Fast object segmentation in unconstrained video

Anestis Papazoglou
University of Edinburgh

Vittorio Ferrari
University of Edinburgh

Abstract

We present a technique for separating foreground objects from the background in a video. Our method is fast, fully automatic, and makes minimal assumptions about the video. This enables handling essentially unconstrained settings, including rapidly moving background, arbitrary object motion and appearance, and non-rigid deformations and articulations. In experiments on two datasets containing over 1400 *video shots, our method outperforms a state-of-the-art background subtraction technique [4] as well as methods based on clustering point tracks [6, 18, 19]. Moreover, it performs comparably to recent video object segmentation methods based on object proposals [14, 16, 27], while being orders of magnitude faster.*

1. Introduction

Video object segmentation is the task of separating foreground objects from the background in a video [14, 18, 26]. This is important for a wide range of applications, including providing spatial support for learning object class models [19], video summarization, and action recognition [5].

The task has been addressed by methods requiring a user to annotate the object position in some frames [3, 20, 26, 24], and by fully automatic methods [14, 6, 18, 4], which input just the video. The latter scenario is more practically relevant, as a good solution would enable processing large amounts of video without human intervention. However, this task is very challenging, as the method is given no knowledge about the object appearance, scale or position. Moreover, the general unconstrained setting might include rapidly moving backgrounds and objects, non-rigid deformations and articulations (fig. 5).

In this paper we propose a technique for fully automatic video object segmentation in unconstrained settings. Our method is computationally efficient and makes minimal assumptions about the video: the only requirement is for the object to move differently from its surrounding background in a good fraction of the video. The object can be static in a portion of the video and only part of it can be moving in some other portion (e.g. a cat starts running and then stops to lick its paws). Our method does not require a static or slowly moving background (as opposed to classic background subtraction methods [9, 4, 7]). Moreover, it does not assume the object follows a particular motion model, nor that all its points move homogeneously (as opposed to methods based on clustering point tracks [6, 17, 18]). This is especially important when segmenting non-rigid or articulated objects such as animals (fig. 5).

The key new element in our approach is a rapid technique to produce a rough estimate of which pixels are inside the object based on motion boundaries in pairs of subsequent frames (sec. 3.1). This initial estimate is then refined by integrating information over the whole video with a spatio-temporal extension of GrabCut [21, 14, 26]. This second stage automatically bootstraps an appearance model based on the initial foreground estimate, and uses it to refine the spatial accuracy of the segmentation and to also segment the object in frames where it does not move (sec. 3.2).

Through extensive experiments on over 1400 video shots from two datasets [24, 19], we show that our method: (i) handles fast moving backgrounds and objects exhibiting a wide range of appearance, motions and deformations, including non-rigid and articulated objects; (ii) outperforms a state-of-the-art background subtraction technique [4] as well as methods based on clustering point tracks [6, 18, 19]; (iii) is orders of magnitude faster than recent video object segmentation methods based on object proposals [14, 16, 27]; (iv) outperforms the popular method [14] on the large YouTube-Objects dataset [19]; (v) produces competitive results on the small SegTrack benchmark [24]. The source code of our method is released at `http://groups.inf.ed.ac.uk/calvin/software.html`

2. Related Work

Interactive or supervised methods. Several methods for video object segmentation require the user to manually annotate a few frames with object segmentations and then propagate these annotations to all other frames [3, 20, 26]. Similarly, methods based on tracking [8, 24], require the user to mark the object positions in the first frame and then track them in the rest of the video.

Background subtraction. Classic background subtraction methods model the appearance of the background at each pixel and consider pixels that change rapidly to be foreground. These methods typically assume a stationary, or slowly panning camera [9, 4, 7]. The background should change slowly in order for the model to update safely without generating false-positive foreground detections.

Clustering point tracks. Several automatic video segmentation methods track points over several frames and then cluster the resulting tracks based on pairwise [6, 17] or triplet [18] similarity measures. The underlying assumption induced by pairwise clustering [6, 17] is that all object points move according to a single translation, while the triplet model [18] assumes a single similarity transformation. These assumptions have trouble accommodating nonrigid or articulated objects. Our method instead does not attempt to cluster object points and does not assume any kind of motion homogeneity. The object only needs to move sufficiently differently from the background to generate motion boundaries along most of its physical boundary. On the other hand, these methods [6, 17, 18] try to place multiple objects in separate segments, whereas our method produces a simpler binary segmentation (all objects vs background).

Ranking object proposals. The works [14, 16, 27] are closely related to ours, as they tackle the very same task. These methods are based on finding recurring object-like segments, aided by recent techniques for measuring generic object appearance [10], and achieve impressive results on the SegTrack benchmark [24]. While the object proposal infrastructure is necessary to find out which image regions are objects vs background, it makes these methods very slow (minutes/frame). In our work instead, this goal is achieved by a much simpler, faster process (sec. 3.1). In sec. 4 we show that our method achieves comparable segmentation accuracy to [14] while being two orders of magnitude faster.

Oversegmentation. Grundmann et al. [13] oversegment a video into spatio-temporal regions of uniform motion and appearance, analog to still-image superpixels [15]. While this is a useful basis for later processing, it does not solve the video object segmentation task on its own.

3. Our approach

The goal of our work is to segment objects that move differently than their surroundings. Our method has two main stages: (1) efficient initial foreground estimation (sec. 3.1), (2) foreground-background labelling refinement (sec. 3.2). We now give a brief overview of these two stages, and then present them in more detail in the rest of the section.

(1) Efficient initial foreground estimation. The goal of the first stage is to rapidly produce an initial estimate of which pixels might be inside the object *based purely on motion*. We compute the optical flow between pairs of subsequent frames and detect motion boundaries. Ideally, the motion boundaries will form a complete closed curve coinciding with the object boundaries. However, due to inaccuracies in the flow estimation, the motion boundaries are typically incomplete and do not align perfectly with object boundaries (fig. 1f). Also, occasionally false positive boundaries might be detected. We propose a novel, computationally efficient algorithm to robustly determine which pixels reside inside the moving object, taking into account all these sources of error (fig. 2c).

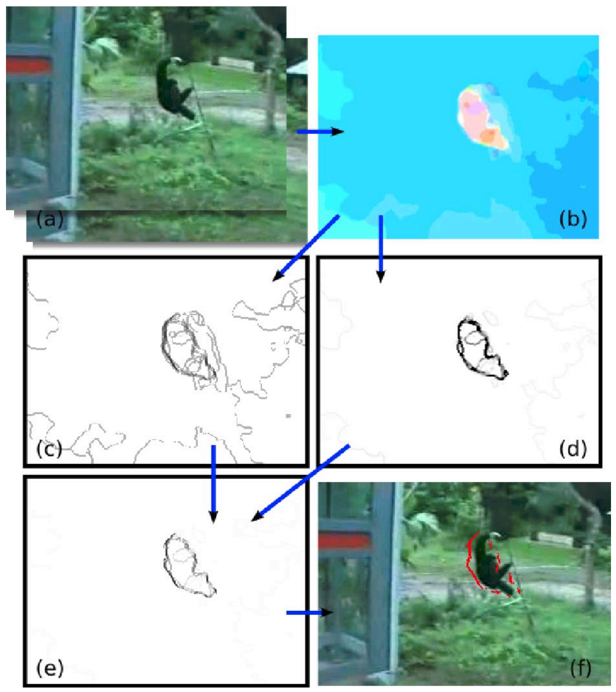

Figure 1. **Motion boundaries.**. *(a) Two input frames. (b) Optical flow $\vec{f_p}$. The hue of a pixel indicates its direction and the color saturation its velocity. (c) Motion boundaries b_p^m, based on the magnitude of the gradient of the optical flow. (d) Motion boundaries b_p^θ, based on difference in direction between a pixel and its neighbours. (e) Combined motion boundaries b_p. (f) Final, binary motion boundaries after thresholding, overlaid on the first frame.*

(2) Foreground-background labelling refinement. As they are purely based on motion boundaries, the inside-outside maps produced by the first stage typically only approximately indicate where the object is. They do not accurately delineate object outlines. Furthermore, (parts of) the object might be static in some frames, or the inside-outside maps may miss it due to incorrect optical flow estimation.

The goal of the second stage is to refine the spatial accuracy of the inside-outside maps and to segment the whole object in all frames. To achieve this, it integrates the information from the inside-outside maps over all frames by (1) encouraging the spatio-temporal smoothness of the output segmentation over the whole video; (2) building dynamic *appearance models* of the object and background under the assumption that they change smoothly over time. Incorporating appearance cues is key to achieving a finer level of detail, compared to using only motion. Moreover, after learning the object appearance in the frames where the

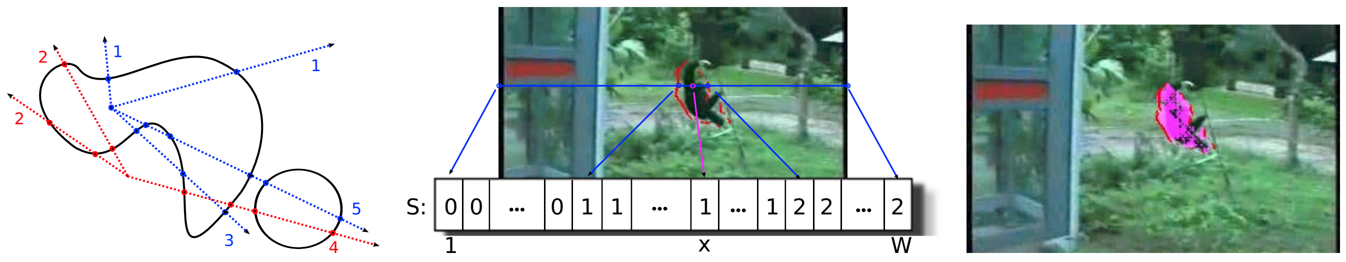

Figure 2. **Inside-outside maps.** *(Left) The ray-casting observation. Any ray originating inside a closed curve intersects it an odd number of time. Any ray originating outside intersects it an even number of times. This holds for any number of closed curves in the image. (Middle) Illustration of the integral intersections data structure S for the horizontal direction. The number of intersections for the ray going from pixel x to the left border can be easily computed as $X_{\text{left}}(x, y) = S(x-1, y) = 1$, and for the right ray as $X_{\text{right}}(x, y) = S(W, y) - S(x, y) = 1$. In this case, both rays vote for x being inside the object. (Right) The output inside-outside map M^t.*

inside-outside maps found it, the second stage uses it to segment the object in frames where it was initially missed (e.g. because it is static).

3.1. Efficient initial foreground estimation

Optical flow. We begin by computing optical flow between pairs of subsequent frames $(t, t+1)$ using the state-of-the-art algorithm [6, 22]. It supports large displacements between frames and has a computationally very efficient GPU implementation [22] (fig. 1a+b).

Motion boundaries. We base our approach on motion boundaries, i.e. image points where the optical flow field changes abruptly. Motion boundaries reveal the location of occlusion boundaries, which very often correspond to physical object boundaries [23].

Let $\vec{f_p}$ be the optical flow vector at pixel p. The simplest way to estimate motion boundaries is by computing the magnitude of the gradient of the optical flow field:

$$b_p^m = 1 - \exp(-\lambda^m ||\nabla \vec{f_p}||) \tag{1}$$

where $b_p^m \in [0, 1]$ is the strength of the motion boundary at pixel p; λ^m is a parameter controlling the steepness of the function.

While this measure correctly detects boundaries at rapidly moving pixels, where b_p^m is close to 1, it is unreliable for pixels with intermediate b_p^m values around 0.5, which could be explained either as boundaries or errors due to inaccuracies in the optical flow (fig. 1c). To disambiguate between those two cases, we compute a second estimator $b_p^\theta \in [0, 1]$, based on the difference in direction between the motion of pixel p and its neighbours $\mathcal{N}$:

$$b_p^\theta = 1 - \exp(-\lambda^\theta \max_{q \in \mathcal{N}} (\delta\theta_{p,q}^2)) \tag{2}$$

where $\delta\theta_{p,q}$ denotes the angle between $\vec{f_p}$ and $\vec{f_q}$. The idea is that if n is moving in a different direction than all its neighbours, it is likely to be a motion boundary. This estimator can correctly detect boundaries even when the object is moving at a modest velocity, as long as it goes in a different direction than the background. However, it tends to produce false-positives in static image regions, as the direction of the optical flow is noisy at points with little or no motion (fig. 1d).

As the two measures above have complementary failure modes, we combine them into a measure that is more reliable than either alone (fig. 1e):

$$b_p = \begin{cases} b_p^m, & \text{if } b_p^m > T \\ b_p^m \cdot b_p^\theta, & \text{if } b_p^m \leq T, \end{cases} \tag{3}$$

where T is a high threshold, above which b_p^m is considered reliable on its own. As a last step we threshold b_p at 0.5 to produce a binary motion boundary labelling (fig. 1f).

Inside-outside maps. The produced motion boundaries typically do not completely cover the whole object boundary. Moreover, there might be false positive boundaries, due to inaccurracy of the optical flow estimation. We present here a computationally efficient algorithm to robustly estimate which pixels are inside the object while taking into account these sources of error.

The algorithm estimates whether a pixel is inside the object based on the point-in-polygon problem [12] from computational geometry. The key observation is that any ray starting from a point inside the polygon (or any closed curve) will intersect the boundary of the polygon an odd number of times. Instead, a ray starting from a point outside the polygon will intersect it an even number of times (figure 2a). Since the motion boundaries are typically incomplete, a single ray is not sufficient to determine whether a pixel lies inside the object. Instead, we get a robust estimate by shooting 8 rays spaced by 45 degrees. Each ray casts a vote on whether the pixel is inside or outside. The final inside-outside decision is taken by majority rule, i.e. a pixel with 5 or more rays intersecting the boundaries an odd number of times is deemed inside.

Realizing the above idea with a naive algorithm would be computationally expensive (i.e. quadratic in the number of pixels in the image). We propose an efficient algorithm which we call *integral intersections*, inspired by the use of integral images in [25]. The key idea is to create a special

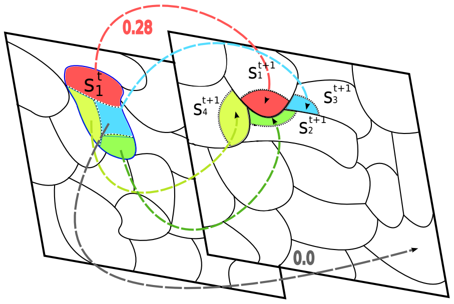

Figure 3. **Example connectivity $\mathcal{E}_t$ over time.** *Superpixel s_1^t contains pixels that lead to s_1^{t+1}, s_2^{t+1}, s_3^{t+1}, s_4^{t+1}. As an example, the weight $\phi(s_1^t, s_1^{t+1})$ is* 0.28 *(all others are omitted for clarity).*

data structure that enables very fast inside-outside evaluation by massively reusing the computational effort that went into creating the datastructure.

For each direction (horizontal, vertical and the two diagonals) we create a matrix S of the same size $W \times H$ as the image. An entry $S(x, y)$ of this matrix indicates the number of boundary intersections along the line going from the image border up to pixel (x, y). For simplicity, we explain here how to build S for the horizontal direction. The algorithm for the other directions is analogous. The algorithm builds S one line y at a time. The first pixel $(1, y)$, at the left image border, has value $S(1, y) = 0$. We then move rightwards one pixel at a time and increment $S(x, y)$ by 1 each time we transition from a non-boundary pixel to a boundary pixel. This results in a line $S(:, y)$ whose entries count the number of boundary intersections (fig. 2b.).

After computing S for all horizontal lines, the data structure is ready. We can now determine the number of intersections X for both horizontal rays (left→right, right→left) emanating from a pixel (x, y) in *constant time* by

$$X_{\text{left}}(x, y) = S(x - 1, y) \quad (4)$$
$$X_{\text{right}}(x, y) = S(W, y) - S(x, y) \quad (5)$$

where W is the width of the image, i.e. the rightmost pixel in a line (fig. 2b).

Our algorithm visits each pixel exactly once per direction while building S, and once to compute its vote, and is therefore *linear* in the number of pixels in the image. The algorithm is very fast in practice and takes about 0.1s per frame of a HD video (1280x720 pixels) on a modest CPU (Intel Core i7 at 2.0GHz).

For each video frame t, we apply the algorihtm on all 8 directions and use majority voting to decide which pixels are inside, resulting is an inside-outside map M^t (fig. 2c).

3.2. Foreground-background labelling refinement

We formulate video segmentation as a pixel labelling problem with two labels (foreground and background). We oversegment each frame into superpixels $\mathcal{S}^t$ [15], which greatly reduces computational efficiency and memory usage, enabling to segment much longer videos.

Each superpixel $s_i^t \in \mathcal{S}^t$ can take a label $l_i^t \in \{0, 1\}$. A labelling $\mathcal{L} = \{l_i^t\}_{t,i}$ of all superpixels in all frames represents a segmentation of the video. Similarly to other segmentation works [14, 21, 26], we define an energy function to evaluate a labeling

$$E(\mathcal{L}) = \sum_{t,i} A_i^t(l_i^t) + \alpha_1 \sum_{t,i} L_i^t(l_i^t) \quad (6)$$
$$+ \alpha_2 \sum_{(i,j,t)\in\mathcal{E}_s} V_{ij}^t(l_i^t, l_j^t) + \alpha_3 \sum_{(i,j,t)\in\mathcal{E}_t} W_{ij}^t(l_i^t, l_j^{t+1})$$

A^t is a unary potential evaluating how likely a superpixel is to be foreground or background according to the appearance model of frame t. The second unary potential L^t is based on a location prior model encouraging foreground labellings in areas where independent motion has been observed. As we explain in detail later, we derive both the appearance model and the location prior parameters from the inside-outside maps M^t. The pairwise potentials V and W encourage spatial and temporal smoothness, respectively. The scalars α weight the various terms.

The output segmentation is the labeling that minimizes (6):

$$\mathcal{L}^* = \operatorname*{argmin}_{\mathcal{L}} E(\mathcal{L}) \quad (7)$$

As E is a binary pairwise energy function with submodular pairwise potentials, we minimize it exactly with graph-cuts. Next we use the resulting segmentation to re-estimate the appearance models and iterate between these two steps, as in GrabCut [21]. Below we describe the potentials in detail.

Smoothness V, W. The spatial smoothness potential V is defined over the edge set $\mathcal{E}_s$, containing pairs of spatially connected superpixels. Two superpixels are spatially connected if they are in the same frame and are adjacent.

The temporal smoothness potential W is defined over the edge set $\mathcal{E}_t$, containing pairs of temporally connected superpixels. Two superpixels s_i^t, s_j^{t+1} in subsequent frames are connected if there at least one pixel of s_i^t moves into s_j^{t+1} according to the optical flow (fig. 3).

The functions V, W are standard contrast-modulated Potts potentials [21, 26, 14]:

$$V_{ij}^t(l_i^t, l_j^t) = \text{dis}(s_i^t, s_j^t)^{-1}[l_i^t \neq l_j^t] \exp(-\beta \text{col}(s_i^t, s_j^t)^2) \quad (8)$$
$$W_{ij}^t(l_i^t, l_j^{t+1}) = \phi(s_i^t, s_j^{t+1})[l_i^t \neq l_j^t] \exp(-\beta \text{col}(s_i^t, s_j^{t+1})^2) \quad (9)$$

where dis is the Euclidean distance between the centres of two superpixels and col is the difference between their average RGB color. The factor that differs from the standard definition is ϕ, which is the percentage of pixels within the two superpixels that are connected by the optical flow. This is a better weight than the Euclidean distance, as it is invariant of the speed of the motion.

Appearance model $\mathbf{A^t}$. The appearance model consists of two Gaussian Mixture Models over RGB colour values[1], one for the foreground (fg) and one for the background (bg). In the task of interactive segmentation [21], where this methodology originated, the appearance model parameters are estimated from some manually labelled pixels. In this paper instead, we estimated them automatically based on the inside-outside maps M^t (sec. 3.1).

We estimate appearance models A^t for each frame t. However, since the appearance of the fg and bg typically changes smoothly over time, these models are tightly coupled as their estimation integrates information over the whole video. Hence, the collection of per-frame models can be seen as a single *dynamic appearance model*.

At each frame t we estimate a fg model from all superpixels in the video, weighted by how likely they are to be foreground and by how close in time they are to t. More precisely, the weight of each superpixel $s_i^{t'}$ in frame t' is

$$\exp(-\lambda^A \cdot (t-t')^2) \cdot r_i^{t'} \tag{10}$$

The first factor discounts the weight of $s_i^{t'}$ over time. The second factor is the percentage of pixels of $s_i^{t'}$ that are inside the object according to the inside-outside map $M^{t'}$. The estimation of bg appearance models is analogous, with the second factor replaced by $1 - r_i^{t'}$ (i.e. the ratio of pixels considered to be outside the object).

After estimating the foreground-background appearance models, the unary potential $A_i^t(l_i^t)$ is the log-probability of s_i^t to take label l_i^t under the appropriate model (i.e. the foreground model if $l_i^t = 1$ and the background one otherwise).

Having these appearance models in the segmentation energy (6) enables to segment the object more accurately than possible from motion alone, as motion estimation is inherently inaccurate near occlusion boundaries. Moreover, the appearance models are integrated over large image regions and over many frames, and therefore can robustly estimate the appearance of the object, despite faults in the inside-outside maps. The appearance models then transfer this knowledge to other positions within a frame and to other frames, by altering towards foreground the unary potential of pixels with object-like appearance, even if the inside-outside maps missed them. This enables completing the segmentation in frames where only part of the object is moving, and helps segmenting it even in frames where it does move at all.

Location model $\mathbf{L^t}$. When based only on appearance, the segmentation could be distracted by background regions with similar colour to the foreground (even with perfect appearance models). Fortunately, the inside-outside maps can provide a valuable *location prior* to anchor the segmentation to image areas likely to contain the object, as they move differently from the surrounding region. However, in some frames (part of) the object may be static, and in others the inside-outside map might miss it because of incorrect optical flow estimation (fig. 4, middle row). Therefore, directly plugging the inside-outside maps as unary potentials in L^t would further encourage an all-background segmentation in frames where they missed the object.

We propose here to *propagate* the per-frame inside-outside maps over time to build a more complete location prior L^t. The key observation is that 'inside' classifications are more reliable than 'outside' ones: the true object boundaries might not form a near-closed motion boundary due to the reasons above, but accidental near-closed boundaries rarely form out of noise. Therefore, our algorithm *accumulates* inside points over the entire video sequence, following the optical flow (fig. 4, bottom row).

The algorithm proceeds recursively. The value of the location prior at a superpixel s_i^t is initially $L_i^t := r_i^t$, i.e. the percentage of its pixels that are inside the object according to the inside-outside map M^t. We start propagating from frame 1 to frame 2, then move to frame 3 and so on. At each step, the value of the location prior for a superpixel s_j^{t+1} in frame $t+1$ gets updated to

$$L_j^{t+1} := L_j^{t+1} + \gamma \frac{\sum_i \phi(s_i^t, s_j^{t+1}) \cdot \psi(s_i^t) \cdot L_i^t}{\sum_i \phi(s_i^t, s_j^{t+1})} \tag{11}$$

where the summation runs over all superpixels in frame t; the connection weight ϕ is the percentage of pixels in superpixel s_i^t that connect to superpixel s_j^{t+1} by following the optical flow (fig. 3); $\gamma \in [0,1]$ controls the rate of accumulation; ψ is a transfer quality measure, down-weighting propagation if the optical flow for s_i^t is deemed unreliable

$$\psi(s_i^t) = \exp(-\lambda^\psi \sum_{p \in s_i^t} ||\nabla \vec{f_p}||) \tag{12}$$

In essence, ψ measures the sum of the flow gradients in s_i^t; large gradients can indicate depth discontinuities, where the optical flow is often inaccurate, or that s_i^t might cover bits of two different objects.

We run the forward propagation step above and an analogous backward step, starting from the last frame towards the first one. These two steps are run independently. The final location prior L^t is the normalised sum of the two steps.

4. Experimental evaluation

We evaluate our method on two datasets: SegTrack [24] and YouTube-Objects [19]. The parameters $\lambda, T, \beta, \gamma$ are kept fixed to the same values in all experiments.

4.1. SegTrack

Dataset. SegTrack [24] was originally introduced to evaluate tracking algorithms, and it was adopted to benchmark

[1]As the basic units are superpixels, all measurements refer to their average RGB value.

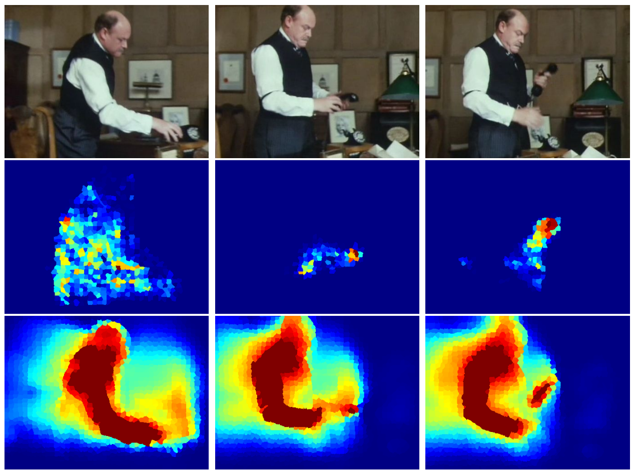

Figure 4. **Location model.** *Top row: three video frames. Middle row: likelihood of foreground based on the inside-outside maps in individual frames. They miss large parts of the person in the second and third frames, as the head and torso are not moving. Bottom row: the location model based on propagating the inside-outside maps. It includes most of the person in all frames.*

video object segmentation by [14]. It contains 6 videos (monkeydog, girl, birdfall, parachute, cheetah, penguin) and pixel-level ground-truth for the foreground object in every frame. Following [14], we discard the penguin video, since only a single penguin is labelled in the ground-truth, amidst a group of penguins. The videos offer various challenges, including objects of similar color to the background, non-rigid deformations, and fast camera motion (fig. 5).

Setup. As in [14, 24], we quantify performance with the number of wrongly labeled pixels, averaged over all frames of a video. We set the weights α of the energy function (6) by two-fold cross-validation. We split the dataset into two sets of 3 and 2 videos respectively, and train the α weights in each set. When testing our method on the videos in one set, we use the weights trained on the other.

We compare to several methods [14, 16, 27, 6, 18, 4]. The video object segmentation method of Lee at al. [14] returns a ranked list of spatio-temporal segments likely to be objects. We report the results from their paper, which evaluates the segment corresponding to the ground-truth object, out of the top 4 segments returned by the algorithm ([14], fig. 6). In contrast, our method directly returns a single foreground segment, as it discovers the foreground object automatically. We also report the results of another two methods based on ranking object proposals [16, 27].

We also compare to a state-of-the-art background subtraction method [4] and with two state-of-the-art clustering point tracks based methods [6, 18]. We used the implementations provided by the respective authors[2]. As the latter are designed to return multiple segments, we report results for the segment best matching the ground-truth segmentation.

[2]http://www2.ulg.ac.be/telecom/research/vibe/ http://lmb.informatik.uni-freiburg.de/resources/software.php

precision	ours	[14]	[16]	[27]	[6]	[18]	[4]
birdfall	217	288	189	155	468	468	606
cheetah	890	905 (34228)	806	633	1968	1175	11210
girl	3859	1785	1698	1488	7595	5683	26409
monkey	284	521 (64339)	472	365	1434	1434	12662
parachute	855	201	221	220	1113	1595	40251

Table 1. **Results on SegTrack.** *The entries show the average number of mislabelled pixels per frame. For [14], the numbers in parenthesis refer to the single top ranked hypothesis, as given to us by the authors in personal communication.*

Results. As table 1 shows, even the recent background-subtraction method [4] performs poorly on this data, since it cannot handle fast camera motion. The point clustering methods [6, 18] produce better results, as they can better cope with these conditions.

Our method considerably outperforms [6, 4, 18] in all videos, as it handles non-rigid objects better, and tightly integrates appearance along with motion as segmentation cues. Overall, our performance is about on par with [14]. This is remarkable, given that our approach is simpler, does not require manual selection of the output segment, and is two orders of magnitude faster (sec. 4.3). For reference, we also reports the accuracy of the single top-ranked segment by [14]. In this fully automatic mode, their method completely misses the object in *cheetah* and *monkey*. The very recent method [27] achieves lower errors than ours on average, but is much slower (sec. 4.3).

Fig. 5 shows example frames from all 5 videos. Our method accurately segments all videos but *girl*, as it misses parts of her legs and arms. The higher error on *parachute* is due to including the paratrooper in the segmentation, as it is not annotated in the ground-truth. Note the high quality of the segmentation on monkeydog and cheetah, which feature fast camera motion and strong non-rigid deformations.

In general, inspecting the results reveals that all of [14, 16, 27] and our method solve this dataset well. All methods lock on the object in all videos and accuracy differences between methods are due to finer localization of the object boundaries. When also taking into account that it contains only 5 very short videos, we believe this dataset is saturated.

4.2. YouTube-Objects

Dataset. YouTube-Objects [19][3] is a large database collected from YouTube containing many videos for each of 10 diverse object classes. The videos are completely unconstrained and very challenging, featuring large camera motion, diverse backgrounds, illumination changes and editing

[3]http://groups.inf.ed.ac.uk/calvin/learnfromvideo

	aero	bird	boat	car	cat	cow	dog	horse	mbike	train	avg
Clustering tracks [6]	53.9	19.6	38.2	37.8	32.2	21.8	27.0	34.7	45.4	37.5	34.8
Automatic segment selection [19]	51.7	17.5	34.4	34.7	22.3	17.9	13.5	26.7	41.2	25.0	28.5
ours	65.4	67.3	38.9	65.2	46.3	40.2	65.3	48.4	39.0	25.0	50.1

Table 2. **Results on YouTube-Objects.** *The entries show the average per-class CorLoc ('aero' to 'train') as well as the average over all classes ('avg'). Top row: the best segment returned by the method of [6]. Middle row: the segment automatically selected by the method of [19], out of those produced by [6]. Bottom row: the segment output by our method.*

effects (e.g. fade-ins, flying logos). The objects undergo rapid movement, strong scale and viewpoint changes, non-rigid deformations, and are sometimes clipped by the image border (fig. 5). The dataset also provides ground-truth bounding-boxes on the object of interest in one frame for each of **1407 video shots**.

Setup. We adopt the CorLoc performance measure of [19], i.e. the percentage of ground-truth bounding-boxes which are correctly localized up to the PASCAL criterion [11] (intersection-over-union ≥ 0.5). For the purpose of this evaluation, we automatically fit a bounding-box to the largest connected component in the pixel-level segmentation output by our method. We set the α weights by manual inspection on a few shots (about 5). The same weights are then used for all 1407 shots in the database.

We compare to [6, 19] and report their performance as originally stated in [19]. For [6] they report results for the segment with the maximum overlap with the ground-truth bounding-box (analogous to our experiment on SegTrack). Prest et al. [19] automatically select one segment per shot among those produced by [6], based on its appearance similarity to segments selected in other videos of the same object class, and on how likely it is to cover an object according to a class-generic objectness measure [2]. As it returns a single foreground segment per shot, this method is directly comparable to ours.

We also run [14] on 50 videos (5/class) using the implementation by their authors[4], as it is too slow to run on the whole database. For evaluation we fit a bounding-box to the top ranked output segment.

Results. As table 2 shows, our method substantially improves over the result of [19], from 28.5% to 50.1% on average over all classes. Moreover, our method also outperforms the best segment produced by [6], confirming what we observed on the SegTrack dataset. On the 50-video subset, our method produces 42.0% CorLoc, considerably above the 28.0% reached by [14]. This departs from what observed on SegTrack and suggests that our method generalizes better to a wide variety of videos.

Fig. 5 shows example results. The cat, dog, and motorbike examples show fast camera motion, large scale and viewpoint changes, and non-rigid deformations. On the bird video our method segments both the bird and the hand, as it considers them both foreground. The horse example shows our method correctly segment objects even if largely clipped by the image border in some frames, as it automatically transfers object appearance learned in other frames.

[4]https://webspace.utexas.edu/yl3663/~ylee/

4.3. Runtime

Given optical flow and superpixels, our method takes **0.5 sec/frame** on SegTrack (0.05 sec for the inside-outside maps and the rest for the foreground-background labelling refinement). In contrast, [14] takes $>$ 300 sec/frame, with about 120 sec/frame for generating the object proposals [10]. The point track clustering method [6] takes 7-44 sec/frame depending on the video, and [18] takes 43-360 sec/frame. While [16, 27] do not report timings nor have code available for us to measure, their runtime must be $>$ 120 sec/frame as they also use the object proposals [10].

All timings were measured on the same computer (Intel Core i7 2.0GHz), and exclude optical flow computation, which all methods require as input. High quality optical flow can be computed rapidly using [22] ($<$ 1 sec/frame). Currently, we use TurboPixels as superpixels [15] (1.5 sec/frame), but even faster alternatives are available [1].

This analysis shows that our method is a lot faster than these competitors and is in fact efficient enough to be applied to very large collections of videos.

References

[1] R. Achanta, A. Shaji, K. Smith, A. Lucchi, P. Fua, and S. Ssstrunk. Slic superpixels compared to state-of-the-art superpixel methods. *IEEE Trans. on PAMI*, 34(11), 2012.

[2] B. Alexe, T. Deselaers, and V. Ferrari. What is an object? In *CVPR*, 2010.

[3] X. Bai, J. Wang, D. Simons, and G. Sapiro. Video snapcut: robust video object cutout using localized classifiers. In *SIGGRAPH*, 2009.

[4] O. Barnichm and M. Van Droogenbroeck. Vibe: A universal background subtraction algorithm for video sequences. *IEEE Trans. Image Processing*, 2011.

[5] M. Blank, L. Gorelick, E. Shechtman, M. Irani, and R. Basri. Actions as space-time shapes. In *ICCV*, 2005.

[6] T. Brox and J. Malik. Object segmentation by long term analysis of point trajectories. In *ECCV*, 2010.

[7] S. Brutzer, B. Hoeferlin, and G. Heidemann. Evaluation of background subtraction techniques for video surveillance. In *CVPR*, 2011.

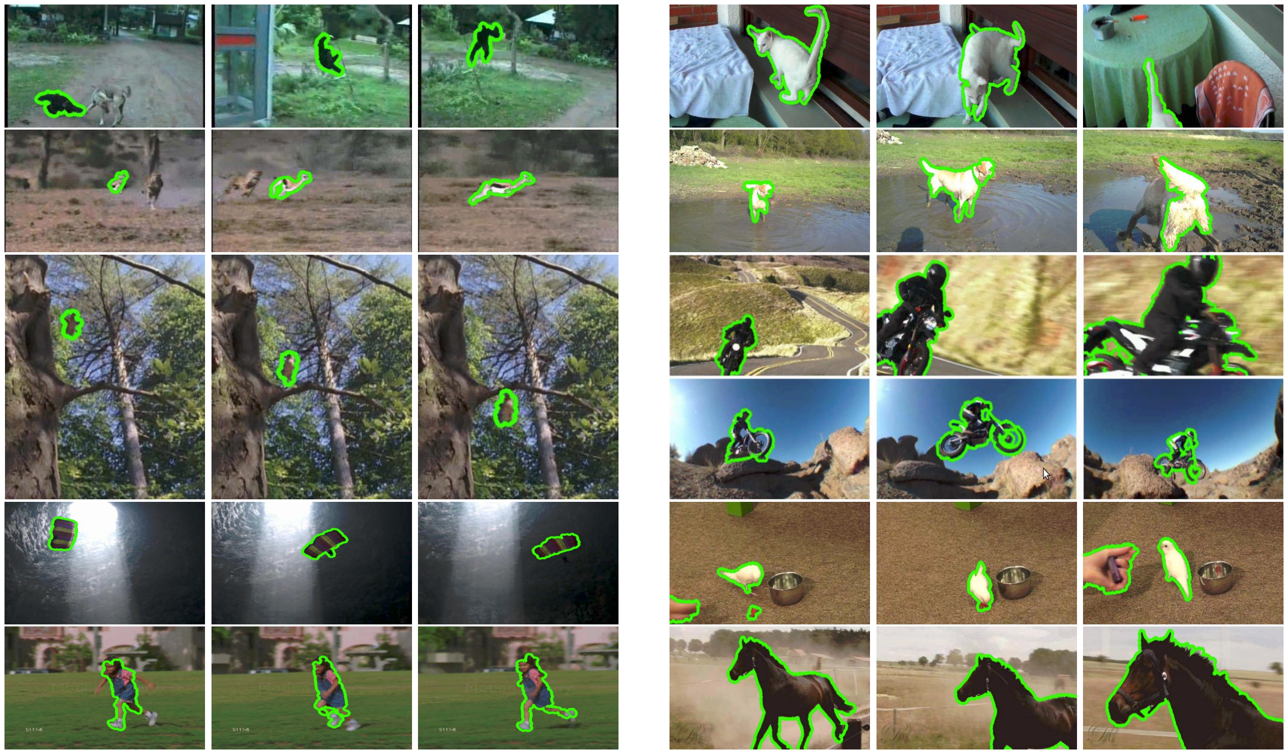

Figure 5. **Example results.** *We show 3 example frames per video, with the output of our method overlaid in green. (Left)* **SegTrack**. *Top to bottom: monkeydog, cheetah, birdfall, parachute, girl. (Right)* **YouTube-Objects**. *Top to bottom: cat, dog, motorbike, bird, horse. We include example result videos in the supplementary material.*

[8] P. Chockalingam, S. N. Pradeep, and S. Birchfield. Adaptive fragments-based tracking of non-rigid objects using level sets. In *ICCV*, 2009.

[9] R. Cucchiara, C. Grana, M. Piccardi, and A. Prati. Detecting moving objects, ghosts, and shadows in video streams. *IEEE Trans. on PAMI*, 2003.

[10] I. Endres and D. Hoiem. Category independent object proposals. In *ECCV*, 2010.

[11] M. Everingham et al. The PASCAL Visual Object Classes Challenge 2010 Results, 2010.

[12] J. Foley, A. van Dam, S. Feiner, and J. Hughes. *Computer Graphics: Principles and Practice, 2nd Edition*. Addison-Wesley, 1990.

[13] M. Grundmann, V. Kwatra, M. Han, and I. Essa. Efficient hierarchical graph-based video segmentation. In *CVPR*, 2010.

[14] Y. J. Lee, J. Kim, and K. Grauman. Key-segments for video object segmentation. In *ICCV*, 2011.

[15] A. Levinshtein, A. Stere, K. N. Kutulakos, D. J. Fleet, S. J. Dickinson, and K. Siddiqi. "TurboPixels: Fast Superpixels Using Geometric Flows. *IEEE Trans. on PAMI*, 2009.

[16] T. Ma and L. J. Latecki. Maximum weight cliques with mutex constraints for video object segmentation. In *CVPR*, 2012.

[17] P. Ochs and T. Brox. Object segmentation in video: A hierarchical variational approach for turning pont trajectories into dense regions. In *ICCV*, 2011.

[18] P. Ochs and T. Brox. Higher order motion models and spectral clustering. In *CVPR*, 2012.

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

[20] B. L. Price, B. S. Morse, and S. Cohen. Livecut: Learning-based interactive video segmentation by evaluation of multiple propagated cues. In *ICCV*, 2009.

[21] C. Rother, V. Kolmogorov, and A. Blake. Grabcut: Interactive foreground extraction using iterated graph cuts. In *SIGGRAPH*, 2004.

[22] N. Sundaram, T. Brox, and K. Keutzer. Dense point trajectories by gpu-accelerated large displacement optical flow. In *ECCV*, 2010.

[23] P. Sundberg, T. Brox, M. Maire, P. Arbelaez, and J. Malik. Occlusion boundary detection and figure/ground assignment from optical flow. In *CVPR*, 2011.

[24] D. Tsai, M. Flagg, and J. Rehg. Motion coherent tracking with multi-label mrf optimization. In *BMVC*, 2010.

[25] P. Viola and M. Jones. Robust real-time object detection. In *IJCV*, 2001.

[26] T. Wang and J. Collomosse. Probabilistic motion diffusion of labeling priors for coherent video segmentation. *IEEE Trans. Multimedia*, 2012.

[27] J. O. Zhang, D. and M. Shah. Video object segmentation through spatially accurate and temporally dense extraction of primary object regions. In *CVPR*, 2013.