitively, the scoring function gives a high score if the two boxes R_t and R_{t+1} overlap significantly and if each of them most likely contains an object of an action.

Finally, we are interested in finding the optimal path over all frames. To this end, we first compute the overall score for each path P by $\sum_{t=1}^{T-1} \sum_{i,j \in P} S(R_t^i, R_{t+1}^j)$. Computing the score for all possible paths can be done efficiently using the Viterbi algorithm. The optimal path P^{opt} is then the one with the highest score. After finding the best path, all boxes in P^{opt} are removed and we solve the optimization again in order to find the second best path. This procedure is repeated until the last feasible path (those paths whose scores are higher than $-Inf$) is found. We consider each of these paths as an action proposal.

6 Experiments on Object Proposals

To evaluate our proposals, like previous works on object proposal generation, we focus on the well-known PASCAL VOC 2007 dataset. PASCAL VOC 2007 (Everingham et al. 2010) includes 9, 963 images divided in 20 object categories. 4952 Images are used for testing, while the remaining ones are used for training.

We first evaluate the performance of each component of our approach and its influence in terms of recall and localization accuracy on Alexnet Architecture. We then compare the quality of our DeepProposals with multiple state-of-the-art methods. Detection results and run-time are reported for PASCAL VOC 2007 (Everingham et al. 2010), integrating DeepProposals in the Fast-RCNN framework (Girshick 2015). Finally, we evaluate the generalization performance of DeepProposals on unseen categories and some qualitative comparisons are presented.

6.1 Evaluation Metrics

We use two different evaluation metrics; the first is Detection Rate (or Recall) versus Number of proposals. This measure indicates how many objects can be recalled for a certain number of proposals. We use Intersection over Union (`IoU`) as evaluation criterion for measuring the quality of an object proposal ω. `IoU` is defined as $|\frac{\omega \cap b}{\omega \cup b}|$ where b is the ground truth object bounding box. Initially, an object was considered correctly recalled if at least one generated window had an `IoU` of 0.5 with it, the same overlap used for evaluating the detection performance of a method. Unfortunately, this measure is too loose because a detector, to work properly, also needs a good alignment with the object (Hosang et al. 2015). Thus we evaluate our method for an overlap of 0.7 as well. We also evaluate recall versus overlap for a fixed number of proposals. As shown in Hosang et al. (2015), the average recall obtained from this curve seems highly correlated with the performance of an object detector built on top of these proposals.

6.2 Analysis of the Components

In this section, we investigate the effect of different parameters of our method, namely the different convolutional layers,

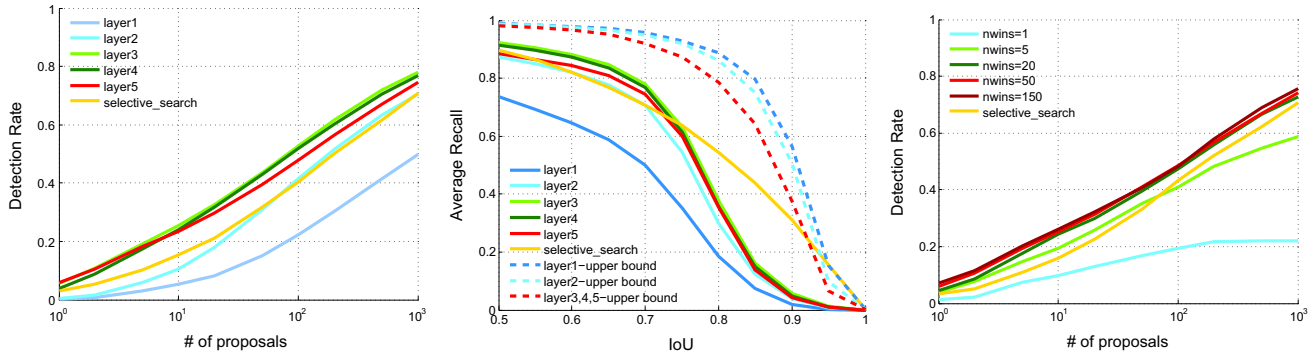

Fig. 4 (*Left*) Recall versus number of proposals for IoU=0.7. (*Middle*) Recall versus overlap for 1000 proposals for different layers of Alexnet. (*Right*) Recall versus number of proposals at IoU=0.7 on layer 5 for different number of window sizes. All are reported on the PASCAL VOC 2007 test set

the number of used window sizes and different levels of spatial pyramid pooling. We conduct this set of experiments without any cascading. Afterwards we investigate the effectiveness of different stages of DeepProposals.

6.3 Layers

We evaluate each convolutional layer (from 1 to 5) of Alexnet (Krizhevsky et al. 2012) using the sliding window settings explained above. For the sake of simplicity, we do not add spatial pyramids on top of pooled features in this set of experiments. As shown in Fig. 4 (left) the top convolutional layers of the CNN perform better than the bottom ones. Also their computational cost is lower as their representation is coarser. Note this simple approach already performs on par or even better than the best proposal generator approaches from the literature. For instance, our approach at layer 3 for 100 proposals achieves a recall of 52%, whereas selective search (Van de Sande et al. 2011) obtains only 40%. This makes sense because the CNN features are specific for object classification and therefore can easily localize the object of interest.

However, this is only one side of the coin. If we compare the performance of the CNN layers for high overlap [see Fig. 4 (middle)], we see that segmentation-based methods like Van de Sande et al. (2011) are much better. For instance the recall of selective search for 1000 proposals at 0.8 overlap is around 55% whereas ours at layer 3 is only 38%. This is due to the coarseness of the CNN feature maps that do not allow a precise bounding box alignment to the object. In contrast, lower levels of the net have a much finer resolution that can help to align better, but their encoding is not powerful enough to properly recall objects. In Fig. 4 (middle) we also show the maximum recall for different overlaps that a certain layer can attain with our selected sliding windows. In this case, the first layers of the net can recall many more objects with high overlap. This shows that a problem of the higher layers of the CNN is the lack of good spatial resolution.

In this sense we could try to change the structure of the net in a way that the top layers still have high spatial resolution. However, this would be computationally expensive and, more importantly, it would not allow us to reuse the same features used for detection. Instead, we use a cascade as an efficient way to leverage the expressiveness of the top layers of the net together with the better spatial resolution of the bottom layers.

6.4 Number of Window Sizes

In Fig. 4 (right) we present the effect of a varying number of window sizes in the sliding window procedure for proposal generation. The windows are selected based on the greedy algorithm explained in Sect. 4. As the number of used window sizes increases, we obtain a better recall at the price of a higher cost. In the following experiments we will fix the number of windows to 50 because that is a good trade-off between speed and top performance. The values in the figure refer to layer 5 of Alexnet, however, similar behavior has been observed for the other layers as well.

6.5 Spatial Pyramid

We evaluate the effect of using a spatial pyramid pooling in Fig. 5 (left). As expected, adding geometry improves the quality of the proposals. Moving from a pure average pooling representation (sp_level = 0) to a 2×2 pyramid (sp_level = 1) gives a gain that varies between 2 and 4 percent in terms of recall, depending on the number of proposals. Moving from the 2×2 pyramid to the 4×4 (sp_level = 2) gives a slightly lower gain. At 4×4 the gain does not saturate yet. However, as we aim at a fast approach, we also need to consider the computational cost, which is linear in the number of spatial bins used. Thus, the representation of a window with a 2×2 spatial pyramid is 5 times slower than a flat representation and the 4×4 pyramid is 21 times slower. For this reason, in

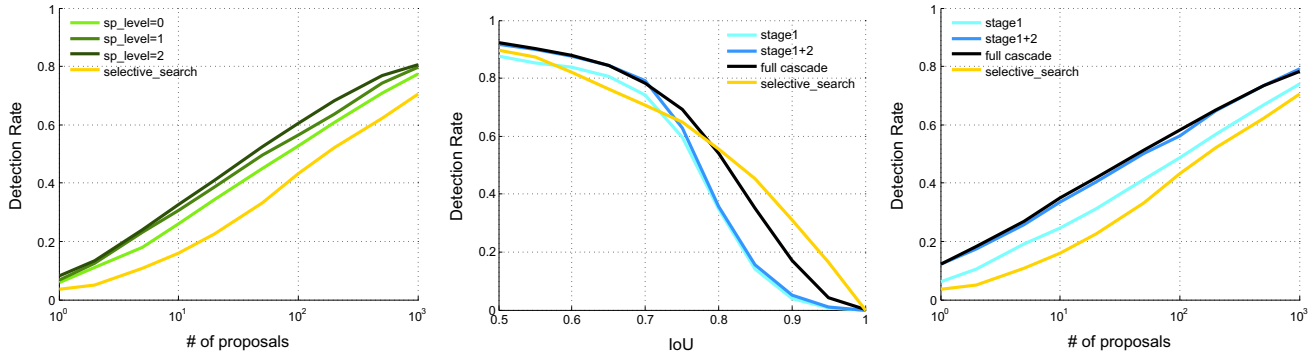

Fig. 5 (*Left*) Recall versus number of proposals in IoU=0.7 for different spatial pyramid levels (*Middle*) Recall versus IoU for 1000 proposals for different stages of the cascade. (*Right*) Recall versus number of proposals in IoU=0.7 for the different stages of the cascade. All are reported on the PASCAL VOC 2007 test set

Table 1 Characteristics of the stages of our inverse cascade applied to AlexNet (*NMS* non maximum suppression)

Stage	Layer	Input candidates	Method	Pyramid	NMS	Total time per image (s)
1	5	∼80.000	Slid. window	1	Yes	0.30
2	3	4.000	Re-scoring	$1+2\times 2$	Yes	0.25
3	2	1.000	Refinement	–	No	0.20

our final representation we limit the use of the spatial pyramid to a 2×2 spatial pyramid.

6.6 Stages

We now discuss the performance of the inverse cascade stage by stage in terms of both computational cost and performance. For Alexnet architecture we use layer numbers 14, 10, 6 for stage one to three respectively. A summary of the computational cost of each stage is given in Table 1. The entire cascade has a computational cost of 0.75 on a 8-core CPU of 3.50 GHz, which is the composition of 0.3 , 0.25 and 0.2 for the first, second and third stage, respectively. Note that the first stage is very fast because even if we use a dense sliding window approach, with the integral image and without any pyramid level the cost of evaluating each window is very low.

As shown in Fig. 5 (middle and right), the second stage is complementary to the first and employed with a 2×2 pyramid improves the recall of the cascade by 5%. However, this boost is valid only up to an overlap of 0.75. After this point the contribution of the second stage is negligible. This is due to the coarse resolution of layer 5 and 3 that do not allow for a precise overlap of the candidate windows with the ground truth object bounding boxes. We found that, for our task, layer 3 and 4 have a very similar performance (Recall@1000 is 79% in both cases) and adding the latter in the pipeline did not help in improving performance (Recall@1000 is still 79%).

As shown in Hosang et al. (2015), for a good detection performance, not only the recall is important, but also a good alignment of the candidates is needed. At stage 3 we improve the alignment without performing any further selection of windows; instead we refine the proposals generated by the previous stages by aligning them to the edges of the object. In our experiments for contour detection we observed that the first layer of the CNN did not provide as good a performance as layer 2 [0.61 vs. 0.72 AP on BSDS dataset (Arbelaez et al. 2011)], so we choose the second layer of the network for this task. Figure 5 (middle) shows that this indeed improves the recall for high IoU values (above 0.7) (Table 2).

6.7 Network Architecture

So far, we evaluated DeepProposals on a pre-trained Alexnet architecture trained on the Imagenet dataset. However, one advantage of DeepProposals is that it can be implemented on networks trained on different datasets or networks with different architectures. To this end, we setup our method on two other networks. The first is an alexnet-like architecture trained on the Places dataset (Zhou et al. 2014). We used the pre-trained network of Zhou et al. (2014) called placeNet for this purpose and use exactly the same layer numbers as our original DeepProposals (i.e. layer numbers 14, 10, 6). The second architecture is the VGG-16 (Simonyan and Zisserman 2015) network trained on Imagenet. This network is deeper compared to Alexnet and we use layers number 30, 24 and 12 for stage one to three, respectively. We did not change any other hyper-parameters of the method. Figure 6 shows the

Table 2 Characteristics and performance of the CNN layers

Layer	Feature map size	Recall(#1000,0.5) (%)	Max(0.5) (%)	Recall(#1000,0.8) (%)	Max(0.8) (%)
5	$36 \times 52 \times 256$	88	97	36	70
4	$36 \times 52 \times 256$	91	97	36	79
3	$36 \times 52 \times 256$	92	97	38	79
2	$73 \times 105 \times 396$	87	98	29	86
1	$146 \times 210 \times 96$	73	99	18	89

Feature map size is reported for an image of size 600×860. Recall (#1000,β) is the recall of 1000 proposals for the overlap threshold β. Max(β) is the maximum recall for the overlap threshold β using our selected window sizes set

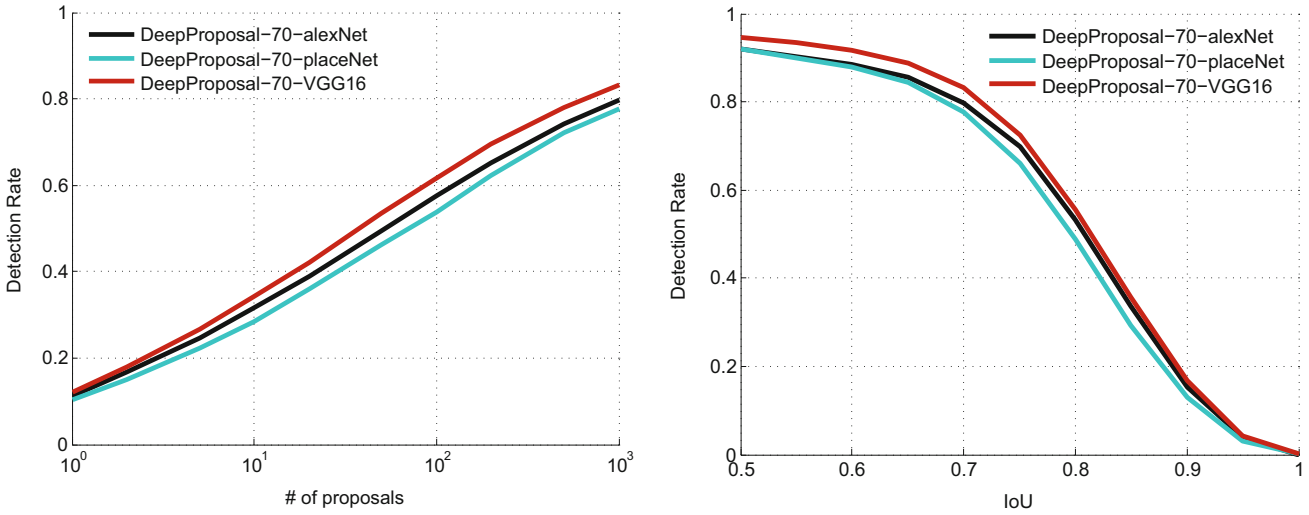

Fig. 6 Recall versus number of proposals on the Pascal 2007 evaluation set for different network architectures with (*left*) IoU threshold of 0.7 and (*right*) recall versus IoU threshold for 1000 proposal windows

performance of DeepProposals on different networks. Surprisingly, placeNet works well particularly considering that it is trained to recognize scenes while we are using its feature maps for discovering objects. Its slightly lower performance compared to original DeepProposals can also be explained due to this fact. When the VGG-16 network is used we can observe a clear improvement in the quality of the generated proposals. This shows that with a more powerful network also the quality of the proposals is improved.

6.8 Comparison with State-of-the-Art

In this section we compare the quality of the proposed DeepProposals with state-of-the-art object proposals on PASCAL 2007 (Everingham et al. 2010) and Microsoft COCO 2014 (Lin et al. 2014a) dataset.

Figures 7 and 9 show the recall with a varying number of object proposals or IoU threshold, respectively, for the PASCAL dataset. From Fig. 7, we can see that, even with a small number of windows, DeepProposals can achieve a higher recall for any IoU threshold. Methods like BING (Cheng et al. 2014) and objectness Alexe et al. (2010) provide a high recall only at IoU=0.5, because they are tuned for IoU of 0.5.

In Table 3 we summarize the quality of the proposals generated by the most promising methods. Achieving 75% recall with IoU of 0.7 would be possible with 540 proposals of DeepProposals, 800 of Edge boxes, 922 of RPN-ZF, 1400 of selective search proposals and 3000 of randomized Prim's proposals (Manen et al. 2013) on the PASCAL dataset (Fig. 8).

Figure 9 (left) and (middle) show the curves related to recall over IoU with 100 and 1000 proposals for PASCAL dataset. Again, DeepProposals obtain good results. The hand crafted segmentation based methods like selective search and MCG have a good recall rate at higher IoU values. Instead DeepProposals perform better in the range of IoU=[0.6, 0.8] which is desirable in practice and playing an important role in the object detectors performance (Hosang et al. 2015).

Figure 9 (right) shows average recall (AR) versus number of proposals for different methods for the PASCAL dataset. For a specific number of proposals, AR measures the proposal quality across IoU of [0.5, 1]. Hosang et al. (2015) show that AR correlates well with detection performance.

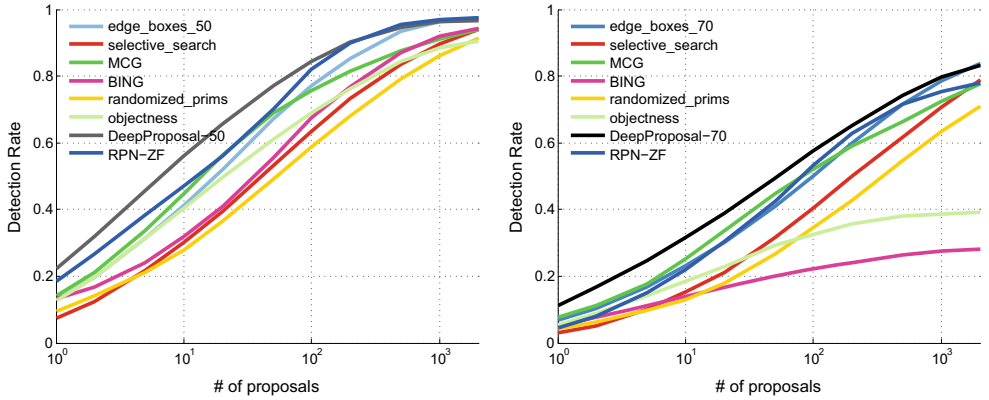

Fig. 7 Recall versus number of proposals on the PASCAL VOC 2007 test set for (*left*) IoU threshold 0.5 and (*right*) IoU threshold 0.7

Table 3 Our method compared to other methods for IoU threshold of 0.7

	AUC	N@25%	N@50%	N@75%	Recall (%)	Time (s)
BING (Cheng et al. 2014)	.19	292	–	–	29	.2
Objectness (Alexe et al. 2010)	.26	27	–	–	39	3
Rand. Prim's (Manen et al. 2013)	.30	42	349	3023	71	1
Selective search (Van de Sande et al. 2011)	.34	28	199	1434	79	10
Edge boxes 70 (Zitnick and Dollár 2014)	.42	12	108	800	**84**	.3
MCG (Arbelaez et al. 2014)	.42	9	81	1363	78	30
RPN-ZF (Ren et al. 2015)	.42	13	83	922	78	**.1**[a]
DeepProposals70	**.48**	**5**	**53**	**540**	83	.75

Bold values indicate the best results
AUC is the area under recall versus IoU curve for 1000 proposals. N@25to achieve a recall of 25, 50 and 75%, respectively. For reporting recall, at most 2000 boxes are used. The run-times for the other methods are obtained from Hosang et al. (2015)
[a] In contrast to the other methods, for RPM-ZF the run-time is evaluated on a GPU

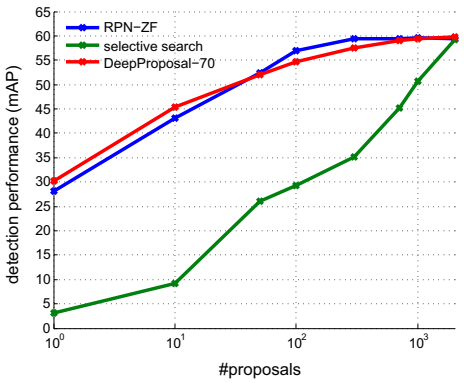

Fig. 8 Detection results on PASCAL VOC 2007

Using this criteria, DeepProposals are on par or better than other methods with 700 or fewer boxes but with more boxes, selective search and Edgeboxes perform better.

We also evaluate DeepProposals on the COCO 2014 evaluation set. We train our SVM classifiers on 5000 randomly sampled images from the COCO training set and evaluate our method on the evaluation set. For this dataset, the areas less than 32×32 are considered as difficult and are ignored during evaluation. The same concept is also available for PASCAL but the definition of difficult objects is different than that of COCO. However, our method is not designed to localize small objects since the window sizes and scales we chose are tuned for PASCAL. In Fig. 10 we show the recall with a varying number of object proposals. Figure 11 left and middle show the curves related to recall over IoU with 100 and 1000 proposals, respectively, and Fig. 11 (right) shows the average recall (AR) versus number of proposals. In general, the trend is the same as with the PASCAL dataset, except that MCG performs better in some cases.

6.9 Run-Time

The run-time tests for our proposed method and the others are also available in Table 3. Since our approach uses the same CNN features used by state-of-the-art object detectors like RCNN (Girshick et al. 2014) and SppNet (He et al. 2015), it does not need any extra cues and features and we can consider just the running time of our algorithm without the CNN

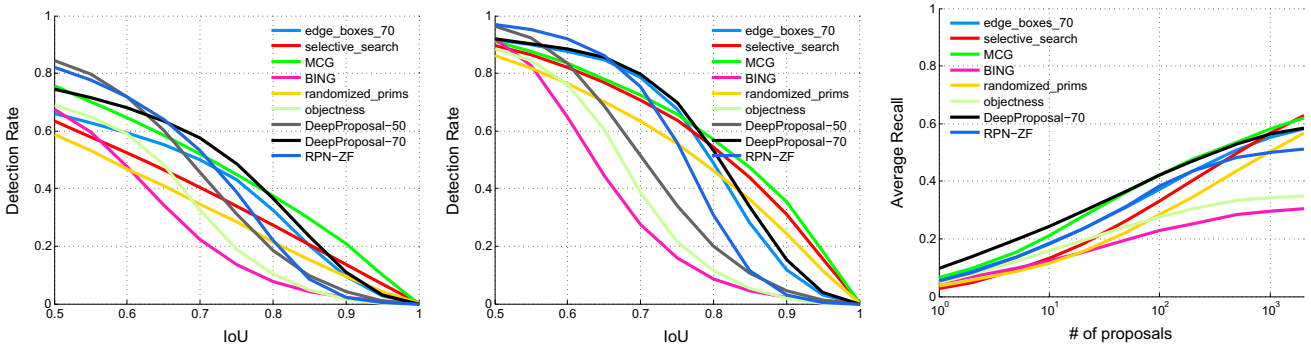

Fig. 9 Recall versus IoU threshold on the PASCAL VOC 2007 test set for (*left*) 100 proposal windows and (*middle*) 1000 proposal windows. (*Right*) average recall between [0.5,1] `IoU` on the PASCAL VOC 2007 test set

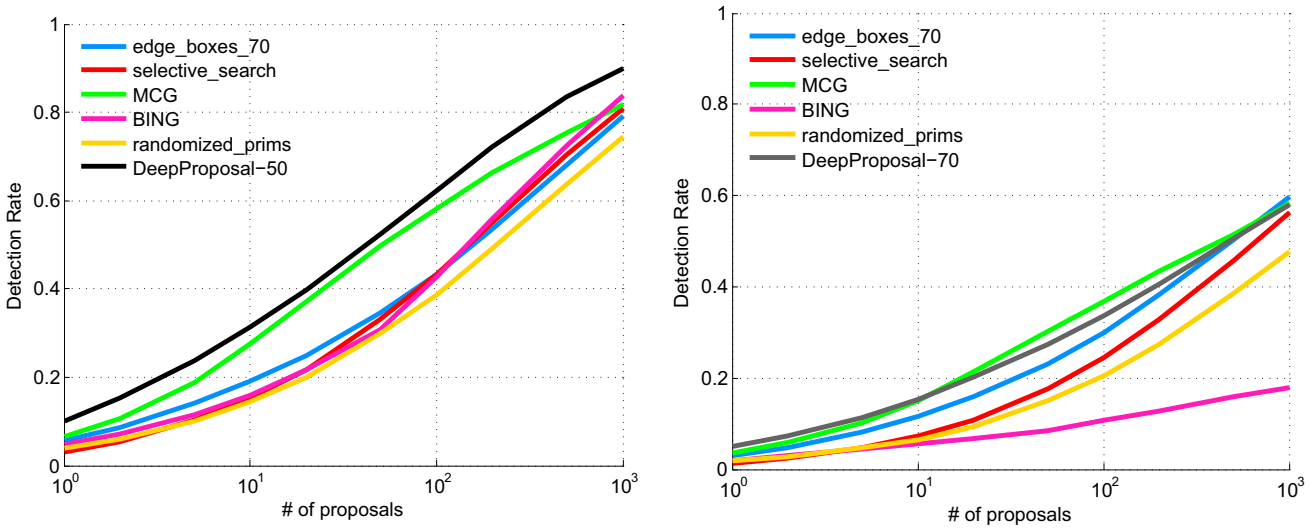

Fig. 10 Recall versus number of proposals on the COCO 2014 evaluation set for (*left*) IoU threshold 0.5 and (*right*) IoU threshold 0.7

extraction time.[1] DeepProposals takes 0.75 s on CPU and 0.4 s to generate object proposals on a GeForce GTX 750 Ti GPU, which is slightly slower than Edgeboxes. The fastest method is RPN-ZF, a convolutional network based on Zeiler and Fergus (2014) network architecture, tuned for generating object proposals. Note that for RPN-ZF, the running-time on a GPU is reported while the others are reported on a CPU. The remaining methods are segmentation based and take considerably more time.

6.10 Qualitative Results

Figure 12 shows some qualitative results of DeepProposals and another state of the art method, Edge boxes. In general, when the image contains high-level concepts cluttered with many edges (e.g. Fig. 12 rows 1, first column) our method gives better results. However, for small objects with clear boundaries edge boxes performs better since it is completely based on contours and can easily detect smaller objects.

6.11 Object Detection Performance

In the previous experiments we evaluated our proposal generator with different metrics and showed that it is among the best methods for all of them. However, we believe that the best way to evaluate the usefulness of the generated proposals is a direct evaluation of the detector performance. Indeed, recently it has become clear (see Hosang et al. (2015)) that an object proposal method with high recall at 0.5 IoU does not automatically lead to a good detector.

Some state-of-the-art detectors at the moment are: RCNN (Girshick et al. 2014), SppNet (He et al. 2015), fast-RCNN (Girshick 2015). All are based on CNN features and use object proposals to localize the object of interest. The first uses the window proposals to crop the corresponding regions of the image, compute the CNN features and obtain

[1] If CNN features have to be (re)computed, that would add 0.15 s. extra computation time on our GPU.

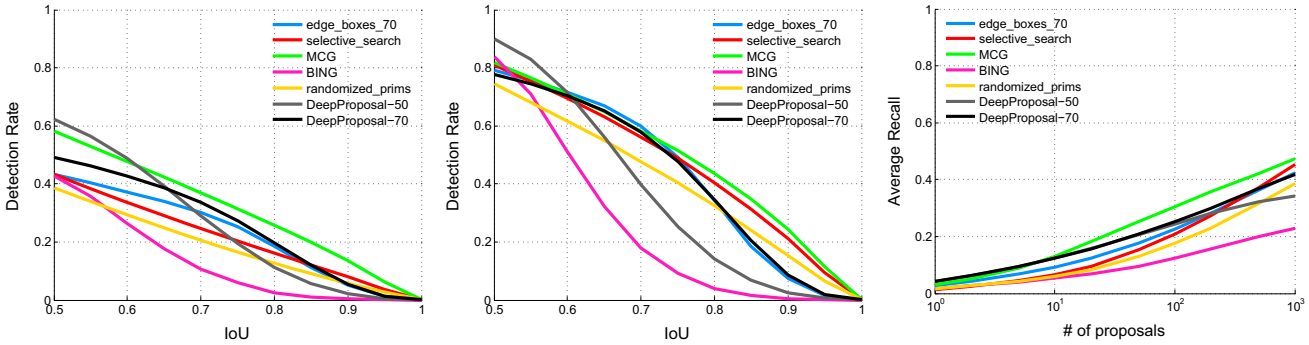

Fig. 11 Recall versus IoU threshold on the COCO 2014 evaluation set for (*left*) 100 proposal windows and (*middle*) 1000 proposal windows. (*Right*) average recall between [0.5,1] IoU on the COCO 2014 evaluation set

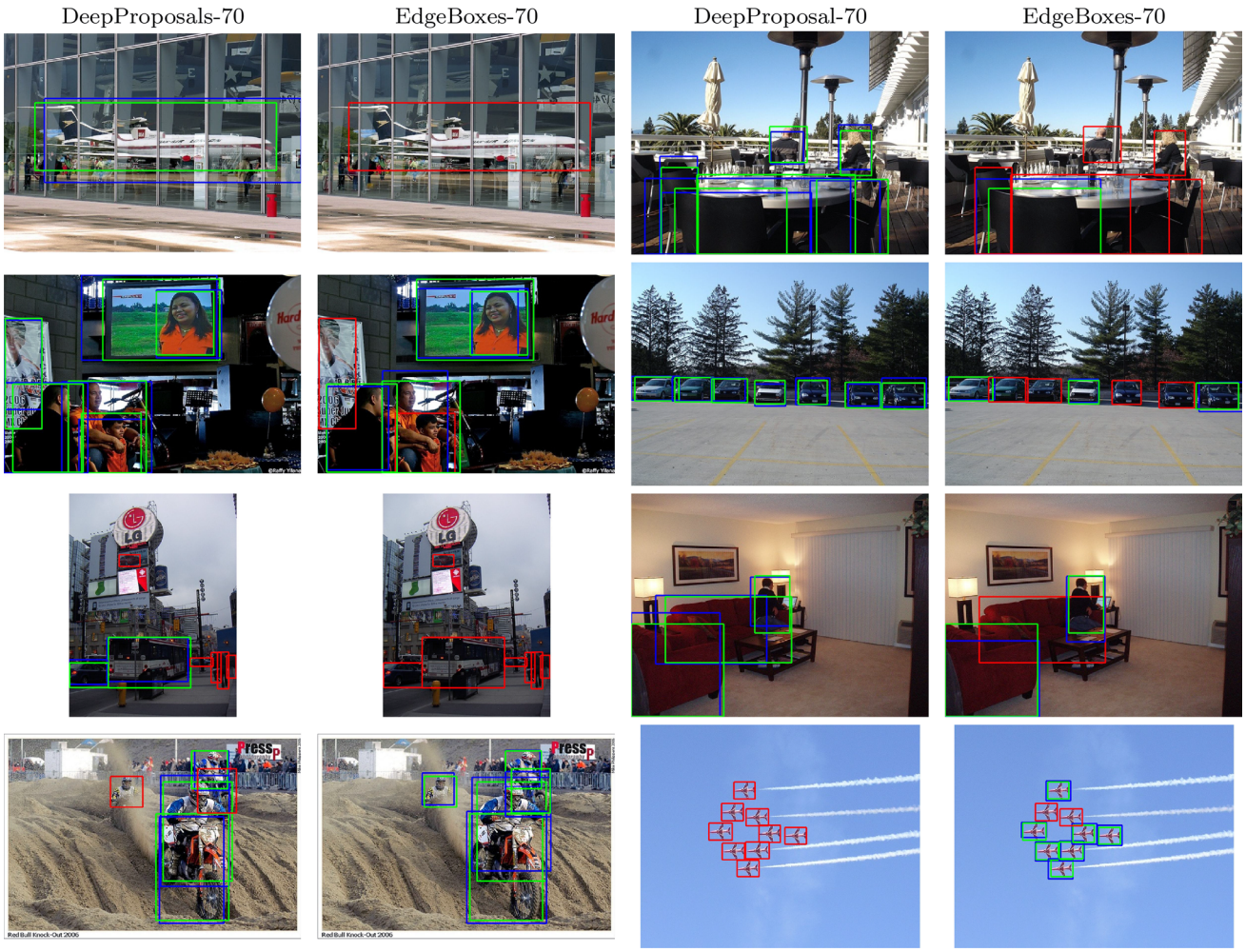

Fig. 12 Qualitative examples of our object proposals (*1st and 3rd column*) versus Edge boxes proposals (*2nd and 4th column*). For the first three rows our method performs better, whereas in the last row edge boxes is better. An object is correctly localized if its IoU with the ground-truth bounding box is more than 0.7. We use **1000** proposals for each method. *Blue boxes* are the closest proposals to each ground truth bounding box. *Red* and *green boxes* are ground-truth boxes where *green* indicates a localized object while *red* indicates a missed object (Color figure online)

a classification score for each region. This approach is slow and takes around 10 s on a high-end GPU and more than 50 s on the GPU used for our experiments (GeForce GTX 750 Ti). SppNet and fast-RCNN instead compute the CNN features only once, on the entire image. Then, the proposals are used to select the sub-regions of the feature maps from

where to pull the features. This allows this approach to be much faster. With these approaches then, we can also reuse the CNN features needed for the generation of the proposals so that the complete detection pipeline can be executed without any pre-computed component, roughly in 1 second on our GPU (GeForce GTX 750 Ti).

Concurrently to our method also the Faster-RCNN was recently introduced (Ren et al. 2015). It uses a Region Proposal Network (RPN) for generating proposals that shares full-image convolutional features with the detection network.

We compare the detection performance of our DeepProposals70 with selective search and RPN proposals. For RPN proposals the detector is trained as in the original paper (Ren et al. 2015) with an alternating procedure, where detector and localization sub-network update the shared parameters alternatively. Our method and selective search are instead evaluated using a detector fine-tuned with the corresponding proposals, but without any alternating procedure, i.e. the boxes remain the same for the entire training. The training is conducted using the faster-RCNN code on PASCAL VOC 2007 with 2000 proposals per image. In Fig. 8 we report the detector mean average precision on the PASCAL VOC 2007 test data for different number of used proposals.

The difference of selective search with CNN-based approaches is quite significant and it appears mostly in a regime with low number of proposals. For instance, when using 50 proposals selective search obtains a mean average precision (mAP) of 28.1, while RPN and our method obtain a mAP already superior to 50. We believe that this difference in behavior is due to the fact that our method and RPN are supervised to select good object candidates, whereas selective search is not.

Comparing our proposals with RPN, we observe a similar trend. DeepProposals produces superior results with a reduced amount of proposals (<100), while RPN performs better in the range of between 100 and 700 proposals. With more than 700 proposals both methods perform again similarly and better than selective search. Finally, with 2000 proposals per image, selective search, RPN and DeepProposals reach the detection performance of 59.3, 59.4 and 59.8, respectively.

Thus, from these results we can see that RPN and our approach perform very similarly. The main difference between the two approaches lies in the way they are trained. Our approach assumes an already pre-trained network, and learns to localize the object of interest by leveraging the convolutional activations generated for detection. RPN instead needs to be trained together with the detector with an alternating approach. In this sense, our approach is more flexible because it can be applied to any CNN based detector without modifying its training procedure.

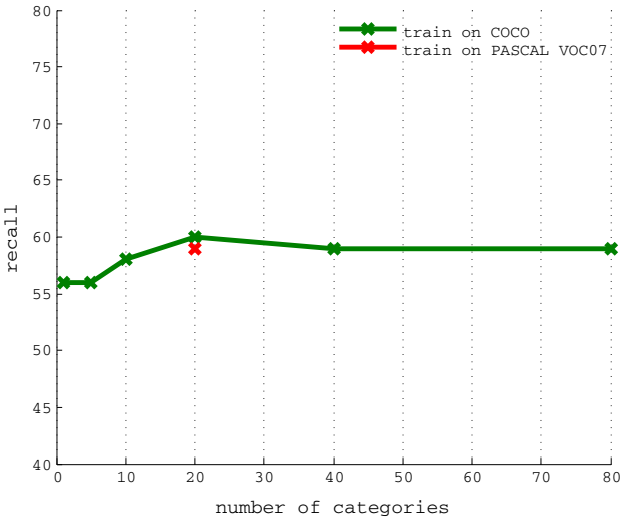

Fig. 13 Generalization of DeepProposals: we train models with different numbers of categories and evaluate them on the whole eval-set of the COCO dataset. We set the IoU threshold to 0.5 and the number of proposals to 1000

6.12 Generalization to Unseen Categories

We evaluate the generalization capability of our approach on the Microsoft COCO dataset (Lin et al. 2014b). The evaluation of the approach has been done by learning either from the 20 classes from VOC07 or COCO or from 1, 5, 20, 40, or 80 categories randomly sampled from COCO. As shown in Fig. 13, when the DeepProposals are trained using only 5 classes, the recall at 0.5 IoU with 1000 proposals is slightly reduced (56%). With more classes, either using VOC07 or COCO, recall remains stable around 59–60%. This shows that the method can generalize well over all classes. We believe this is due to the simplicity of the classifier (average pooling on CNN features) that avoids over-fitting to specific classes. Note that in this case our recall is slightly lower than the Selective Search with 1000 proposals (63%). This is probably due to the presence of very small objects in the COCO dataset, that are missed by our method as it was not tuned for this setting. These results on COCO demonstrate that our proposed method is capable to generalize learnt objectness beyond the training categories.

7 Experiments on Action Proposals

7.1 Evaluation

We evaluate the performance of the inverse cascade for action proposals on the UCF-Sports (Rodriguez et al. 2008) dataset. We train our models on a training set that contains a total of 6604 frames. Additionally, we have 2976 frames in the test set, spread over 47 videos.

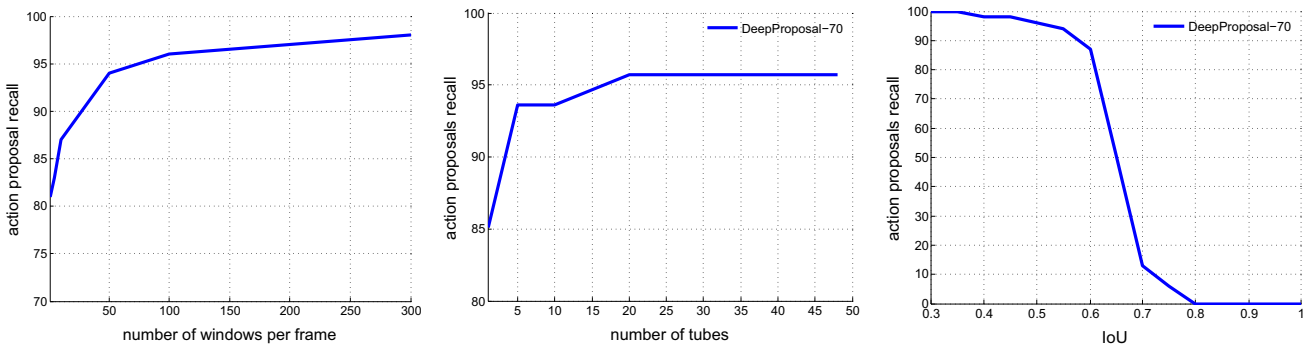

Fig. 14 (*Left*) action proposals recall at IoU of 0.5 varying the number of windows per frame. (*Middle*) recall versus number of proposals in IoU=0.5 for the different number of action proposals. (*Right*) action proposal recall at different IoU values for 20 action proposals. All results are reported on the UCF-sports test set

Like van Gemert et al. (2015), we measure the overlap between an action proposal and the ground-truth video tubes using the average intersection-over-union score of 2D boxes for all frames where there is either a ground-truth box or proposal box. Formally:

$$Ovr(P, G) = \frac{1}{|F|} \sum_{t \in F} \frac{P_t \cap G_t}{P_t \cup G_t},$$

where P and G are action proposal and ground-truth action tube, respectively. F is the set of frames where either P or G is not empty. G_t is empty if there is no action in the frame t of the ground-truth tube. In this case, $P_t \cap G_t$ is set to 0 for that frame.

Considering each frame as an image and applying DeepProposals on each frame individually, the frame-based recall of objects/actors for an IoU of 0.7 is 78% for 10 windows and 96% for 100 windows. One possible explanation for such a promising frame-based recall is that an action mainly contains an actor performing it and hunting that actor in each frame is relatively easier than hunting general objects in the task of object proposal generation. However, this does not take into account any temporal continuity. Constructing tubes from these static windows, which results in our action proposals, is our final goal.

The extension of the inverse cascade for actions introduces an additional parameter which is the number of windows that we select in each frame. In Fig. 14 (left) we show the recall of the action proposals while varying the number of windows we select per frame. As expected, selecting more windows in each frame leads to a higher recall of the action proposals. However, it also leads to an increasing computational cost, since the computational complexity of the Viterbi algorithm is proportional to the square of the number of windows per frame. For example, the Viterbi algorithm for a video of length 240 frames takes 1.3 and 12.1 s for $N = 100$ and $N = 300$ respectively. From now on, during all the following experiments we select $N = 100$ windows per frame to have a good balance between performance and time complexity.

Table 4 Our action proposals generator compared to other methods at a IoU threshold of 0.5

	Recall	# Proposals
UCF-Sports		
Brox and Malik (2010)	17.02	**4**
Jain et al. (2014)	78.72	1642
On average Oneata et al. (2014)	68.09	3000
Gkioxari and Malik (2015)	87.23	100
APT (van Gemert et al. 2015)	89.36	1449
DeepProposals	**95.7**	20
UCF101		
APT (van Gemert et al. 2015)	37.0	2304
DeepProposals	**38.6**	**34**

Bold values indicate best results
The number of proposals is averaged over all test videos. All the reported numbers on UCF-sports except ours are obtained from van Gemert et al. (2015). For UCF101, like ours, we report the APT performance for split3

Figure 14 (middle) shows the action proposals recall for different number of proposals. As it is shown even for a very small number of proposals, DeepProposals obtains very good performance as already observed also for object proposals. Figure 14 (right) shows the recall of our method for 20 action proposals (tubes) per video over different IoU values. Our method works very well in the regime of [0.3 . . . 0.6]. Notice that the definition of action proposals recall is different than object proposals recall and the performance in IoU=0.5 is already quite promising.

7.2 Comparison with the State of the Art

We evaluate our action proposals on two different datasets namely UCF-Sports (Rodriguez et al. 2008) and UCF101

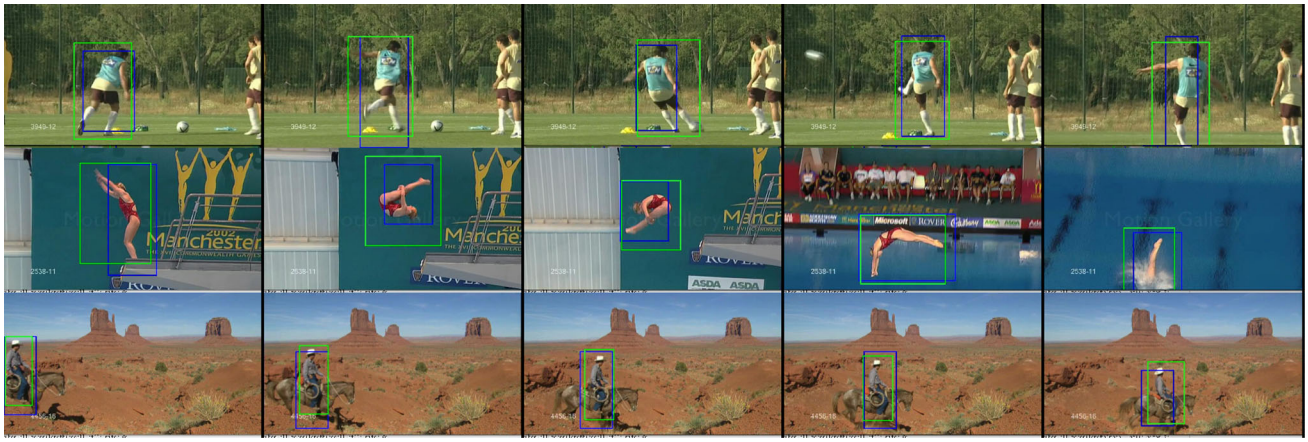

Fig. 15 Qualitative examples of our action proposals. In each row, we select 5 frames of a video for visualization purposes. *Blue boxes* are the closest proposals to each ground truth box (shown in *green*) (Color figure online)

(Soomro et al. 2012). UCF-Sports contains 10 action categories and consists of 150 video samples, extracted from sport broadcasts. The actions in this dataset are temporally trimmed. For this dataset we use the train and test split proposed in Lan et al. (2011). UCF101 is collected from YouTube and has 101 action categories where 24 of the annotated classes (corresponding to 3204 videos) are used in literature for action localization. In this dataset, for evaluation we report the average recall of 3 splits. Finally, for both datasets, we select the first top 100 boxes in each frame and find the N best paths over time for each video.

In Table 4 we compare our proposal generation method against state-of-the-art methods in the presented datasets. As shown, our method is competitive or improves over all other methods with fewer proposals. In the UCF-Sports dataset, DeepProposals have higher recall compared to the recently published APT proposal generator (van Gemert et al. 2015) with almost 70x fewer proposals. Notice that the method proposed by Brox and Malik (2010) is designed for motion segmentation and we use it here to emphasize the difficulty of generating good video proposals. In the UCF101 dataset we see the same trend, we outperform APT while using 67× fewer proposals.

7.3 Run-Time

Computationally, given the optical flow images, our method needs 1.2 s per frame to generate object proposals and on average 1.3 s for linking all the windows. Most of the other methods are an order of magnitude more expensive mainly because of performing super-pixel segmentation and grouping.

7.4 Qualitative Results

In Fig. 15 we provide examples of our action proposals extracted from some videos of UCF-sports dataset. For each video we show 5 cross sections of a tube. These sections are equally distributed in the video.

8 Conclusion

DeepProposalsis a new method to produce fast proposals for object detection and action localization. In this paper, we have presented how DeepProposals produces proposals at a low computational cost through the use of an efficient coarse-to-fine cascade on multiple layers of a detection network, reusing the features already computed for detection. We have evaluated the method on most recent benchmarks and against previous approaches, and have shown that in most cases it is comparable to or better than state-of-the-art approaches in terms of both accuracy and computation. The source code of DeepProposals is available online.[2]

Acknowledgements This work was supported by a DBOF PhD scholarship, the KU Leuven CAMETRON Project and the FWO Project "Monitoring of Abnormal Activity with Camera Systems".

References

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

Arbelaez, P., Maire, M., Fowlkes, C., & Malik, J. (2011). Contour detection and hierarchical image segmentation. *PAMI*, *33*(5), 898–916.

[2] https://github.com/aghodrati/deepproposal.

Arbelaez, P., Pont-Tuset, J., Barron, J., Marques, F., & Malik, J. (2014). Multiscale combinatorial grouping. In *CVPR*

Bergh, M., Roig, G., Boix, X., Manen, S., & Gool, L. (2013). Online video seeds for temporal window objectness. In *ICCV*

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

Brox, T., & Malik, J. (2010). Object segmentation by long term analysis of point trajectories. In *ECCV*

Carreira, J., & Sminchisescu, C. (2012). Cpmc: Automatic object segmentation using constrained parametric min-cuts. *PAMI*, *34*(7), 1312–1328.

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

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

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

Dollár, P., & Zitnick, C. L. (2013). Structured forests for fast edge detection. In *ICCV*

Erhan, D., Szegedy, C., Toshev, A., & Anguelov, D. (2014). Scalable object detection using deep neural networks. In *CVPR*

Everingham, M., Van Gool, L., Williams, C. K., Winn, J., & Zisserman, A. (2010). The pascal visual object classes (voc) challenge. *IJCV*, *88*(2), 303–338.

Fan, R. E., Chang, K. W., Hsieh, C. J., Wang, X. R., & Lin, C. J. (2008). Liblinear: A library for large linear classification. *The Journal of Machine Learning Research*, *9*, 1871–1874.

Ghodrati, A., Diba, A., Pedersoli, M., Tuytelaars, T., & Van Gool, L. (2015). Deepproposal: Hunting objects by cascading deep convolutional layers. In *ICCV*

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

Girshick, R., Donahue, J., Darrell, T., & Malik, J. (2014). Rich feature hierarchies for accurate object detection and semantic segmentation. In *CVPR*

Gkioxari, G., & Malik, J. (2015). Finding action tubes. In *CVPR*

Hariharan, B., Arbelez, P., Girshick, R., & Malik, J. (2014). Hypercolumns for object segmentation and fine-grained localization. arXiv preprint arXiv:1411:5752

He, K., Zhang, X., Ren, S., & Sun, J. (2015). Spatial pyramid pooling in deep convolutional networks for visual recognition. *PAMI*, *37*(9), 1904–1916.

Hosang, J., Benenson, R., Dollár, P., & Schiele, B. (2015). What makes for effective detection proposals? *PAMI*, *38*(4), 814–830.

Jain, M., Gemert, J., J.égou, H., Bouthemy, P., & Snoek, C. (2014). Action localization with tubelets from motion. In *CVPR*

Krähenbühl, P., & Koltun, V. (2014). Geodesic object proposals. In *ECCV*

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

Lan, T., Wang, Y., & Mori, G. (2011). Discriminative figure-centric models for joint action localization and recognition. In *ICCV*

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

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

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

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

Oneata, D., Revaud, J., Verbeek, J., & Schmid, C. (2014). Spatio-temporal object detection proposals. In *ECCV*

Pinheiro, P. O., Collobert, R., & Dollar, P. (2015). Learning to segment object candidates. In *NIPS*

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

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

Rodriguez, M. D., Ahmed, J., & Shah, M. (2008). Action mach a spatio-temporal maximum average correlation height filter for action recognition. In *CVPR*

Sermanet, P., Eigen, D., Zhang, X., Mathieu, M., Fergus, R., & LeCun, Y. (2013). Overfeat: Integrated recognition, localization and detection using convolutional networks. arXiv preprint arXiv:1312.6229

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

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

Soomro, K., Zamir, A. R., & Shah, M. (2012). Ucf101: A dataset of 101 human actions classes from videos in the wild. arXiv preprint arXiv:1212.0402

Tran, D., Yuan, J., & Forsyth, D. (2014). Video event detection: From subvolume localization to spatiotemporal path search. *PAMI*, *36*(2), 404–416.

van Gemert, J. C., Jain, M., Gati, E., & Snoek, C. G. (2015). Apt: Action localization proposals from dense trajectories. In *BMVC*

Van de Sande, K. E., Uijlings, J. R., Gevers, T., & Smeulders, A. W. (2011). Segmentation as selective search for object recognition. In *ICCV*

Viola, P., & Jones, M. J. (2004). *Robust real-time face detection. IJCV*, *57*(2), 137–154.

Wang, X., Yang, M., Zhu, S., & Lin, Y. (2013). Regionlets for generic object detection. In *ICCV*

Weinzaepfel, P., Harchaoui, Z., & Schmid, C. (2015). Learning to track for spatio-temporal action localization. In *ICCV*

Xinggang, W., Yan, W., Xiang, B., & Zhijiang, Z. (2015). Deepcontour:, A. deep convolutional feature learned by positive-sharing loss for contour detection. In *CVPR*

Yu, G., & Yuan, J. (2015). Fast action proposals for human action detection and search. In *CVPR*

Zeiler, M. D., & Fergus, R. (2014). Visualizing and understanding convolutional networks. In *ECCV*

Zhao, Q., Liu, Z., & Yin, B. (2014). Cracking bing and beyond. In *BMVC*

Zhou, B., Lapedriza, A., Xiao, J., Torralba, A., & Oliva, A. (2014). Learning deep features for scene recognition using places database. In *NIPS*

Zitnick, C. L., & Dollár, P. (2014). Edge boxes: Locating object proposals from edges. In *ECCV*