Objects do not disappear:
Video object detection by single-frame object location anticipation

Xin Liu1 Fatemeh Karimi Nejadasl2 Jan C. van Gemert1 Olaf Booij1 Silvia L. Pintea1
Computer Vision Lab, Delft University of Technology1
Institute for Biodiversity and Ecosystem Dynamics, University of Amsterdam2

Abstract

Objects in videos are typically characterized by continuous smooth motion. We exploit continuous smooth motion in three ways. 1) Improved accuracy by using object motion as an additional source of supervision, which we obtain by anticipating object locations from a static keyframe. 2) Improved efficiency by only doing the expensive feature computations on a small subset of all frames. Because neighboring video frames are often redundant, we only compute features for a single static keyframe and predict object locations in subsequent frames. 3) Reduced annotation cost, where we only annotate the keyframe and use smooth pseudo-motion between keyframes. We demonstrate computational efficiency, annotation efficiency, and improved mean average precision compared to the state-of-the-art on four datasets: ImageNet VID, EPIC KITCHENS-55, YouTube-BoundingBoxes and Waymo Open dataset. Our source code is available at https://github.com/L-KID/Video-object-detection-by-location-anticipation.

1. Introduction

Humans assume object permanence: blink your eyes, and the world is still there. Similarly, video frames are redundant, and missing some frames when watching a movie does not drastically change the scene. Actually, for the parts that did change, if these parts changed coherently, they might hint at a sense of objectness, as hypothesized by the Gestalt law of common fate.

As illustrated in Fig. 1, here we explore these observations in the context of video object detection in three ways: 1) Improved accuracy by exploiting an additional source of supervision: the coherent motion from the law of common fate; by predicting object motion from a static image. 2) Improved efficiency by exploiting redundancy to reduce computational cost by only processing sampled keyframes and predict object motion for the missing frames in-between. 3) Reduced annotation cost by only annotating sampled keyframes. Thus, we improve accuracy and save computation and annotation time by simply skipping the feature computation and/or the annotation for a large majority of the frames.

We make the following contributions: (i) A video object detection method that samples static keyframes and predicts object motion for unseen future frames. (ii) Computational efficiency, as our method only extracts features for sampled static keyframes; (iii) Data efficiency, as we use sparse annotations only at the sampled keyframes, hallucinating motion in-between these sparse annotations. (iv) Our extensive experimental results on ImageNet-VID [58], EPIC KITCHENS-55 [11], YouTube-BoundingBoxes [56] and Waymo Open dataset [15] show that our approach improves accuracy over the state-of-the-art methods, while being faster at both training and inference time.
2. Related work

**Video object detection.** Several methods do temporal modeling by post-processing still-image object detectors such as post-processing Faster-RCNN [27, 40] or post-processing Mask R-CNN [4, 44]. Alternatives include recurrent blocks [49, 65] or optical flow [80, 78]. This can be further extended with instance and pixel-level calibrations over time [70], or using a space-time lattice [6]. More recently, the field has advanced by aggregating temporal information: either by defining detection correlations as a graph in SELSA [72], or by using global and local temporal pooling in MEGA [7]. MEGA is further extended by considering all pairwise frames in TF-Blender [9], while HVR-Net [24] integrates inter-video and intra-video object relations. IFF-Net [38] uses a feature flow estimating module to indicate the feature displacement. LWDN [36] does not simply aggregate features, but aligns the features between keyframes by adopting a memory mechanism. In contrast, we avoid computationally demanding optical flow and recurrent blocks, and we do not aggregate neighboring frames, or use temporal heuristics to post-process still-image detections. Instead, we anticipate future object locations over time from a single static keyframe, placing temporal prediction at the heart of our model.

**Single image motion prediction versus tracking.** Object tracking aims to predict object location in a video. A recent overview of object tracking is given in [8]. In classical methods, the bounding box around the object to track is given a priori [61], but can also be estimated by an object detector [3, 18, 32, 64]. The object location in the next frame can be estimated with siamese networks [3], or object detections at every frame are linked into tracks by motion regression [18], or by trajectory forecasting [12]. Our method is inspired by forecasting in tracking, yet we are different from tracking because we predict the object location in future frames from a single static input image, without actually using the image features of the future frames.

**Anticipating motion from a static single image.** A single static frame is rich enough to allow predicting of future appearance [33, 46, 52, 67], actions [1, 20, 50, 60], and motion [19, 54, 68]. Such motion predictions, in turn, can then be used for predicting pedestrians’ behaviors [55, 76] or other road agents [23]. Moreover, motion prediction can be used as a self-supervision cue [17, 25, 26, 43, 71] to improve feature learning. Inspired by these works, we use location anticipation as an additional source of supervision to improve the accuracy of the still-image detector, while we also exploit it for efficiency as it allows us to only compute features on a subset of still-images, avoiding expensive feature computations on all frames.

**Efficiency in video.** Because videos typically sample several frames per second (FPS) it is important to have efficient video analysis methods. Successful prior work proposes network architecture adaptations to reduce computations [16, 41, 45, 47, 53, 81]. For video object detection, it is efficient to adapt the detector online [59] or to transfer an image detector to video [39]. Alternatively, computations can be reduced by focusing only on specific video regions [37, 63, 75]. Similarly, we also focus on efficiency in video object detection. We are efficient by predicting object trajectories a few frames ahead and therefore saving computations by not processing those frames.

**Sparse annotations in video.** Training a model on sparse video annotations can be done by iteratively updating the model in a boosting fashion [2], or propagating groups of pixels through time [35]. More recent work generates dense object masks in videos from sparse bounding boxes [74]. Rather than focusing on improving the model accuracy on sparsely annotated videos, prior work has analyzed what is the accuracy vs. annotation effort trade-off [51]. Specifically, for object locations in video, annotations can be generated through a combination of tracking, frame selection and active learning [10, 34, 42, 66]. Inspired by these works we also explore a variant of our model that allows sparse annotations. We require bounding-box annotations on the static keyframes, and we experimentally show that we can hallucinate the motion between keyframes, removing the need to annotate all frames.

3. Anticipating object locations

**Object detection backbone.** We illustrate our method in Fig. 2. We start from a standard static image object detection backbone. We input static frames uniformly sampled from the video with a time step $T$; we call such static frames “keyframes”. For each keyframe at time $t$, the object detector gives a set of $N$ proposal bounding boxes: $B_t = \{B_i^t\}_{i=1}^N$, where $B_i^t = (x_i^t, y_i^t, w_i^t, h_i^t)$, and $(x_i^t, y_i^t)$ are the top-left corner and $(w_i^t, h_i^t)$ are the width and height of the bounding box. Each $B_i^t$ corresponds to a possible object and has an associated class $c_i^t$: $\{(B_i^t, c_i^t)\}_{i=1}^N$.

**Trajectory subnetwork.** Starting from the keyframe detection bounding boxes $B_t$, we anticipate object trajectories – defined as the future bounding box locations of an object over the following $T$ frames. The trajectory subnetwork is highlighted in green in Fig. 2. For each keyframe indexed by $t$, we define a batch of length $T$ and we input into the trajectory network three types of inputs: (a) a vector of time indices of the future trajectories relative to the keyframe index $f$ batched from $[1, \ldots, T]$; (b) the set of bounding boxes detected at the keyframe $B_t = \{B_i^t\}_{i=1}^N$ repeated over all $T$ time steps: $\{\{B_i^t\}_{i=1}^N\} \times T$; and (c) the static keyframe feature maps extracted via the network mapping $f$ from the the bounding boxes $\{f_i^t = f(B_i^t)\}_{i=1}^N$ also broadcast for
We want to optimize for each object with the best matching IoU score of the ground truth box.

### 3.2. Trajectory loss

To optimize the trajectories we need an associated object for each frame based on their overlap with the ground truth boxes. Moreover, we also need to associate ground truth boxes to their associated trajectory, starting at keyframe bounding box, yielding:

\[
\{B_t\}_{t=1}^T \rightarrow \{\text{predicted boxes}\}_{t=1}^T
\]

We find the best matching object trajectories, one trajectory for each of the \(N\) detected objects. Each trajectory starts at the keyframe indexed by \(t\) and extends up to frame \(t+T\). Concretely, our trajectory predictions are:

\[
\{T_i\}_{i=1}^N = (B_{t_{i}}, B_{t_{i}+1}, \ldots, B_{t_{i}+T})_{i=1}^N
\]

where we also add the keyframe bounding box \(B_{t_{i}}\) to the trajectory.

The output of the trajectory subnetwork is a list of trajectories, one trajectory for each of the \(N\) detected objects. Each trajectory starts at the keyframe indexed by \(t\) and extends up to frame \(t+T\). Concretely, our trajectory predictions are:

\[
\{T_i\}_{i=1}^N = (B_{t_{i}}, B_{t_{i}+1}, \ldots, B_{t_{i}+T})_{i=1}^N
\]

where we also add the keyframe bounding box \(B_{t_{i}}\) to the trajectory.

### 3.1. Associating trajectories to ground truth

To optimize the trajectories we need an associated object class for each trajectory. Each object trajectory \(T_i\), indexed by \(i\), starts with a keyframe bounding box \(B_{t_{i}}\), which has a corresponding object class \(c_{t_{i}}\). Each trajectory corresponds to one object, thus we let each trajectory inherit the class of its starting keyframe bounding box, yielding:

\[
\{(T_i, c_i)\}_{i=1}^N
\]

Moreover, we also need to associate ground truth boxes \(B^*\) with all predicted boxes along each object trajectory \(T_i=(B_{t_{i}}, B_{t_{i}+1}, \ldots, B_{t_{i}+T})\). Following the standard procedure [30, 57] we rank the predicted boxes \(B_{t_{i}+l}\), \(l \in \{0, \ldots, T\}\) based on their overlap with the ground truth boxes \(B^*_{t_{i}+l}\) at each frame \(t+l\). We associate each of the \(N\) predicted boxes with the best matching IoU score of the ground truth box.

### 3.2. Trajectory loss

#### Bag of boxes loss.

We want to optimize for each object indexed by \(i\) its associated trajectory, starting at keyframe \(t\):

\[
L_{\text{bag}}(B^*, \{B_{t_{i}}, \ldots, B_{t_{i}+T}\}) = \sum_{t=0}^{T} L_1(B^*_{t+l} - B_{t+l}).
\]

#### Trajectory cumulative loss.

The downside of Eq. (1) is that it treats each prediction \(B_{t_{i}+l}\) as if it were independent of its neighboring predictions along the trajectory. \(B_{t_{i}+l-1}\) and \(B_{t_{i}+l+1}\). Therefore, there is no temporal ordering enforced in the \(L_{\text{bag}}\) loss. Not enforcing the ordering of the
predictions along the trajectory could lead to discontinuous trajectories. We want to enforce smoothness in the predictions over time: objects cannot disappear or appear at random locations, between neighboring frames. Specifically, the trajectory is piecewise continuous: for an object to move from a location \( B_{t+1} \) to a location \( B_{t+4} \) it has to travel through the intermediate locations \( B_{t+2} \) and \( B_{t+3} \), as illustrated in Fig. 3.

To add this insight, we define a loss that constrains the pairwise offsets along the trajectory from frame \( t \) up to every frame \( t+l \): \( \delta_{t+k} = (B_{t+k} - B_{t+k-1}) \), \( k \in \{1, \ldots, l\} \), to add up to the offset from the ground truth at the frame \( t+l \) to the keyframe prediction \( (B^*_t - B_t) \). Concretely:

\[
L^{\Sigma}(B^*, \hat{t}_T) = \sum_{l=0}^{T} \mathcal{L}_1 \left( (B^*_{t+l} - B_t) - \sum_{k=1}^{l} \delta_{t+k} \right),
\]

(2)

where we redefine the trajectory to predict \( \delta_{t+l} \) values describing pairwise offsets instead of bounding boxes: \( \hat{t}_T = (\delta_{t+1}, \ldots, \delta_{t+T}) \). Additionally, we ignore the calculation of the coordinates, if the ground truth bounding boxes \( B^*_{t+l} \) are not valid in Eq. (2). Note, that if we would predict bounding boxes \( B_{t+l} \) in the trajectory network, instead of offsets between pairs of bounding boxes, \( \delta_{t+l} \), Eq. (2) would reduce to Eq. (1) and the temporal ordering would not be enforced (See the derivation in the supplementary material).

In Eq. (2) the inner loop over pairwise offsets \( \delta_{t+k} \) accumulates the errors in the predictions over time from frame \( t+1 \) to frame \( t+l \). To make sure the errors do not accumulate along the trajectory, and the change from frame to frame is smooth, we add another loss \( L^{\text{bag}}(\delta) \) constraining the pairwise offsets \( \delta_{t+l} \) at every timestep \( t+l \) to map back to ground truth offsets: \( \delta^*_{t+l} = (B^*_{t+l} - B^*_{t+l-1}) \):

\[
L^{\text{bag}}(\delta)(B^*, \hat{t}_T) = \sum_{l=1}^{T} \mathcal{L}_1 \left( \delta^*_{t+l} - \delta_{t+l} \right).
\]

(3)

Our final loss \( L_{\text{trajectory}} \) is then a combination of the two losses where the cumulative trajectory loss \( L^{\Sigma} \) enforces piecewise continuity in the predictions and the bag of offsets loss \( L^{\text{bag}}(\delta) \) discourages errors from accumulating along the trajectory and makes the trajectory smooth:

\[
L_{\text{trajectory}} = L^{\Sigma} + L^{\text{bag}}(\delta).
\]

(4)

We investigate the effect of each loss term in the experimental section, together with the effect of predicting offsets \( \delta_{t+l} \) instead of box coordinates \( B_{t+l} \) in the trajectory network.

**Sparse annotation loss.** When there are no annotations in-between keyframes then we cannot optimize the anticipated trajectory. Since the task in the sparse annotation cases is object detection on annotated keyframes, we can hypothesize that the precise true location of an object along a trajectory \( B^*_{t+l} \), is not essential, as long as the trajectory is piecewise continuous and smooth, and the starting and ending points of the trajectory are known. Therefore, we can rewrite our losses in Eq. (4) to a sparsely-annotated variant \( L^{\text{trajectory}} \) by changing the way in which we define the box-supervision. Explicitly, we replace the set of ground truth boxes \( \{B^*_1, B^*_2, \ldots, B^*_T\} \) with a pseudo box-trajectory. The pseudo-box trajectory is defined by a continuous function, \( r_t(\cdot) \) describing the trajectory at every timestep as: \( T_{r_t} = (B^*_t, r_{t+1}, \ldots, r_{t+T}(B^*_t)) \), relative to the true keyframe location \( B^*_t \). Because the next keyframe is the last trajectory location, we also constrain the pseudo trajectory to match the true bounding box at the end of the trajectory: \( r_{t+T}(B^*_t) = B^*_{t+T} \).

In practice, we choose \( r_t(\cdot) \) to be either linearly interpolated box annotations, or a parabola as it is continuous and does not assume linear object trajectories. Here, we only need bounding-box annotations every \( T \) frames, for correcting the starting- and end-point of our predicted trajectory. Our sparsely-annotated loss \( L^{\text{trajectory}} \) variants are useful when the dataset only has sparse annotations available.

### 4. Experiments

**Datasets and evaluation setup.** We test our hypotheses on a fully controlled MovingDigits dataset. We ablate model choices on a subset of ImageNet VID [58]. We show the practical benefits of our approach on the full ImageNet VID [58] and on EPIC KITCHENS-55 [11]. As a realistic sparsely annotated scenario, we evaluate on YouTube-BoundingBoxes [36] which has approximately 1 keyframe annotation per second, and the Waymo Open dataset [15]. For the ImageNet VID experiments, we train our model on a combination of ImageNet VID and DET datasets as is common practice in [9, 24, 72, 80]. To quantify detection accuracy we adopt the common approach [7, 9, 70, 72] by computing mean Average Precision (mAP), where a detection is correct if its Intersection over Union (IoU) with the ground truth is sufficiently large. Note that although our method samples keyframes during training, we evaluate on all frames at test time, unless stated otherwise.

**Implementation details.** We test either using a Faster RCNN [57] detector or a Deformable DETR [79] with an ImageNet pre-trained ResNet [31] or SwinB [48] base as the object detection backbone. Our trajectory prediction sub-network contains three fully-connected layers with 1024 dimensions for the middle layer, see Fig. 2. We train our network for 120k iterations on ImageNet VID and 173k iterations on Epic Kitchens with the SGD optimizer, on 4 GPUs. For ImageNet VID, the initial learning rate is \( 10^{-3} \) and is divided by 10 at 80K iterations. For Epic Kitchens, the initial learning rate is \( 5 \times 10^{-4} \) and is divided by 10 at 120K iterations. For YouTube-BoundingBoxes, the initial learning rate is \( 5 \times 10^{-4} \) and is divided by 10 at 100K iterations.
We test our hypotheses by creating our own fully controlled dataset, MovingDigits, where we pair each of the 10 MNIST digit classes with a unique, linear motion of 2 px per frame, see Fig. 4. Each video has 32 frames with a frame size of 64×64 px. We created 200 videos for training and 80 videos for testing, with an equal number of videos per class. We use a trajectory length of $T=8$, train for 1.25k iterations and use a ResNet-18 feature extractor and a Faster RCNN detector. We cannot use the existing MovingMNIST [62] because it does not provide detection bounding boxes, and it does not contain motion-appearance correlations.

[H1]: Can the model anticipate motion trajectories? To verify if our model can anticipate trajectories from a single static input frame $t$, we train on our MovingDigits, where each digit has its own linear motion. We calculate the average IoU over the predicted trajectories for the test videos. The IoU is 0.95, which is near-perfect compared to the IoU of 0.79 for no motion anticipation. We show an example of the predicted bounding boxes (Bbox) by our method and the ground truth bounding boxes (GT Bbox) from time steps $t$ to $t+3$ in Fig. 4. Our model can successfully learn motion by anticipating trajectories from a static input frame.

[H2]: Anticipating improves static detection. The motion cues in-between the static keyframes offer an additional source of supervision. Here, we investigate how the motion anticipation affects the static object detector, when we evaluate at test-time only on keyframes. We consider four types of motion supervision for predicting box trajectories:

1. **Ground truth motion**: trained on true bounding-box trajectories annotated at every frame; 2. **Simulated smooth motion**: bounding boxes move between keyframes according to a smooth parabola. (Details and examples are in the supplementary material); 3. **Randomized positions**: there is motion, but it is not smooth, the boxes in-between the keyframes can occur at any random position in the image; 4. **No motion**: without motion prediction, i.e., the static object detector baseline trained at every video frame.

Tab. 1 shows the keyframe mAP scores for IoU$@[0.5:0.05:0.95]$. The No motion static object detector is the baseline, which uses no motion. The Randomized positions supervision is detrimental for object detection because the motion is random, and unpredictable. Interestingly, both Ground truth motion and Simulated smooth motion supervision improve the static keyframe detection. We speculate that the anticipation loss encourages detecting static regions that are most likely to move coherently, with consistent motion offsets: i.e. same direction. Thus, for non-random motions, adding motion anticipation as additional supervision improves static object detection at keyframes, even without knowing the ground truth motion between the keyframes.

<table>
<thead>
<tr>
<th>Motion</th>
<th>Keyframe mAP (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Randomized positions</td>
<td>62.68</td>
</tr>
<tr>
<td>No motion</td>
<td>73.51</td>
</tr>
<tr>
<td>Simulated smooth motion</td>
<td>76.57</td>
</tr>
<tr>
<td>Annotated motion</td>
<td>79.31</td>
</tr>
</tbody>
</table>

4.2. Ablation of model components

We run all ablation experiments on an ImageNet VID [58] subset containing the classes ‘dog’, ‘giant_panda’, and ‘hamster’. We use a ResNet-101 for feature extraction and Faster RCNN for detection.

[A1]: The effect of the trajectory loss. We evaluate each term in our trajectory loss $L_{traj} = L_{\Sigma} + L_{bag(\delta)}$ in Eq. (4), compared to the standard loss $L_{bag}$ in Eq. (1). The mAP scores for trajectory lengths $T=4$ and $T=8$ are in Tab. 2. The $L_{traj}$ loss has a higher mAP than the bag loss $L_{bag}$ for both trajectory lengths. $L_{traj}$ enforces continuity and smoothness in the predictions over time thus leads to more precise predictions. Furthermore, the improvement of $L_{traj}$ over $L_{bag}$ is larger for longer trajectories, e.g. for $T=4$ and $T=8$, $L_{traj}$ outperforms $L_{bag}$ by 1.66% and 2.18% respectively. Our $L_{traj}$ loss is defined using offsets, without offsets the $L_{\Sigma}$ would be equivalent to $L_{bag}$. We observe that the effect of predicting offsets between neighboring boxes $\delta_{t+t}$, instead of bounding-box coordinates $B_{t+t}$ gives...
Table 2. [A1]: Loss choice. Compared to $L_{\text{bag}}$, the $L_{\text{bag}(\delta)}$ loss does worse, whereas $L_{\Sigma}$ performs on par. Their combination in $L_{\text{traj}} = L_{\Sigma} + L_{\text{bag}(\delta)}$ outperforms $L_{\text{bag}}$. Predicting offsets instead of bounding-box coordinates in the trajectory network gives better results. These patterns are consistent over both trajectory lengths.

<table>
<thead>
<tr>
<th>Trajectory length (mAP %)</th>
<th>T=4</th>
<th>T=8</th>
</tr>
</thead>
<tbody>
<tr>
<td>$L_{\text{bag}}$</td>
<td>87.54 ± 0.16</td>
<td>83.97 ± 0.71</td>
</tr>
<tr>
<td>$L_{\Sigma}$</td>
<td>87.95 ± 0.27</td>
<td>84.09 ± 0.67</td>
</tr>
<tr>
<td>$L_{\text{bag}(\delta)}$</td>
<td>85.19 ± 0.66</td>
<td>79.73 ± 1.01</td>
</tr>
<tr>
<td>$L_{\text{traj}}$</td>
<td>89.20 ± 0.21</td>
<td>86.15 ± 0.75</td>
</tr>
</tbody>
</table>

Table 3. [A2]: Sparse annotation loss analysis. We sub-sample keyframes of the fully annotated ImageNet VID subset to mimic the sparse annotations, and evaluate our sparse annotation loss on keyframe detection. The $L_{\text{bag}(\delta)}$ does not improve the detection accuracy, while $L_{\Sigma}$ performs on par with the $L_{\text{traj}}$. The sparse annotation loss achieves a comparable accuracy on keyframe detection with our fully-supervised loss $L_{\text{traj}}$.

<table>
<thead>
<tr>
<th>Keyframe mAP%</th>
<th>$L_{\text{traj}}$</th>
<th>Sparse annotation loss</th>
</tr>
</thead>
<tbody>
<tr>
<td>$L_{\Sigma}$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$L_{\text{bag}(\delta)}$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$L_{\text{traj}}$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>T=4</th>
<th>T=8</th>
</tr>
</thead>
<tbody>
<tr>
<td>$L_{\Sigma}$</td>
<td>91.03 ± 0.19</td>
<td>90.35 ± 0.34</td>
</tr>
<tr>
<td>$L_{\text{bag}(\delta)}$</td>
<td>86.38 ± 0.66</td>
<td>86.38 ± 0.73</td>
</tr>
<tr>
<td>$L_{\text{traj}}$</td>
<td>87.94 ± 0.66</td>
<td>87.12 ± 0.69</td>
</tr>
</tbody>
</table>

Figure 5. [A3] Inference speed vs. accuracy trade-off. We can increase inference speed (FPS) by sampling fewer keyframes and predicting longer trajectories. Yet, by increasing trajectory length the mAP decreases. The trajectory length can be selected according to the needed speed-accuracy trade-off.

4.3. Comparison with state-of-the-art

[C1]: Experiments on ImageNet VID. We compare our method with state-of-the-art video object detection methods on ImageNet VID in Tab. 4. We mark methods without post-processing with ✓. Methods with post-processing add extra computational cost. Our method does not use any post-processing. We use a prediction trajectory of length $T = 4$, as this already gives a considerable reduction in computation speed. For our model, we test using a ResNet-101 [31] and a SwinB [48] backbone with either a Faster RCNN [57] or a Deformable DETR [79].

Among all methods using a ResNet-101 backbone, our method is the most accurate with a 87.2% mAP, which has a 2.7% improvement over the leading MEGA [7]. We also report runtime (FPS) and train-time (hrs/epoch) to show that our method is fast and efficient. We measure the efficiency using the code provided by the original papers, and where the code is not available we mark this with ‘-‘. Note that for the training time this is a rough estimate, when considering the same settings (batch-size, GPU) for all methods. We tested the inference runtime speed on a single NVIDIA GTX 1080 Ti. In terms of both training time and runtime our method is most efficient. Our method has fast inference time with 39.6 FPS, which is $\approx 1.8$ times faster than existing fast methods like LWND [36] and ST-Lattice [6]. Our effi-
Table 4. [C1]: Experiments on ImageNet VID. We indicate the methods without video-level post-processing with a ✓. ‘No Post-proc.’ means no post-processing. R101 here is ResNet-101. The runtime is measured on a NVIDIA GTX 1080 Ti. Our method has the best performance and fastest runtime among all the methods using two-stage detectors (e.g. Faster RCNN). With a stronger backbone SwinB [48], our method exceeds state-of-the-art.

Table 5. [C1]: ImageNet VID across different motion speeds. Our method improves mAP on different motion speeds.

Table 6. [C2]: Experiments on EPIC KITCHENS-55. S1 and S2 represent Seen and Unseen splits, respectively. Our method achieves promising results for both test sets and IoU thresholds.

4.4. Sparsely annotated videos

[S1] Sparsely annotated YouTube-BoundingBoxes. The YouTube-BoundingBoxes dataset [56] has sparse annotations: the video frame rate is 30 fps, and on average it only has annotations at 1 fps. We compare to the Faster RCNN [57] on YouTube-BoundingBoxes in Tab. 7, when using a Faster RCNN detector in our method. The inputs are keyframes sampled with a step 60 and 30 frames in the video, respectively. During training, for Faster RCNN we use the labels at every 60 frames and 30 frames in the video, respectively. For evaluation, 106 sequences collected in the same 28 kitchens (S1) and 54 sequences collected in 4 other unseen kitchens (S2) are used. We use a prediction trajectory length of $T=4$ and evaluate for two IoU thresholds of 0.5 and 0.75. As summarized in Tab. 6, our method is more accurate than previous state-of-the-art methods for both Seen/Unseen splits. Our method is applicable to complex video detection tasks.
Table 7. [S1] Sparsely annotated YouTube-BoundingBoxes. mAP on the sparsely annotated YouTube-BoundingBoxes. The sparse annotation loss $L_{sa}^{traj}$ using interpolated pseudo-trajectory labels improves keyframe detection.

<table>
<thead>
<tr>
<th>Methods</th>
<th>Keyframe step</th>
<th>Label step</th>
<th>mAP (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Faster RCNN [57]</td>
<td>60</td>
<td>60</td>
<td>47.6</td>
</tr>
<tr>
<td>Faster RCNN [57]</td>
<td>30</td>
<td>30</td>
<td>58.7</td>
</tr>
<tr>
<td>Ours $L_{sa}^{traj}$</td>
<td>60</td>
<td>30</td>
<td>51.3</td>
</tr>
<tr>
<td>Ours $L_{sa}^{traj}$</td>
<td>30</td>
<td>1</td>
<td>59.8</td>
</tr>
</tbody>
</table>

Table 8. [S2]: Experiments on the Waymo Open dataset. We report average precision over the $L_1$ and $L_2$ instance difficulty levels. Even when learning from object annotations at 1 fps, our method using interpolated pseudo-trajectories outperforms the Faster RCNN baseline.

<table>
<thead>
<tr>
<th>Methods</th>
<th>AP/L1 (%)</th>
<th>AP/L2 (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Faster RCNN</td>
<td>55.66</td>
<td>49.63</td>
</tr>
<tr>
<td>Ours $L_{sa}^{traj}$</td>
<td>64.53</td>
<td>59.28</td>
</tr>
</tbody>
</table>

4.5. Method limitations

Despite our method’s successful predictions as in Fig. 6(a), we also identify several limitations. In practice, multiple motion patterns can be associated with the same appearance: e.g. people can walk or jump. In this case, our method may fail to predict the correct object locations. Another failure case is if objects appear or disappear in the middle of a trajectory. In these cases, we will either miss the objects or over-predict. Yet another limitation of our method is the assumption that the motion changes smoothly. If the object trajectories have large variations in a short time because of the video frame rate, our trajectory network may not be able to learn this. One such example is the dog in Fig. 6(b) which suddenly changes its moving direction. Even if we miss some detections of intermediate frames, as shown in Fig. 6(b), our method can recover at the next keyframe detection. While these limitations exist, they only influence our method minimally, since we miss only a few hundred milliseconds, which is reflected in our state-of-the-art video object detection results.

5. Conclusion

We propose a method to efficiently detect objects in videos by predicting their future locations from a static input keyframe and the ground truth locations of all frames. Our method associates appearance with motion. Different motion contexts have different appearances, thus we can model various motion patterns from the keyframes with different appearances. Because we predict the future object locations over multiple frames, we do not need to process every frame of the video, but only a subset of the keyframes, which makes our method efficient. Moreover, by learning to predict object trajectories we improve the object detection accuracy when compared to the state-of-the-art on multiple datasets. Finally, by using pseudo object trajectories defined by smooth continuous functions, we can improve object detection accuracy on sparsely annotated videos.

Acknowledgments. This work is part of the research program Efficient Deep Learning (EDL), which is partly financed by the Dutch Research Council (NWO). Fatemeh Karimi Nejadasl is financed by the University of Amsterdam Data Science Centre.
References


the IEEE/CVF International Conference on Computer Vision (ICCV), pages 2344–2352, 2017. 2
[75] Zhujun Xu, Emir Hrustic, and Damien Vivet. Centernet heatmap propagation for real-time video object detection. In


