Action-Driven Visual Object Tracking With Deep Reinforcement Learning

Sangdoo Yun, *Student Member, IEEE*, Jongwon Choi, *Student Member, IEEE*, Youngjoon Yoo, *Student Member, IEEE*, Kimin Yun, and Jin Young Choi, *Member, IEEE*

***Abstract*— In this paper, we propose an efficient visual tracker, which directly captures a bounding box containing the target object in a video by means of sequential actions learned using deep neural networks. The proposed deep neural network to control tracking actions is pretrained using various training video sequences and fine-tuned during actual tracking for online adaptation to a change of target and background. The pretraining is done by utilizing deep reinforcement learning (RL) as well as supervised learning. The use of RL enables even partially labeled data to be successfully utilized for semisupervised learning. Through the evaluation of the object tracking benchmark data set, the proposed tracker is validated to achieve a competitive performance at three times the speed of existing deep network-based trackers. The fast version of the proposed method, which operates in real time on graphics processing unit, outperforms the state-of-the-art real-time trackers with an accuracy improvement of more than 8%.**

***Index Terms*— Deep neural network, reinforcement learning (RL), visual tracking.**

I. Introduction

THE aim of visual object tracking is to find a bounding box tightly containing the target moving object in every frame of a video, which is one of the fundamental problems in the computer vision field. In recent decades, there have been many advances in visual tracking algorithms, but there are still many challenging issues arising from diverse tracking obstacles, such as motion blur, occlusion, illumination change, and background clutter. In particular, conventional tracking methods [1]–[9] using low-level hand-crafted features encounter the above-mentioned tracking obstacles because of their insufficient feature representation.

Manuscript received April 11, 2017; revised August 29, 2017, December 11, 2017, and January 9, 2018; accepted January 30, 2018. Date of publication March 2, 2018; date of current version May 15, 2018. This work was supported in part by the ICT R&D program of MSIP/IITP (Development of Predictive Visual Intelligence Technology) under Grant B0101-15-0552 and (Development of High Performance Visual BigData Discovery Platform) under Grant B0101-15-0266, in part by the SNU-Samsung Smart Campus Research Center at Seoul National University, and in part by the Brain Korea 21 Plus Project. *(Corresponding author: Jin Young Choi.)*

S. Yun, J. Choi, and J. Y. Choi are with the Department of Electrical and Computer Engineering, Automation and Systems Research Institute, Seoul National University, Seoul 08826, South Korea (e-mail: jychoi@snu.ac.kr).

Y. Yoo is with the CLOVA AI Research, Naver, Seoungnam-si 13561, South Korea.

K. Yun is with Visual Intelligence Research Group, SW Contents Laboratory, Electronics and Telecommunications Research Institute, Daejeon 34129, South Korea.

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TNNLS.2018.2801826

Recently, tracking methods [10]–[12] using convolutional neural networks (CNNs) have been proposed for robust tracking and vastly improved tracking performance with the help of rich feature representation by deep hidden layers. The initial works [10], [11] utilize pretrained CNNs, which are trained on a large-scale classification data set, such as ImageNet [13]. However, a CNN pretrained on a classification data set is not sufficient to solve the problem of adaptation to object shape deformation and illumination changes in tracking, because the deep CNN is not appropriate for online adaptation. An existing state-of-the-art algorithm by Nam and Han [12] adopts a tracking-by-detection approach using CNNs trained with plenty of video data sets, such as [14] and [15]. In addition, it improves the ability to distinguish between the target and the background using subnetworks that learn the discriminative features of the target and the background via online adaptation. This CNN-based tracking-by-detection approach achieves a breakthrough in tracking performance, but it suffers from an inefficient exhaustive search strategy that explores the region of interest and selects the best candidate by referring to the scores obtained by the network. To overcome this exhaustive search problem with the tracking-by-detection methods, we introduce an action-driven tracking mechanism that actively pursues the target movement considering the context change of the image within the bounding box.

In addition, there is a critical problem with constructing training data for a deep CNN-based tracker. Deep CNN-based trackers require a large amount of training data in order to learn convolutional features from scratch. Even though there are plenty of video sequences, it is extremely expensive to annotate the target position in every frame for the construction of training data. If we can train a deep CNN-based tracker using a partially labeled video sequence, a variety of videos can be utilized with less effort for the training. However, the existing deep CNN-based trackers [10]–[12] adopt a supervised learning (SL) scheme and so have difficulties with utilizing the partially labeled video sequences. In this paper, in contrast to the existing deep CNN-based trackers using SL, we try to develop a reinforcement learning (RL) scheme to utilize the partially labeled video sequences effectively for training our action-driven deep tracker.

The proposed action-driven deep tracker pursues a change of target by repetitive actions controlled by an action-decision network (ADNet) consisting of deep CNN architecture. We cast the visual tracking problem as selecting the sequential

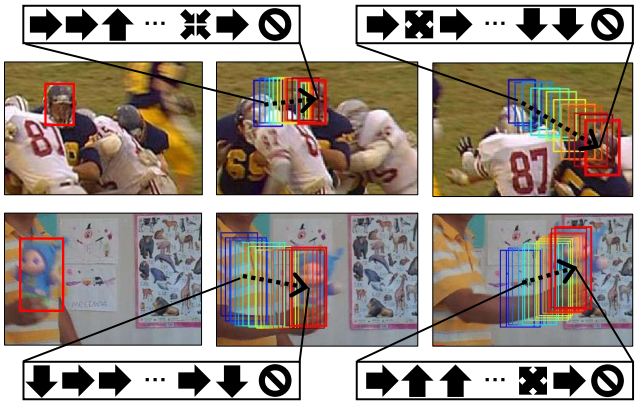

Fig. 1. Concept of the proposed visual tracking controlled by sequential actions. The first column shows the initial location of the target, and the second and third columns show the iterative action flow to find the target bounding box in each frame. The sequential actions selected by the proposed method control the tracker to iteratively move the initial bounding box (blue) to the target bounding box (red) in each frame.

actions and suggest a training method using RL. The basic concept of the proposed visual tracking is shown in Fig. 1. The ADNet is designed to generate actions to find the location and the size of the target object in a new frame. The ADNet learns the policy that selects the optimal actions to track the target from the state of its current position. In the ADNet, the policy network is designed with a CNN [16], in which the input is an image patch cropped at the position of the previous state and the output is the probability distribution of actions, including translation and scale changes. This action-selecting process has fewer searching steps than sliding window or candidate sampling approaches [12], [17]. In addition, since our method can precisely localize the target by selecting actions, postprocessing, such as bounding box regression [12], is not necessary.

We also propose a learning algorithm with deep RL to train the ADNet. The whole training framework is composed of an SL stage and an RL stage. In the SL stage, we train our network to select actions to track the position of the target using samples extracted from training videos. In this step, the network learns to track general objects without sequential information. In the RL stage, the trained network in the SL stage is used as an initial network. We obtain training samples for RL by performing tracking simulation on training sequences. The network is trained with deep RL based on a policy gradient [18], using the rewards obtained during the tracking simulation. Even in the case where training frames are partially labeled [semisupervised (SS) case], the proposed framework successfully learns the unlabeled frames by assigning the rewards according to the results of tracking simulation.

A preliminary version of this paper was presented as a spotlight in CVPR2017 [19], which is attached to a supplementary material for this paper. The prior work initially suggests the tracking mechanism by selecting sequential actions and the network architecture. In this paper, we supplement the related works by adding action-driven methods adopted in computer vision. In the text, additional derivations and algorithms have been added to explain the training scheme of RL and online adaptation. While the preliminary version evaluated the proposed tracker in general situations, the extended work includes evaluations with the various attributes of tracking scenes. In addition, the additional experiments and the analysis have been added to rigorously validate the proposed action-driven approach. We investigate the effect of the movement size of the action on tracking performance and speed. We analyze the action dynamic factors in the fc6 layer to examine the impact of the past actions. For self-evaluation, two additional variants of the ADNet have been conducted and discussed: 1) the policy gradient method is replaced with a value-function-based method and 2) the reward function is replaced with a continuous one.

The main contributions of this paper are summarized as follows.

1) The action-driven deep tracker is proposed for the first time to dynamically track the target object by pursuing actions instead of tracking-by-detection scheme.
2) We cast the visual tracking problem as a Markov decision process (MDP) and design a deep network architecture to realize the decision process.
3) Deep RL algorithm is developed to train the ADNet with partially labeled data in the SS setting.
4) The proposed deep tracker can control the tradeoff between tracking performance and computational complexity by simply changing the metaparameter in tracking.
5) The proposed tracker achieves a state-of-the-art performance with much more efficient searching complexity than the existing deep network-based trackers using a tracking-by-detection strategy. Also, the fast version of the proposed method operates in real time on graphics processing units (GPUs), outperforming the state-of-the-art real-time trackers.

This paper is organized as follows. We first review the previous works related to the proposed tracking method in Section II. We then introduce the action-driven tracking scheme in Section III and describe the proposed training method in Section IV. We evaluate the proposed tracker with visual tracking benchmarks in Section V. Finally, we conclude this paper in Section VI.

II. Related Work

A. Visual Object Tracking

As surveyed in [20] and [21], various trackers have shown their performance and effectiveness on various tracking benchmarks [14], [15], [22]. The approach based on tracking-by-detection [1]–[4] aims to build a discriminative classifier that distinguishes the target from the surrounding background. Typically these methods capture the target position by detecting the most matching position using the classifier. Online boosting methods [1], [2] were proposed to update the discriminative model in online manner. Multiple instance learning [3] and tracking–learning–detection [4] methods were proposed to update the tracking model robust to a noise.

Tracking methods based on a correlation filter [5]–[9] have attracted attention due to their computational efficiency and competitive performance. This approach learns the correlation filter in a Fourier domain with low computational

load. Bolme *et al.* [5] proposed a minimum output sum of squared error filter and Henriques *et al.* [7] proposed kernelized correlation filters (KCFs) with multichannel features. Hong *et al.* [8] proposed the combined system using short-term correlation tracker and long-term memory stores. Choi *et al.* [9], [23] proposed the integrated system of the various types of correlation filters with attentional weighted map. To overcome the insufficient representation of the handcrafted features, deep convolutional features were utilized in the correlation filters [24], [25], which achieved the state-of-the-art performance. However, these methods require heavy computational load, because there is a need to train multiple scalewise filters using deep convolutional features.

Recently, CNN-based methods [10]–[12], [17], [26]–[30] have been proposed to learn the tracking models. Early attempts [26]–[28] were suffering from the data deficiency problem for training their networks. To solve the insufficient data problem, transferring methods [10], [11] were proposed by utilizing the pretrained CNNs on a large-scale classification data set, such as ImageNet [13]. However, these methods still have a limitation in adaptation to object shape deformation and illumination changes in tracking, because the deep CNN is not appropriate for online adaptation. Recently, the proposed CNN-based methods [12], [17], [30] improved the tracking performance using a large amount of tracking video data sets [14], [15], [21]. In particular, Held *et al.* [30] first utilized deep regression networks to capture a precise position of the target. However, these methods also had no online updating procedure and so had difficulties in tracking an occluded or quickly moving target.

Tao *et al.* [17] and Nam and Han [12] made a breakthrough by adopting tracking-by-detection approaches and adding small-sized networks for online adaptation. However, these methods [12], [17] need computationally inefficient search algorithms, such as sliding window or candidate sampling.

B. Action-Driven Approaches

Action-driven has been actively applied to various computer vision applications [31]–[36]. Mnih *et al.* [31] proposed an attentional method to localize MNIST digit in an image using recurrent neural network. Gonzalez-Garcia *et al.* [32] proposed an active strategy to choose a search window for object detection using an image context. Yoo *et al.* [33] proposed a deep CNN framework (AttentionNet) to capture the object by sequential actions with top-down attention. AttentionNet has achieved satisfactory performance on object detection benchmark, PASCAL VOC 2007 and 2012 [37], by sequentially refining the bounding boxes. Mathe *et al.* [35] proposed a sequential search strategy to detect visual objects in images, where the detection model was trained by RL. Caicedo and Lazebnik [34] proposed a deep RL framework to select a proper action to capture an object in an image. Xiang *et al.* [36] tackled a multiple object tracking problem by formulating a decision-making problem. However, this method learns a decision process to switch the status of a tracker, such as "tracking success" or "tracking failure," which is different from our scheme learning the tracking actions depending on the context of the image.

C. Deep Reinforcement Learning

The goal of RL is to learn a policy that decides sequential actions by maximizing the cumulative future rewards [38]. A Recent trend [39]–[43] in RL field is to utilize the deep neural networks as a function approximation of policy or value function. By resorting to the use of deep features, many difficult problems, such as playing Atari games [39], [40] or Go [43], can be successfully solved in a SS setting.

There are two popular approaches in deep RL algorithms: deep Q networks (DQN) and policy gradient. DQN [39], [40] is a form of Q-learning with function approximation using deep neural networks. The goal of DQN is to learn a state-action value function (Q), which is given by the deep networks, by minimizing temporal-difference errors [39], [40]. Based on the DQN algorithm, various network architectures, such as double DQN [41] and double deep Q-network [44], were proposed to improve performance and keep stability. Gu *et al.* [45] proposed a continuous deep Q-learning with model-based acceleration. Policy gradient methods directly learn the policy by optimizing the deep policy networks with respect to the expected future reward using gradient descent. Williams [18] proposed the REINFORCE algorithm, by simply using the immediate reward to estimate the value of the policy. Silver *et al.* [42] proposed a deterministic algorithm to improve the performance and effectiveness of the policy gradient in high-dimensional action space. Lillicrap *et al.* proposed deep deterministic policy gradient [46] to apply policy gradient in continuous action space. In the work of Silver *et al.* [43], it is shown that pretraining the policy networks with SL before using policy gradient can improve the performance.

The deep RL methodology has been applied to other computer vision applications, such as object localization [32], [35] and action recognition [47]. In visual tracking field, Zhang *et al.* [48] have attempted to utilize a deep RL approach for tracking the target object in video sequences. This method focuses on localizing the target by a regression frame by frame; therefore, it does not consider sequential dynamics of the target objects and cannot utilize the weekly SL strategy, which are distinctive aspects in our method. In this paper, we adopt the MDP to consider both semantics and dynamics of the target object, where the action dynamics of the tracking target can be learned with the proposed RL using partially labeled sequential data.

III. Tracking Scheme Controlled by Actions

A. Overview

Visual tracking solves the problem of finding the position of the target in a new frame from the current position. The proposed tracker dynamically pursues the target by sequential actions controlled by the ADNets shown in Fig. 2 (details are in Section III-B). The ADnet predicts the action to chase the target moving from the position in the previous frame. The bounding box is moved by the predicted action from the previous position, and then, the next action is sequentially predicted from the moved position. By repeating this process over the test sequence, we solve the object tracking problem. The ADNet is pretrained by SL (see Section IV-A) as well

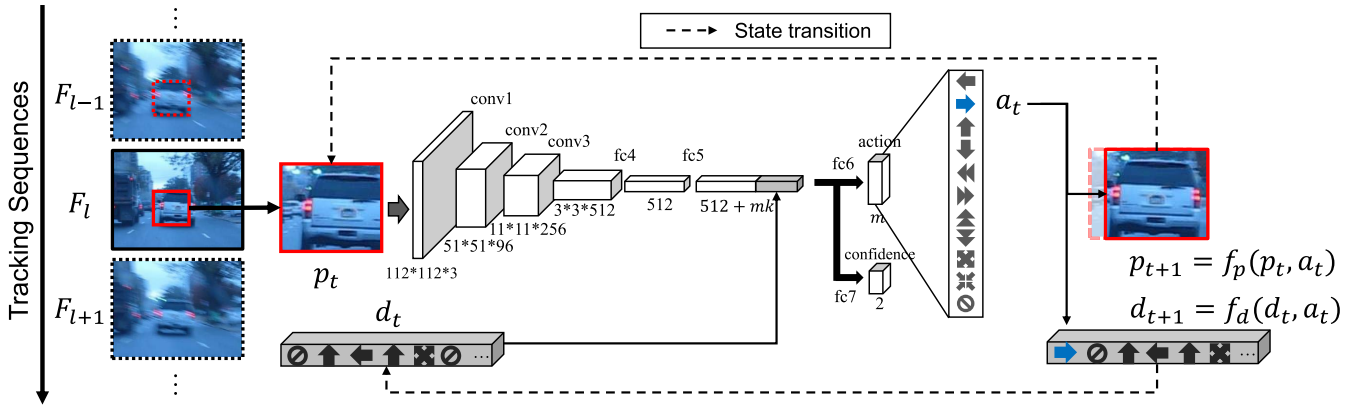

Fig. 2. Architecture of the proposed tracker using ADNet tracker. The dashed lines indicate the state transition. In this example, the "move right" action is selected to capture the target object. This action-decision process is repeated until finalizing the location of the target in each frame.

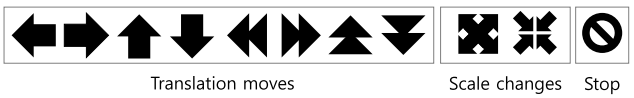

Fig. 3. Defined actions in our method.

as RL (see Section IV-B). During actual tracking, online adaptation (see Section IV-C) is conducted.

B. Problem Settings

Basically, our tracking strategy follows the MDP. The MDP is defined by states $s \in \mathcal{S}$, actions $a \in \mathcal{A}$, state transition function $s' = f(s, a)$, and the reward $r(s, a)$. In our MDP formulation, ADNet tracker is defined as the tracking agent of which the goal is to capture the target with a bounding box shape. The action is defined in a discrete space, and a sequence of actions and states is used to iteratively pursue the resulting location and size of bounding box in each frame.

In every frame, ADNet tracker decides sequential actions until finalizing the target's position, and then goes to the next frame. The state representation includes the shape information of the image within the bounding box and the previous actions. An ADNet tracker receives a reward for the final state of the frame l by deciding whether the ADNet tracker succeeds to track the object or not. State and action are denoted by $s_{t,l}$ and $a_{t,l}$, respectively, for $t = 1, \ldots, T_l$ and $l = 1, \ldots, L$ where T_l is the terminal step at frame l and L denotes the number of frames in a video. The terminal state in the lth frame is transferred to the next frame, i.e., $s_{1,l+1} := s_{T_l,l}$. In the following except Sections IV-B and IV-C, we omit the subscript l when we describe MDP in each frame for simplicity.

1) Action: The action space $\mathcal{A}$ consists of 11 types of actions, including translation moves, scale changes, and stopping action, as shown in Fig. 3. The translation moves include four directional moves: {*left*, *right*, *up*, *down*} and also have their two times larger moves. The scale changes are defined as two types: {*scale up*, *scale down*}, which maintain the aspect ratio of the tracking target. Each action is encoded by the 11-D vector with one-hot form.

2) State: The state s_t is defined as a tuple (p_t, d_t), where $p_t \in \mathbb{R}^{112\times112\times3}$ denotes the image patch within the bounding box (we call simply "patch" in the following) and $d_t \in \mathbb{R}^{11k}$ represents the dynamics of actions denoted by a vector (called by "action dynamics vector" in the following) containing the previous k actions at tth iteration. The patch p_t is pointed by 4-D vector $b_t = [x^{(t)}, y^{(t)}, w^{(t)}, h^{(t)}]$, where $(x^{(t)}, y^{(t)})$ denotes the center position and $w^{(t)}$ and $h^{(t)}$ denote the width and height of the tracking box, respectively. In a frame image F, the patch p_t at iteration t is defined as

$$p_t = \phi(b_t, F) \tag{1}$$

where ϕ denotes the preprocessing function, which crops the patch p_t from F at $b_t \in \mathbb{R}^4$ and resizes it to match the input size of our network. The action dynamics vector d_t is defined as a vector concatenating the past k actions at iteration t. We store past k actions in the action dynamics vector $d_t = [\psi(a_{t-1}), \ldots, \psi(a_{t-k})]$, where $\psi(\cdot)$ denotes one-hot encoding function. Letting $k = 10$, d_t has 110 dimensions, since each action vector has 11 dimensions.

3) State Transition Function: After decision of action a_t in state s_t, the next state s_{t+1} is obtained by the state transition functions: patch transition function $f_p(\cdot)$ and action dynamics function $f_d(\cdot)$. The patch transition function is defined by $b_{t+1} = f_p(b_t, a_t)$, which moves the position of the patch by the corresponding action. The discrete amount of movements is defined as

$$\Delta x^{(t)} = \alpha w^{(t)} \text{ and } \Delta y^{(t)} = \alpha h^{(t)} \tag{2}$$

where α is set to 0.03 via empirical investigation in our experiments. For example, if *"left"* action is selected, the position of the patch b_{t+1} moves to $[x^{(t)} - \Delta x^{(t)}, y^{(t)}, w^{(t)}, h^{(t)}]$, and *"scale up"* action changes the size into $[x^{(t)}, y^{(t)}, w^{(t)} + \Delta x^{(t)}, h^{(t)} + \Delta y^{(t)}]$. The other actions are defined in a similar manner. The action dynamics function is defined by $d_{t+1} = f_d(d_t, a_t)$, which updates the action history in the action dynamics vector. When the *"stop"* action is selected, we finalize the patch position for the target in the current frame, the ADNet tracker will receive the reward, and then,

the resulting state is transferred to the initial state of the next frame.

4) Reward: The reward function is defined as $r(s)$, since the ADNet tracker obtains the reward by the state s regardless of the action a. The reward $r(s_t)$ keeps zero during iteration in MDP in a frame. At the termination step T, that is, a_T is "stop" action, $r(s_T)$ is assigned by

$$r(s_T) = \begin{cases} 1, & \text{if IoU}(b_T, G) > 0.7 \\ -1, & \text{otherwise} \end{cases} \tag{3}$$

where $\text{IoU}(b_T, G)$ denotes overlap ratio of the terminal patch position b_T and the ground truth G of the target with intersection-over-union (IOU) criterion. The tracking score z_t is defined as the terminal reward $z_t = r(s_T)$, which will be used to update ADNet by RL.

C. Action-Decision Network

The pretrained Vision Geometry Group Medium (VGG-M) model [16] is used to initialize ADNet. Small CNN models, such as VGG-M [16], are more effective in the visual tracking problem than deep models [12]. As shown in Fig. 2, ADNet has three convolutional layers {conv1, conv2, conv3}, which are identical to the convolutional layers of VGG-M network. The next two fully connected layers {fc4, fc5} are combined with the rectified linear unit (ReLU) and dropout layers, and each has 512 output nodes. The output of fc5 layer is concatenated with the action dynamics vector d_t, which has 110 dimensions. The final layers {fc6, fc7} predict the action probabilities and confidence score of the given state, respectively. The parameter set of the ith layer is denoted by w_i, and the whole network parameter set is denoted by W.

The fc6 layer has 11 output units and is combined with softmax layer, which represents the conditional action probability distribution $p(a|s_t; W)$ for the given state. The probability $p(a|s_t; W)$ means the probability of selecting action a in the state s_t. As shown in Fig. 2, ADNet iteratively pursues the target position. ADNet tracker selects actions sequentially and updates states until finalizing the position of target. The final state is obtained by selecting stop action or falling in the oscillation case. The oscillation case is detected when the bounding box is coming back to the previous state, for example, the sequential actions as {*left*, *right*, *left*} lead to oscillation. The confidence layer (fc7) has two output units; one produces the probability of the target, and the other yields the probability of background for the given state s_t. The target probability $p(\text{target}|s_t; W)$ is used as the confidence score of the action at s_t. The confidence score is utilized for the online adaptation during tracking (see Section IV-C).

IV. Training of Action-Decision Network

In this section, we describe the training scheme for ADNet. First, in off-line manner, ADNet is pretrained by SL (see Section IV-A) and RL (see Section IV-B) using the training videos to learn the ability of tracking general objects. In SL, ADNet is trained to predict a proper action for a given state. In RL, ADNet is updated through tracking simulation on the training sequence and utilizing action dynamics. During actual tracking in test video, online adaptation (see Section IV-C) is done for ADNet to adapt itself to the changes of surroundings or the deformation of the target.

A. Training ADNet With Supervised Learning

In the SL stage, the network parameters W_{SL}, $\{w_1, \ldots, w_7\}$, are trained. We first need to generate the training samples to train ADNet (W_{SL}). The training samples consist of image patches $\{p_j\}$, action labels $\{o_j^{(\text{act})}\}$, and confidence score labels $\{o_j^{(\text{con})}\}$. In this stage, the action dynamics is not considered, and we set the elements of the action dynamics vector d_j to zero. The training data sets provide video frames and the ground-truth patches within bounding boxes. A sample patch p_j is obtained by adding Gaussian noise to the ground truth, and the corresponding action label $o_j^{(\text{act})}$ is assigned by

$$o_j^{(\text{act})} = \arg\max_a \text{IoU}(\bar{f}(p_j, a), G) \tag{4}$$

where G is the ground-truth patch and $\bar{f}(p, a)$ denotes the moved patch from p by the action a. The confidence score labels $o_j^{(\text{con})}$ corresponding to p_j are defined as the following two labels:

$$o_j^{(\text{con})} = \begin{cases} 1, & \text{if IoU}(p_j, G) > 0.7 \\ 0, & \text{otherwise.} \end{cases} \tag{5}$$

A training batch has a set of randomly selected training samples $\{(p_j, o_j^{(\text{act})}, o_j^{(\text{con})})\}_{j=1}^{m}$. ADNet ($W_{\text{SL}}$) is trained by minimizing the multitask loss function by stochastic gradient descent. The multitask loss function is defined by minimizing the loss L_{SL} as follows:

$$L_{\text{SL}} = \frac{1}{m}\sum_{j=1}^{m} L\left(o_j{}^{(\text{act})}, \hat{o_j}{}^{(\text{act})}\right) + \frac{1}{m}\sum_{i=j}^{m} L\left(o_j{}^{(\text{con})}, \hat{o_j}{}^{(\text{con})}\right) \tag{6}$$

where m denotes the batch size, L denotes the cross-entropy loss, and $\hat{o_j}{}^{(\text{act})}$ and $\hat{o_j}{}^{(\text{con})}$ denote the predicted action and confidence score by ADNet, respectively.

B. Training ADNet With Reinforcement Learning

In the RL stage, network parameters W_{RL} ($\{w_1, \ldots, w_6\}$) except fc7 layer are trained. Training ADNet with RL in this section aims to improve the network by a policy gradient approach [18]. The initial RL network W_{RL} has the same parameters of the network trained by SL (W_{SL}). The action dynamics d_t is updated in every iteration by accumulating the recent k actions and shifting them in first-in-first-out strategy. Since the purpose of RL is to learn the state-action policy, we ignore the confidence layer fc7, which is needed in tracking phase.

The detailed algorithm to train ADNet with RL is described in Algorithm 1. During the training iterations, we first randomly pick a piece of training sequence $\{F_l\}_{l=1}^{\mathcal{L}}$ and the ground truths $\{G_l\}_{l=1}^{\mathcal{L}}$ (line 3 in Algorithm 1). We then perform the RL via tracking simulation with the training image sequences annotated by ground truth (lines 4–9 in Algorithm 1). In frame l, the procedure Tracker in Algorithm 2 performs the tracking simulation to pursue the

Algorithm 1 Training ADNet With RL

Input: Pre-trained ADNet (W_{SL}), Training sequences $\{F_l\}$ and ground truths $\{G_l\}$
Output: Trained ADNet weights W_{RL}
1: Initialize W_{RL} with W_{SL}
2: **repeat**
3: Randomly select $\{F_l\}_{l=1}^{\mathcal{L}}$ and $\{G_l\}_{l=1}^{\mathcal{L}}$
4: Set initial $b_{1,1} \leftarrow G_1$
5: Set initial $d_{1,1}$ as zero vector
6: $T_1 \leftarrow 1$
7: **for** $l = 2$ to $\mathcal{L}$ **do**
8: $\{a_{t,l}\}, \{b_{t,l}\}, \{d_{t,l}\}, T_l \leftarrow$ TRACKER$(b_{T_{l-1},l-1}, d_{T_{l-1},l-1}, F_l)$ in Algorithm 2
9: **end for**
10: Compute tracking scores $\{z_{t,l}\}$ with $\{b_{t,l}\}$ and $\{G_l\}_{l=1}^{\mathcal{L}}$
11: $\Delta W_{RL} \propto \sum_{l=1}^{\mathcal{L}} \sum_{t=1}^{T_l} \nabla_{W_{RL}} \log p(a_{t,l}|s_{t,l}; W_{RL}) z_{t,l}$ by Eq. (12)
12: Update W_{RL} using ΔW_{RL}
13: **until** W_{RL} converges

Algorithm 2 Tracking Procedure of ADNet

1: **procedure** TRACKER$(b_{T_{l-1},l-1}, d_{T_{l-1},l-1}, F_l)$
2: $t \leftarrow 1$
3: $p_{t,l} \leftarrow \phi(b_{T_{l-1},l-1}, F_l)$
4: $d_{t,l} \leftarrow d_{T_{l-1},l-1}$
5: $s_{t,l} \leftarrow (p_{t,l}, d_{t,l})$
6: **repeat**
7: $a_{t,l} \leftarrow arg\max_a p(a|s_{t,l}; W)$
8: $b_{t+1,l} \leftarrow f_p(b_{t,l}, a_{t,l})$
9: $p_{t+1,l} \leftarrow \phi(b_{t+1,l}, F_l)$
10: $d_{t+1,l} \leftarrow f_d(d_{t,l}, a_{t,l})$
11: $s_{t+1,l} \leftarrow (p_{t+1,l}, d_{t+1,l})$
12: $t \leftarrow t + 1$
13: **until** $s_{t,l}$ is a terminal state
14: Set termination step $T_l \leftarrow t$
15: Return $\{a_{t,l}\}, \{b_{t,l}\}, \{d_{t,l}\}, T_l$
16: **end procedure**

position of the target by selecting actions and updating the states in an iterative way from the initial state $b_{T_{l-1},l-1}$ and $d_{T_{l-1},l-1}$. A tracking simulation can generate a set of sequential states $\{s_{t,l}\}$, the corresponding actions $\{a_{t,l}\}$, and the rewards $\{r(s_{t,l})\}$ for the time steps $t = 1, \ldots, T_l$ and frame indices $l = 1, \ldots, \mathcal{L}$. The action $a_{t,l}$ for the state $s_{t,l}$ is assigned by

$$a_{t,l} = \arg\max_a p(a|s_{t,l}; W_{\mathrm{RL}}) \tag{7}$$

where $p(a|s_{t,l}; W_{\mathrm{RL}})$ denotes the probability distribution of possible actions produced by the proposed ADNet for the given conditional state $s_{t,l}$. When the tracking simulation is done, tracking scores $\{z_{t,l}\}$ are computed with the ground truths $\{G_l\}$ (line 10 in Algorithm 1). In lines 11 and12 in Algorithm 1, the score in the tracking simulation $z_{t,l} = r(s_{T_l,l})$ is the reward at the terminal state, which obtains $+1$ for tracking success and -1 for failure at frame l, as defined in (3).

To define the objective to train the ADNet, we first denote the tracking simulation sequences $\{u_l\}_l^{\mathcal{L}}$, where $u_l = \{s_{t,l}, a_{t,l}\}_{t=1}^{T_l}$ is the state and action sequences at frame l. The tracking performance of u_l is denoted as $R(u_l) = \frac{1}{T_l}\sum_t^{T_l} z_{t,l} = z_{T_l,l}$. The goal of the ADNet tracker is to maximize the tracking performance under the distribution $p(u_l; W_{\mathrm{RL}})$

$$\begin{aligned} J(W_{\mathrm{RL}}) &= \mathbb{E}_{u_l \sim p(u_l; W_{\mathrm{RL}})}[R(u_l)] \\ &= \sum_{u_l \in \mathbb{U}} p(u_l; W_{\mathrm{RL}}) R(u_l) \end{aligned} \tag{8}$$

where $\mathbb{U}$ denotes the set of all the possible sequences and $p(u_l; W_{\mathrm{RL}})$ is the distribution over the tracking sequence u_l depending on the policy at lth frame. Using the Markov assumption, the expansion of $p(u_l; W_{\mathrm{RL}})$ is given by

$$p(u_l; W_{\mathrm{RL}}) = p(s_{1,l}) \Pi_{t=1}^{T} p(s_{t+1,l}|s_{t,l}, a_{t,l}) p(a_{t,l}|s_{t,l}; W_{\mathrm{RL}}). \tag{9}$$

Taking the gradient of (8) to maximize the objective J

$$\nabla_{W_{\mathrm{RL}}} J(W_{\mathrm{RL}}) = \sum_{u_l \in U} \nabla_{W_{\mathrm{RL}}} p(u_l; W_{\mathrm{RL}}) R(u_l). \tag{10}$$

Using the log-likelihood trick

$$\nabla_{W_{\mathrm{RL}}} \log p(u; W_{\mathrm{RL}}) = \frac{\nabla_{W_{\mathrm{RL}}} p(u; W_{\mathrm{RL}})}{p(u; W_{\mathrm{RL}})}$$

(10) leads to

$$\begin{aligned} &\nabla_{W_{\mathrm{RL}}} J(W_{\mathrm{RL}}) \\ &\quad = \sum_{u_l \in U} p(u_l; W_{\mathrm{RL}}) \nabla_{W_{\mathrm{RL}}} \log p(u_l; W_{\mathrm{RL}}) R(u_l) \\ &\quad = \mathbb{E}_{u_l \sim p(u_l; W_{\mathrm{RL}})}[\nabla_{W_{\mathrm{RL}}} \log p(u_l; W_{\mathrm{RL}}) R(u_l)]. \end{aligned} \tag{11}$$

Replacing $p(u_l; W_{\mathrm{RL}})$ with (9) and ignoring the terms unrelated with W_{RL}

$$\begin{aligned} &\nabla_{W_{\mathrm{RL}}} J(W_{\mathrm{RL}}) \\ &\quad \propto \mathbb{E}_{u_l \sim p(u_l; W_{\mathrm{RL}})}\left[\sum_t \nabla_{W_{\mathrm{RL}}} \log p(a_{t,l}|s_{t,l}; W_{\mathrm{RL}}) R(u_l)\right] \\ &\quad \approx \frac{1}{\mathcal{L}} \sum_l^{\mathcal{L}} \sum_t^{T_l} \nabla_{W_{\mathrm{RL}}} \log p(a_{t,l}, s_{t,l}; W_{\mathrm{RL}}) R(u_l) \end{aligned}$$

where the last line is obtained from a sample approximation [18]. Since $R(u_l) = z_{T_l,l}$, the network parameters W_{RL} are updated by stochastic gradient ascent to maximize the objective function (8) using the approximated gradient

$$\Delta W_{\mathrm{RL}} \propto \sum_{l=1}^{\mathcal{L}} \sum_{t=1}^{T_l} \nabla_{W_{\mathrm{RL}}} \log p(a_{t,l}|s_{t,l}; W_{\mathrm{RL}}) z_{T_l,l}. \tag{12}$$

Our RL scheme can train ADNet even if the ground truths $\{G_l\}$ are partially given, which means the SS setting as shown in Fig. 4. The SL scheme cannot learn the information of the unlabeled frames; however, the RL can utilize the unlabeled frames in SS manner. In order to train ADNet in RL, the tracking scores $\{z_{t,l}\}$ should be determined, but the

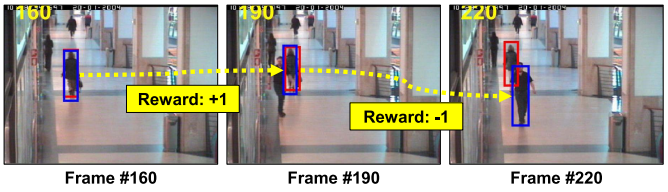

Fig. 4. Illustration of the tracking simulation in SS case on *Walking2* sequence. Red boxes and blue boxes denote the ground truths and the predicted target's positions, respectively. In this example, only the frames #160, #190, and #220 are annotated. Through the sequential actions, ADNet tracker at frame #190 receives $+1$ reward and -1 at frame #220. Therefore, the tracking scores from frame #161 to #190 will be $+1$ and -1 between #191 and #220.

tracking scores in the unlabeled sequences cannot be immediately determined. Instead, we assign the tracking scores with the reward obtained from the result of tracking simulation. In other words, if the result of tracking simulation during the unlabeled sequences is evaluated as success at the labeled frame, the tracking scores for the unlabeled frames are given by $z_{t,l} = +1$. If it is not successful, $z_{t,l}$ is assigned by -1, as shown in Fig. 4.

C. Online Adaptation in Tracking

ADNet is updated in online manner during tracking. This online adaptation can make the tracking algorithm more robust against the changes of surroundings or the deformation of the target. When updating ADNet, we fix the convolutional filters $\{w_1, w_2, w_3\}$ and fine-tune the fully connected layers $\{w_4, \ldots, w_7\}$, because the convolutional layers would have generic tracking information, whereas the fully connected layers would have the video-specific knowledge.

Since rewards are not provided during online tracking, RL could not be utilized for online adaptation. Inverse RL (IRL), which estimates the reward function, could be a choice for updating the ADNet. However, IRL needs to conduct tracking simulations with heavy computations and may require many iterations until convergence to train the ADNet. Instead, for computational efficiency, we use SL to update the ADNet. Since SL has a simple training scheme without tracking simulations, training with SL is faster and more stable than with IRL. Through SL in online adaptation, the ADNet is updated to deal with changes in the surroundings and the target object.

The detailed procedure of the proposed tracking and online adaptation is described in Algorithm 3. The tracking is performed by deciding sequential actions with the state-action probability $p(a|s; W)$ generated by ADNet. We adopt the online update adaptation of [12]. The online adaptation is done by fine-tuning ADNet with an SL using the temporal training samples generated during the tracking process. For the SL, training samples with labels are required. For the labeling, we have to determine the ground truth. The tracked patch position determined by ADNet is used for the temporal ground truth. As similar to SL (see Section IV-A), the training sample set $\mathbb{S}$ for online adaptation consists of image patches $\{p_i\}$ sampled randomly around the tracked patch position and the corresponding action labels $\{o_i^{\text{act}}\}$

Algorithm 3 Online Adaptation of ADNet in Tracking

Input: Trained ADNet (W), Test sequences $\{F_l\}_{l=1}^{L}$ and initial target position $b_{1,1}$
Output: Estimated target positions $\{b_{T_l,l}\}_{l=2}^{L}$

Generate samples $\mathbb{S}_1$ from F_1 and G_1
Update W using $\mathbb{S}_1$
Set initial $d_{1,1}$ as zero vector
$T_1 \leftarrow 1$
for $l = 2$ to L **do**
  $\{a_{t,l}\}, \{b_{t,l}\}, \{d_{t,l}\}, T_l \leftarrow$ TRACKER$(b_{T_{l-1},l-1}, d_{T_{l-1},l-1}, F_l)$ in Algorithm 2
  $s_{T_l,l} \leftarrow (\phi(b_{T_l,l}, F_l), d_{T_l,l})$
  if $c(s_{T_l,l}) > 0.5$ **then**
    Generate samples $\mathbb{S}_l$ from F_l and $b_{T_l,l}$
  else if $c(s_{T_l,l}) < 0.5$ **then**
    Draw $\mathcal{N}_{\text{det}}$ target position candidates $\{\tilde{b}_i\}$
    Re-detect the target position $b^* \leftarrow arg\max_{\tilde{b}_i} c(\tilde{b}_i)$
    $s_{T_l,l} \leftarrow (\phi(b^*, F_l), d_{T_l,l})$
  end if
  if $\text{mod}(l, \mathcal{I}) = 0$ **then**
    Update W using the samples $\{\mathbb{S}_k\}_{k=l-\mathcal{J}+1}^{l}$
  end if
end for

and confidence score labels $\{o_i^{\text{con}}\}$. The labels are obtained via (4) and (5). At the first frame, the initial samples $\mathbb{S}_{init}$ are generated using the initial target position, and ADNet is fine-tuned to fit the given target. At frame $l(\geq 2)$, the training samples $\mathbb{S}_l$ are generated using the tracked patch position $b_{T_l,l}$ if the confidence score $c(s_{t,l})$ of the estimated target is above 0.5. The confidence score $c(s_{t,l})$ of the state $s_{t,l}$ is defined as the target probability $p(\text{target}|s_{t,l}; W)$ of the confidence layer (fc7). Online adaptation is conducted for every $\mathcal{I}$ frames using the training samples $\{\mathbb{S}_k\}_{k=l-\mathcal{J}+1}^{l}$, which means the online adaptation uses the training samples generated from the past $\mathcal{J}$ frames. When the score $c_{t,l}$ is below 0.5, which means the ADNet tracker miss the target, the *redetection* is conducted to capture the missed target. The target position candidates $\{\tilde{b}_i\}_{i=1}^{\mathcal{N}_{\text{det}}}$ are generated around the current target position with random Gaussian noise. The redetected target position b^* is selected by

$$b^* = \arg\max_{\tilde{b}_i} c(\tilde{b}_i) \tag{13}$$

and the state $s_{T_l,l}$ is assigned by the target position b^* and the action dynamics vector $d_{T_l,l}$.

D. Implementation Details

1) Pretraining the ADNet: In each frame, we generated the training samples $\{p_j\}$ by perturbation with Gaussian distribution whose mean is zero and covariance matrix is $\text{diag}((0.3w)^2, (0.3h)^2, (0.1w)^2, (0.1h)^2)$. When pretraining ADNet, we draw 250 samples in each frame. We set the learning rate to 0.0001 for convolutional layers (fc1–fc3) and 0.001 for fully connected layers (fc4–fc7) [12], a momentum to 0.9, a weight decay to 0.0005, and a mini-batch size to 128. For pretraining ADNet with $\mathcal{K}$ training videos, the number

TABLE I
PARAMETERS FOR ADNET AND ADNET-FAST

	ADNet	ADNet-fast
Initial samples # ($\mathcal{N}_I$)	3000	300
Online samples # ($\mathcal{N}_O$)	250	50
Re-detection samples # ($\mathcal{N}_{det}$)	256	64
Online adaptation in every ($\mathcal{I}$) frames	10	30

TABLE II
EXPERIMENTAL RESULTS WITH THREE MOVEMENT SIZES (2%, 3%, AND 4%) ON OTB-100 DATA SET

Movement size	Prec. (20px)	Speed (FPS)
2%	86.5	2.83
3%	88.0	2.90
4%	86.8	2.96

of training iteration was set to 300 for each video. In each iteration of the RL, we randomly selected the sequence of length $\mathcal{L}(= 10)$ for the tracking simulation.

2) Online Adaptation of the ADNet: For online adaptation, we only trained the fully connected layers (fc4–fc7) with the learning rate 0.001. In addition to the normal version (ADNet), to reduce the computation in actual tracking, we can apply the fast version of ADNet using a small number of samples in online adaptation, referred to as "ADNet-fast." The parameters of online adaptation for ADNet and ADNet-fast are described in Table I. The ADNet was fine-tuned with $\mathcal{T}_I(= 300)$ iterations at the first frame and $\mathcal{T}_O(= 30)$ iterations for the online adaptations. The online training was conducted for every $\mathcal{I}(= 10)$ frames, and the training data were sampled from the past $\mathcal{J}(= 20)$ frames. For the *redetection*, we draw $\mathcal{N}_{\text{det}}(= 256)$ target position candidates. In the online adaptation, $\mathcal{N}_{\text{I}}(= 3000)$ samples were generated in the first frame, and $\mathcal{N}_{\text{O}}(= 250)$ samples were generated in the frame whose confidence was above 0.5 during tracking.

In ADNet-fast, we set $\mathcal{N}_{\text{I}}$ to 300, $\mathcal{N}_{\text{O}}$ to 50, $\mathcal{I}$ to 30, and $\mathcal{N}_{\text{det}}$ to 64. Tracking with ADNet-fast endured 3% performance degradation but achieved a real-time speed around four times faster than the standard version of ADNet.

V. EXPERIMENTS

We evaluated our method on the popular visual tracking benchmarks, object tracking benchmark (OTB) [14], [49], compared with existing trackers. Also, we validated the effectiveness of ADNet by demonstrating various self-comparisons. The experiments were conducted on the following specifications: i7-4790K CPU, 32-GB RAM, and GTX TITAN X GPU using MATLAB2016b and MatConvNet toolbox [50]. In our settings, ADNet and ADNet-fast run at 3 and 15 frames/s on the GPU, respectively.

A. Data Sets

We evaluated our method on two OTB data sets: OTB-50 [49], which has 50 video sequences, and OTB-100 [14], which has 100 video sequences, including OTB-50. The videos in the OTB data sets have various attributes, including illumination variation, scale variation, occlusion, deformation, motion blur, fast motion, in-plane rotation, out-of-plane rotation, out-of-view, background clutters, and low resolution, which represent the challenging aspects in visual tracking. In order to pretrain ADNet, we used 360 training videos from VOT2013 [51], VOT2014 [15], VOT2015 [22], and ALOV300 [21], in which videos overlapping with OTB-50 and OTB-100 were excluded. The tracking performance was measured by conducting a one-pass evaluation (OPE) based on two metrics: center location error and overlap ratio [49]. The OPE is an approach to evaluation that initializes with the ground-truth position in the first frame and runs a tracker throughout a test sequence. The center location error measures the distance between the center of the tracked frame and the ground truth, and the bounding box overlap ratio measures the IOU ratio between the tracked bounding box and the ground truth.

B. Analysis on ADNet Property

1) Movement Step Size: Since the action of our decision process is defined in discrete space, the tracker moves with a discrete amount of step size. The movement step size, which determines how much to move by an action, has a tradeoff between computational complexity and tracking accuracy. If the size is too small, the number of iterations to capture the target increases, and the tracker may not cope with fast motion of the target. In contrast, if it is too large, the accuracy is not precise when the tracking action is terminated by the stop action. In this paper, the movement step size was determined in proportion to the object size to control the number of iterations regardless of the size of the target and was empirically set to 3% of the width and height of the current target. Table II shows the tradeoff between the precision and the computation time depending on the movement step size, in experiments on the OTB-100 data set to examine the precision by changing the movement step size to 2%, 3%, and 4% of the width and the height. As shown in Table II, the tracking speed gets higher when the step size is increasing, and the tracking accuracy gets degradations when the step size is smaller (2%) or larger (4%) than the selected step size 3%.

2) Action Dynamics Vector: The goal of the action dynamics vector is to consider the movement dynamics of the target object by utilizing the past actions when deciding the current action. As described in Section III-B, the action dynamics vector is encoded by concatenating past ten actions $d_{t,l} \in \mathbb{R}^{110}$, and the network output has 11-D action probability. Therefore, the network weights connected from action dynamics to output nodes in fc6 layer can be represented as a matrix $W_{\text{dyn}} \in \mathbb{R}^{110\times 11}$. To investigate the effect of the action dynamics feature on the current action, we plotted the squared values of W_{dyn} in Fig. 5(a), where higher values are indicated by brighter colors and lower values by darker ones. The x-axis denotes the index of the action dynamics vector, and the y-axis denotes the index of the output action node. In Fig. 5(a), the 11×11 blocks corresponding to ten actions in the past are shown, where each block indicates the effect of the past action on the action output. For more detailed analysis, in Fig. 5(b), we plotted the first block of W_{dyn}, corresponding

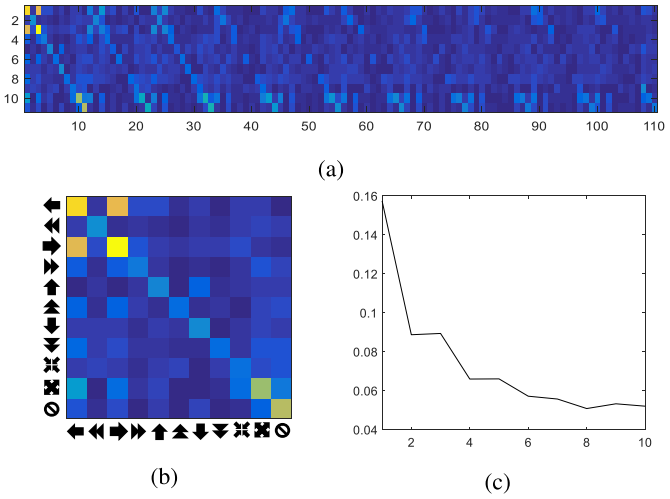

Fig. 5. Analysis of the network weights of action dynamics vector to decide the current action. (a) Squared weight values from action dynamics (110 dimensions) to action outputs (11 dimensions). (b) First action block of the weights shown in (a). (c) Influence of the past actions: the sum of squares of the weight values in the past kth block.

to the action just before the current state. The x-axis and y-axis, respectively, represent the past and current actions. For example, the first row denotes the connected weights from the past actions to the "left" action. The high diagonal elements in Fig. 5(b) mean that the same action as the previous action tends to be selected because of inertial motion dynamics of the moving object. From the first row, the weight connected to the previous "right" action also has a high value to select the "left" action in the current step. In other words, the selection of the opposite action to the previous action is also highly probable. In addition, we measured the influence of the past actions by summing the elements in each block of W_{dyn} as follows:

$$\text{Influence of actions in the past } k\text{th block} = \sum_{w \in W_{\text{dyn}}[k]} w^2$$

where $W_{\text{dyn}}[k]$ denotes the kth block of the matrix W_{dyn}. The influence of the past actions is shown in Fig. 5(c). The influence value is decreasing when the element index of the action dynamics vector is increasing in the x-axis, that is, the more past action has the weaker influence to decide the current action.

3) Efficiency of the Searching Strategy: Fig. 6 shows an examples of the sequential actions selected by the proposed method. The first column shows the target position in the previous frame, and the second column shows the current position of the target. The bounding box flows from the previous position (blue) to the captured target position (red) are shown in the third column. In the remaining part of Fig. 6, the sequential transitions of the state and the corresponding actions are represented by the image patches and the action symbols. In the sequential state transition, we can see that ADNet selects proper actions to capture the target in each state.

Fig. 7 shows the histograms of the number of moves per frame and the number of selections for each action in the test sequences. As shown in Fig. 7(a), the ratio of the frames requiring more than five actions to capture the target to the whole frames was only around 7%. That is, most of the frames require fewer than five actions to pursue the target in each frame. The proposed tracker occasionally conducted redetection, and the ratio of the frames using redetection to the whole frames was around 9%. The average number of searching steps, including the required actions and the candidates by redetection, is 28.26 per frame, which is much smaller than that of the state-of-the-art trackers, such as multi-domain network (MDNet) [12] (= 256 per frame). We also plot how many times each action was selected during the test sequences in Fig. 7(b). With the movement step size and action types defined by the proposed method, the numbers of selected actions are uniformly distributed except stop action. The number of selection of stop action is much higher than others, since the stop action is always selected at each frame. Fig. 8 qualitatively shows the efficiency of the ADNet tracker pursuing the target by sequential actions, compared with the existing tracker based on the tracking-by-detection strategy [12]. In Fig. 8(b), the green, red, and blue boxes denote the previous target position, the current target position, and the generated target candidates, respectively. Fig. 8(c) shows the tracking process of ADNet by selecting sequential actions.

TABLE III
WEIGHT CHANGES OF ADNET FROM THE INITIAL NETWORK (VGG-M [16]). THE MEAN VALUES OF THE ABSOLUTE DIFFERENCE BETWEEN THE WEIGHTS OF THE TRAINED LAYER AND INITIAL WEIGHTS ARE PRESENTED TO SHOW THE WEIGHT CHANGES BY THE PROPOSED TRAINING METHOD

Layer	Weight changes (Mean abs.)
conv1	0.0049
conv2	0.0011
conv3	0.0013
fc4	0.0102
fc5	0.0119
fc6	0.0113
fc7	0.0115

4) Weight Changes Resulting From the Proposed Training: To investigate the influence of SL and RL, we analyzed the weight changes from the initial network. The three convolutional (conv) layers of the ADNet {conv1, conv2, conv3} were initialized by adopting the VGG-M network [16], and the fully connected (fc) layers {fc4, fc5, fc6, fc7} were initialized using Gaussian noises. Table III describes the weight change for each layer by computing the mean of the absolute difference between the trained weights and the initial weights. As shown in Table III, the fc layers {fc4, fc5, fc6, fc7} tend to have larger weight changes than the conv layers {conv1, conv2, conv3}. Although the fc layers have an important role in learning the policy to decide on an action, the weights of the conv layers were also changed to learn the feature representation corresponding to the visual tracking problem.

C. Self-Evaluation

To verify the effectiveness of the components of ADNet, we conducted four variants of ADNet and evaluated

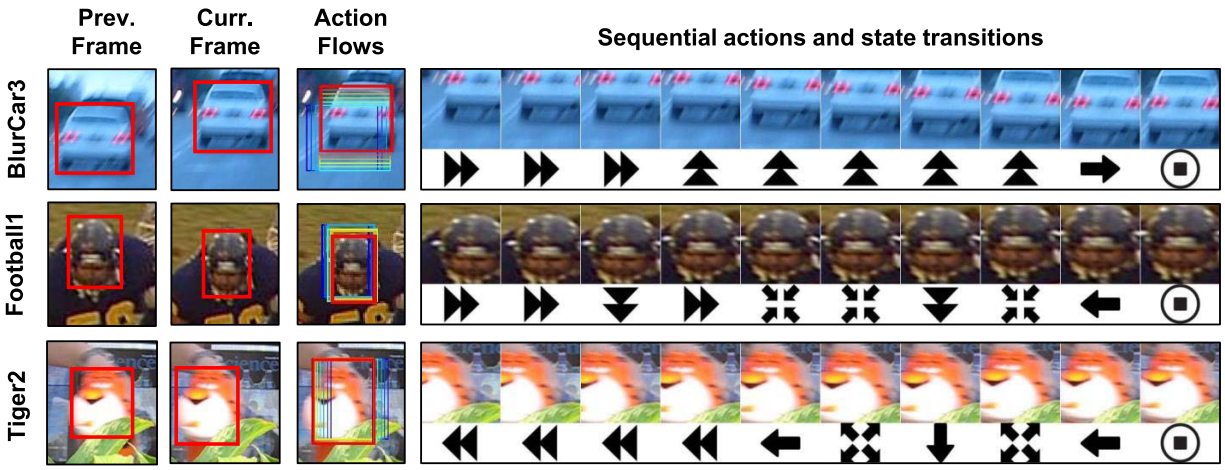

Fig. 6. Examples of sequential actions selected by ADNet on *BlurCar3*, *Football1*, and *Tiger2* sequences. The action flows from the previous position to the current position, and the state transitions are presented.

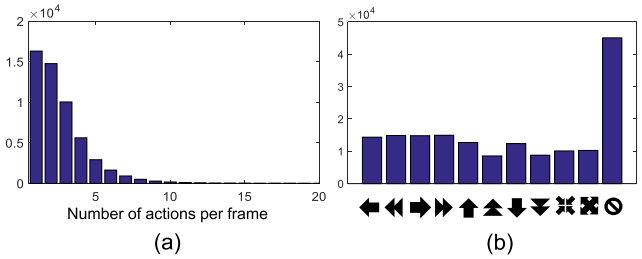

Fig. 7. Histograms of (a) number of moves per frame and (b) number of selections of each action in the test sequences.

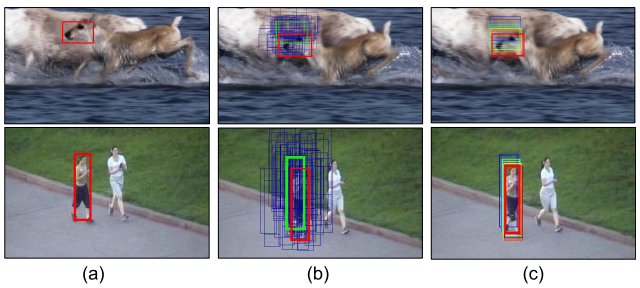

Fig. 8. Searching strategy comparison between the existing tracking-by-detection approach [12] (second column) and the proposed method (third column) on the *Deer* and *Jogging-1* sequences. (a) Previous frame. (b) Tracking-by-detection method. (c) Our method.

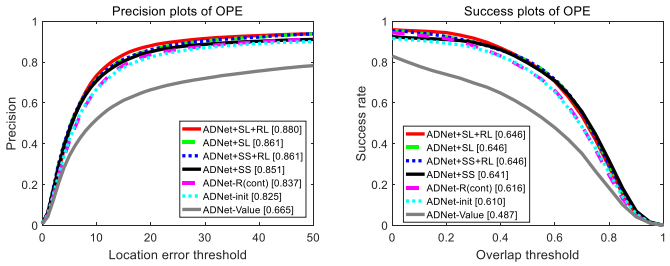

Fig. 9. Self-evaluation results in experiments on OTB-100. The scores in the legend indicate the mean precisions when the location error threshold is 20 pixels for the precision plots and area under curve for the success plots.

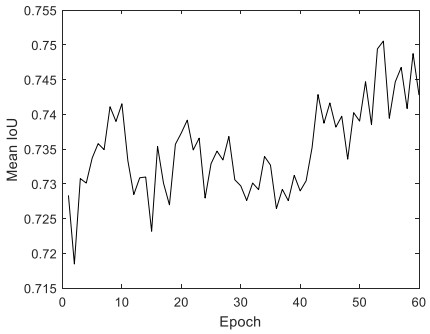

Fig. 10. Plot of average tracking IoU between the tracking box and the ground-truth box for training sequences during RL.

them using OTB-100. We first conducted the baseline "ADNet-init," which is not pretrained and simply uses the initial parameters. In ADNet-init, the parameters of convolutional networks (conv1–conv3) are initialized with the VGG-M [16] model, and the fully connected layers (fc4–fc7) are initialized with random noises. "ADNet + SL" is the pretrained models with SL using fully labeled frames of the training sequences. "ADNet + SS" is trained using partially labeled data in the SS settings. In the training of ADNet + SS, the ground-truth annotation is provided only every ten frames. Then, we conducted "ADNet + SL + RL" and "ADNet + SS + RL" by training ADNet + SL and ADNet + SS using RL, respectively. ADNet + SL + RL is the final version of the proposed method. The precision and the success rate of the self-comparisons are shown in Fig. 9. By conducting SL, ADNet + SL and ADNet + SS achieved 3.6% and 2.6% improvement in precision performance, respectively, compared with ADNet-init. In the SS case, the precision is 1% lower than that in the supervised case because of the lack of ground-truth annotations. When conducting RL, ADNet + SL + RL and ADNet + SS + RL gained 1.9% and 1.0% additional improvement in precision performance to ADNet + SL and ADNet + SS, respectively. The results show that the RL can improve performance not only the SS case but also the supervised case.

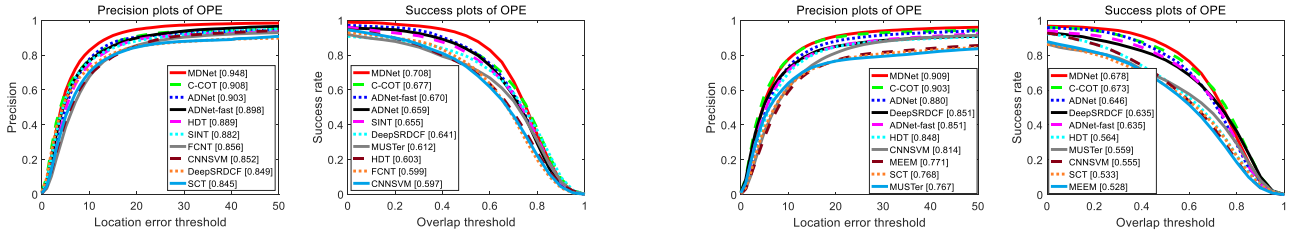

Fig. 11. Precision and success plots on OTB-50 [49] and OTB-100 [14]. Only the top ten trackers are presented. (a) OTB-50. (b) OTB-100.

The change of the tracking performance during RL is shown in Fig. 10. To investigate trends in tracking performance improvement during the tracking simulation, we plot the average of IoU scores at the end of each epoch of the tracking simulation for RL. As shown in Fig. 10, the average IoU score increases as the epoch increases.

To examine the tracking performance according to the type of the reward function, we conducted an "ADNet-R(cont)," whose reward function was defined as a *continuous* function. The continuous reward function uses IoU scores as follows:

$$r(s_T) = \text{IoU}(b_T, G). \tag{14}$$

In this experiment, the network architecture and the training method were the same as the standard version of the ADNet, except for the reward function. As shown in Fig. 9, the tracking precision of ADNet-R(IoU) is 4.3% lower than the ADNet using the discrete reward function of (3). In our RL, the reward function is computed at the end of the tracking simulation sequence. Since the discrete reward function can give an obvious reward depending on whether the tracking simulation has succeeded or failed, the discrete reward function is more suitable than the continuous one. In addition, since the output of the ADNet is formed in a *one-hot* vector, the discrete reward is more appropriate for training the ADNet than the continuous one. In detail, the ADNet with discrete rewards learns only successful actions by using a positive reward (+1) when the IoU score is larger than 0.7. On the other hand, ADNet-R(IoU) with continuous rewards learns the obscure tracking results with low IoU scores, which interferes with learning the appropriate action distribution given in the one-hot form.

Furthermore, we conducted an "ADNet-Value" where the *value-function-based* method DQN [39] was used instead of the policy gradient method adopted in our RL. The architecture of ADNet-Value was the same as the proposed method, where the softmax layer was removed to obtain Q-values instead of action probability. We adopted the training algorithm and the parameters from the DQN paper [39]. ADNet-Value also has SL and online adaptation steps, which are critical components for tracking performance. The SL and online adaptation were the same as described in Sections IV-A and IV-C. In the experiments, the tracking precision of ADNet-Value was about 20% lower than the proposed method. The reason for the performance degradation is that the value-function-based method is not suitable for our ADNet tracking scheme with one-hot form outputs. One of the reasons for choosing the one-hot form is to utilize SL, where the ground truth is given in one-hot form. For additional reinforcement, the policy gradient method is appropriate for learning the action probability distribution of the one-hot form. In contrast, Q-values are not in the one-hot form. Thus, the Q-values could destroy the one-hot form policy trained by SL, which leads to much worse performance than the ADNet using the policy gradient method.

TABLE IV
SUMMARY OF EXPERIMENTS ON OTB-100

	Algorithm	Prec.(20px)	IOU(AUC)	FPS	GPU
	ADNet	88.0%	0.646	2.9	O
	ADNet-fast	85.1%	0.635	15.0	O
Non real-time	MDNet [12]	90.9%	0.678	< 1	O
	C-COT [25]	90.3%	0.673	< 1	O
	DeepSRDCF [24]	85.1%	0.635	< 1	O
	HDT [52]	84.8%	0.564	5.8	O
	MUSTer [8]	76.7%	0.528	3.9	X
Real-time	MEEM [53]	77.1%	0.528	19.5	X
	SCT [9]	76.8%	0.533	40.0	X
	KCF [7]	69.7%	0.479	223	X
	DSST [6]	69.3%	0.520	25.4	X
	GOTURN [30]	56.5%	0.425	125	O

D. Comparative Evaluation

We compared ADNet in a comprehensive comparison with 13 state-of-the-art trackers, including MDNet [12], continuous convolution operator tracker (C-COT) [25], generic object tracking using regression network (GOTURN) [30], hedged deep tracker (HDT) [52], deep spatially regularized discriminative correlation filters (DeepSRDCF) [24], siamese instance search for tracking [17], fully convolutional network tracker [29], structuralist cognitive model for visual tracking (SCT) [9], multi-store tracker (MUSTer) [8], CNN-support vector machine [11], multiple experts using entropy minimization (MEEM) [53], discriminative scale space tracker (DSST) [6], and KCF [7]. Fig. 11 shows the plots of precision and success rate based on center location error and overlap ratio, respectively, and Table IV summarizes the comparison of tracking performance with computational speed (frames/s) and GPU usage. The evaluation was conducted on OTB-100 data set, and the tracking performances and computational speeds of the comparing trackers were adopted from their original papers except the GOTURN tracker [30]. Since the GOTURN tracker was not evaluated on OTB data set, we conducted the GOTURN tracker [30] on the OTB data set using the author's code. The proposed method shows comparable performance with the state-of-the-art trackers,

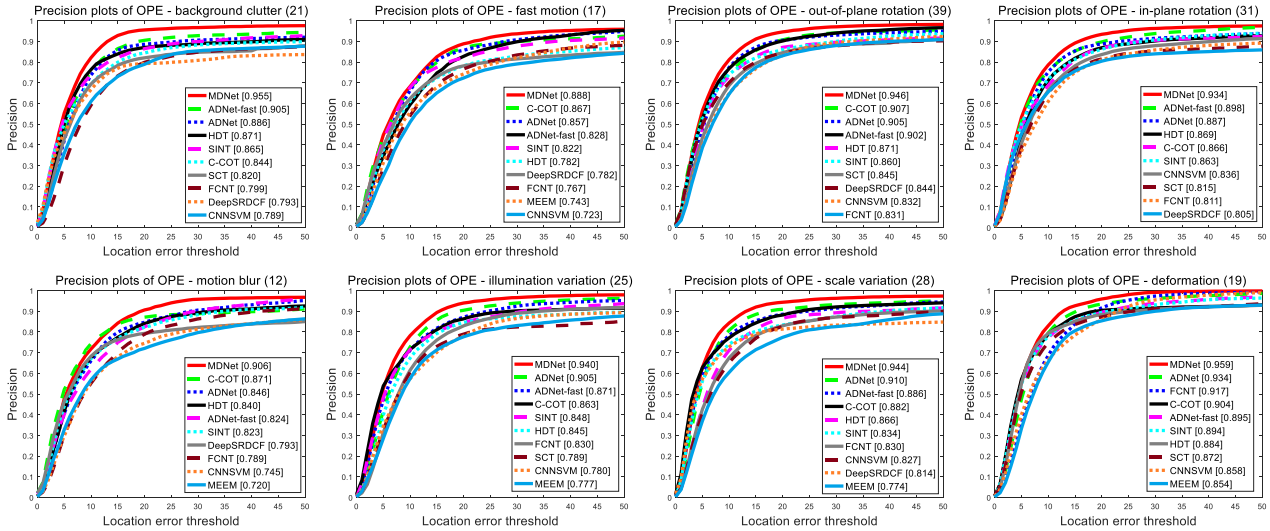

Fig. 12. Precision plots on OTB-50 for the subset of challenging attributes: *background clutter, fast motion, out-of-plane rotation, in-plane rotation, motion blur, illumination variation, scale variance, and deformation*. The plotted curves are based on the center location error. Only the top ten trackers are presented.

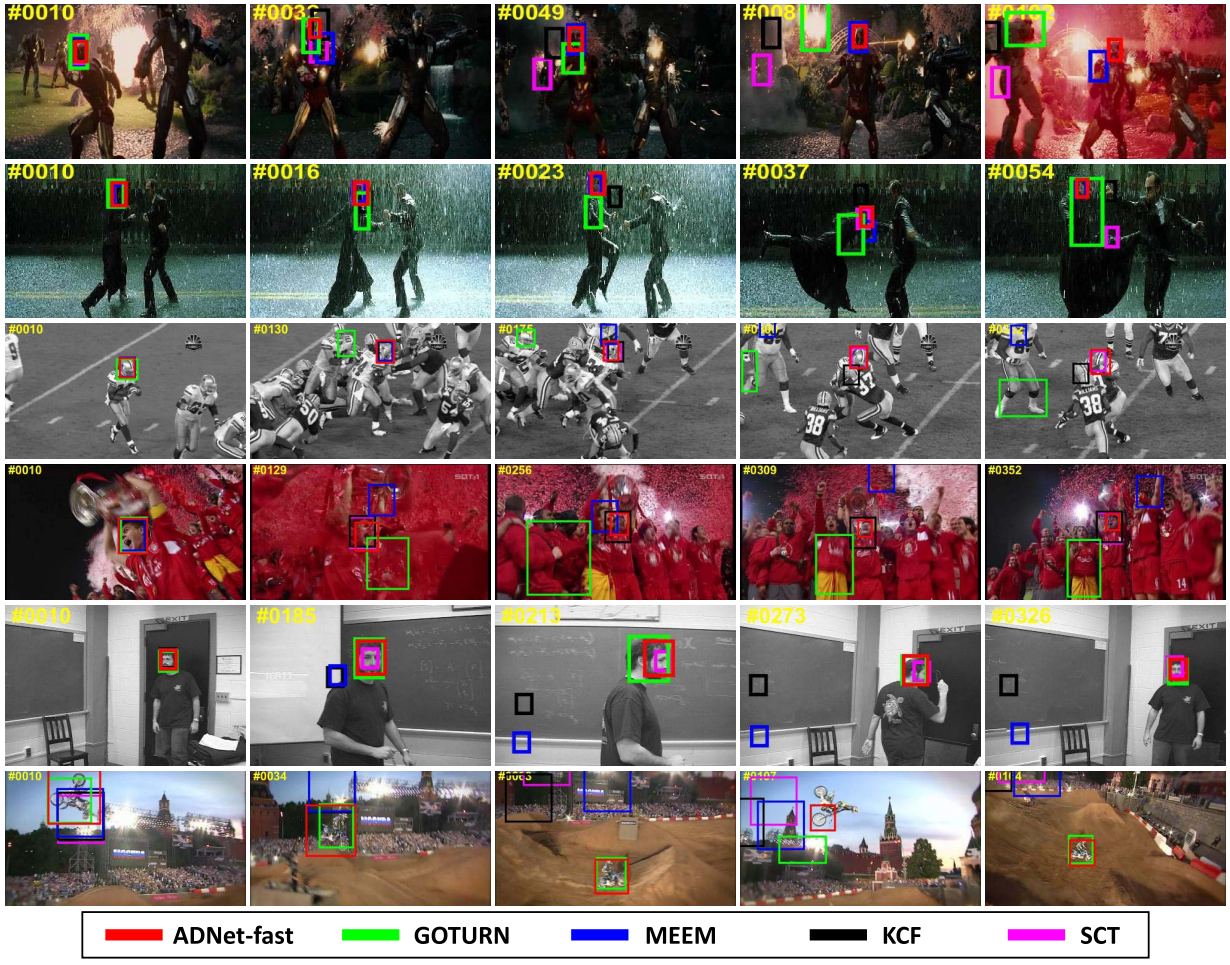

Fig. 13. Qualitative results of ADNet. The test sequences are *Ironman, Matrix, Football, Soccer, Freeman1,* and *MotorRolling*.

MDNet [12] and C-COT [25], in both precision and success rate. The proposed method is much efficient in computation, where ADNet is about three times faster (3 frames/s) than MDNet and C-COT. However, the proposed method (ADNet) is still much slower than the real-time speed (15 frames/s). ADNet-fast, the fast version of ADNet, shows the real-time speed (15 frames/s) without much loss of performance (3% performance degradation) compared with the existing non-real-time CNN-based trackers, such as DeepSRDCF [24] and HDT [52]. As shown in Table. IV, ADNet-fast achieved

the new state-of-the-art performance with more than 8% improvement of the best performance among the existing real-time tracking algorithms. MEEM [53], SCT [9], KCF [7], and DSST [6] used hand-crafted features with only CPUs. GOTURN [30] used deep neural networks with GPUs similar to our methods, but the tracking performance of GOTURN is much worse than the proposed method.

Fig. 12 shows the tracking performance on various challenging tracking situations, such as background clutter, fast motion, deformation, and so on. The proposed methods, ADNet and ADNet-fast, demonstrate robust tracking performance to handle the challenging situations, where high-level semantics and contextual information are required.

The qualitative results of the proposed method are shown in Fig. 13. The results compare the proposed method (ADNet-fast) with the real-time state-of-the-art trackers, GOTURN [30], MEEM [53], KCF [7], and SCT [9]. The results show the robustness of the trackers in the challenging sequences, where the scale variation, illumination change, occlusion, and deformation are appeared. We plot critical frames in the test sequences, where the proposed tracker successfully captures the target, while the other trackers often fail to track. The existing real-time state-of-the-art trackers have difficulty to track in the complex scene as in Fig. 13, since they use lightweight features or simple tracking models to obtain fast speed. In contrast, the proposed method achieves the satisfactory tracking performance as well as a real-time speed with the help of the efficient action-driven search strategy.

VI. Conclusion

In this paper, we have proposed a novel action-driven method using deep convolutional networks for visual tracking. The proposed tracker is controlled by an ADNet, which pursues the target object by sequential actions iteratively. The action-driven tracking strategy makes a significant contribution to the reduction of computation complexity in tracking. In addition, RL makes it possible to use partially labeled data, which could greatly contribute to the building of training data with a little effort. According to the evaluation results, the proposed tracker achieves the state-of-the-art performance in 3 frames/s, which is three times faster than the existing deep network-based trackers using a tracking-by-detection strategy. Furthermore, the fast version of the proposed tracker achieves a real-time speed (15 frames/s) by adjusting the metaparameters of the ADNet, with an accuracy that outperforms state-of-the-art real-time trackers.

References

[1] H. Grabner, M. Grabner, and H. Bischof, "Real-time tracking via on-line boosting," in *Proc. Brit. Mach. Vis. Conf.*, 2006, vol. 1. no. 5, p. 6.

[2] H. Grabner, C. Leistner, and H. Bischof, "Semi-supervised on-line boosting for robust tracking," in *Proc. Eur. Conf. Comput. Vis.*, 2008, pp. 234–247.

[3] B. Babenko, M.-H. Yang, and S. Belongie, "Robust object tracking with online multiple instance learning," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 33, no. 8, pp. 1619–1632, Aug. 2011.

[4] Z. Kalal, K. Mikolajczyk, and J. Matas, "Tracking-learning-detection," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 34, no. 7, pp. 1409–1422, Jul. 2012.

[5] D. S. Bolme, J. R. Beveridge, B. A. Draper, and Y. M. Lui, "Visual object tracking using adaptive correlation filters," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2010, pp. 2544–2550.

[6] M. Danelljan, G. Häger, F. Khan, and M. Felsberg, "Accurate scale estimation for robust visual tracking," in *Proc. Brit. Mach. Vis. Conf.*, Nottingham, U.K., Sep. 2014.

[7] J. F. Henriques, R. Caseiro, P. Martins, and J. Batista, "High-speed tracking with kernelized correlation filters," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 37, no. 3, pp. 583–596, Mar. 2015.

[8] Z. Hong, Z. Chen, C. Wang, X. Mei, D. Prokhorov, and D. Tao, "MUltI-store tracker (MUSTer): A cognitive psychology inspired approach to object tracking," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2015, pp. 749–758.

[9] J. Choi, H. Jin Chang, J. Jeong, Y. Demiris, and J. Y. Choi, "Visual tracking using attention-modulated disintegration and integration," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2016, pp. 4321–4330.

[10] N. Wang, S. Li, A. Gupta, and D.-Y. Yeung. (2015). "Transferring rich feature hierarchies for robust visual tracking." [Online]. Available: https://arxiv.org/abs/1501.04587

[11] S. Hong, T. You, S. Kwak, and B. Han. (2015). "Online tracking by learning discriminative saliency map with convolutional neural network." [Online]. Available: https://arxiv.org/abs/1502.06796

[12] H. Nam and B. Han. (2015). "Learning multi-domain convolutional neural networks for visual tracking." [Online]. Available: https://arxiv.org/abs/1510.07945

[13] O. Russakovsky *et al.*, "ImageNet large scale visual recognition challenge," *Int. J. Comput. Vis.*, vol. 115, no. 3, pp. 211–252, Dec. 2015.

[14] Y. Wu, J. Lim, and M. H. Yang, "Object tracking benchmark," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 37, no. 9, pp. 1834–1848, Sep. 2015.

[15] M. Kristan, *et al.*, "The visual object tracking VOT2014 challenge results," in *Proc. Eur. Conf. Comput. Vis. Workshops*, Berlin, Germany, Mar. 2014, pp. 191–217.

[16] K. Chatfield, K. Simonyan, A. Vedaldi, and A. Zisserman. (2014). "Return of the devil in the details: Delving deep into convolutional nets." [Online]. Available: https://arxiv.org/abs/1405.3531

[17] R. Tao, E. Gavves, and A. W. Smeulders. (2016). "Siamese instance search for tracking." [Online]. Available: https://arxiv.org/abs/1605.05863

[18] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Mach. Learn.*, vol. 8, nos. 3–4, pp. 229–256, 1992.

[19] S. Yun, J. Choi, Y. Yoo, K. Yun, and J. Young Choi, "Action-decision networks for visual tracking with deep reinforcement learning," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2017, pp. 1349–1358.

[20] H. Yang, L. Shao, F. Zheng, L. Wang, and Z. Song, "Recent advances and trends in visual tracking: A review," *Neurocomputing*, vol. 74, no. 18, pp. 3823–3831, Nov. 2011.

[21] A. W. M. Smeulders, D. M. Chu, R. Cucchiara, S. Calderara, A. Dehghan, and M. Shah, "Visual tracking: An experimental survey," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 36, no. 7, pp. 1442–1468, Jul. 2014.

[22] M. Kristan *et al.*, "The visual object tracking vot2015 challenge results," in *Proc. IEEE Int. Conf. Comput. Vis. Workshops*, Dec. 2015, pp. 1–23.

[23] J. Choi, H. J. Chang, S. Yun, T. Fischer, Y. Demiris, and J. Y. Choi, "Attentional correlation filter network for adaptive visual tracking," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jul. 2017, pp. 4828–4837.

[24] M. Danelljan, G. Häger, F. S. Khan, and M. Felsberg, "Convolutional features for correlation filter based visual tracking," in *Proc. IEEE Int. Conf. Comput. Vis. Workshops*, Dec. 2015, pp. 58–66.

[25] M. Danelljan, A. Robinson, F. Shahbaz Khan, and M. Felsberg, "Beyond correlation filters: Learning continuous convolution operators for visual tracking," in *Proc. ECCV*, 2016, pp. 472–488.

[26] N. Wang and D.-Y. Yeung, "Learning a deep compact image representation for visual tracking," in *Proc. Adv. Neural Inf. Process. Syst.*, 2013, pp. 809–817.

[27] H. Li, Y. Li, and F. Porikli, "Robust online visual tracking with a single convolutional neural network," in *Proc. Asian Conf. Comput. Vis.*, 2014, pp. 194–209.

[28] H. Li, Y. Li, and F. Porikli, "DeepTrack: Learning discriminative feature representations online for robust visual tracking," *IEEE Trans. Image Process.*, vol. 25, no. 4, pp. 1834–1848, Apr. 2016.

[29] L. Wang, W. Ouyang, X. Wang, and H. Lu, "Visual tracking with fully convolutional networks," in *Proc. IEEE Int. Conf. Comput. Vis.*, Dec. 2015, pp. 3119–3127.

[30] D. Held, S. Thrun, and S. Savarese. (2016). "Learning to track at 100 FPS with deep regression networks." [Online]. Available: https://arxiv.org/abs/1604.01802
[31] V. Mnih *et al.*, "Recurrent models of visual attention," in *Proc. Adv. Neural Inf. Process. Syst.*, 2014, pp. 2204–2212.
[32] A. Gonzalez-Garcia, A. Vezhnevets, and V. Ferrari, "An active search strategy for efficient object class detection," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2015, pp. 3022–3031.
[33] D. Yoo, S. Park, J.-Y. Lee, A. S. Paek, and I. So Kweon, "Attentionnet: Aggregating weak directions for accurate object detection," in *Proc. IEEE Int. Conf. Comput. Vis.*, Dec. 2015, pp. 2659–2667.
[34] J. C. Caicedo and S. Lazebnik, "Active object localization with deep reinforcement learning," in *Proc. IEEE Int. Conf. Comput. Vis.*, Dec. 2015, pp. 2488–2496.
[35] S. Mathe, A. Pirinen, and C. Sminchisescu, "Reinforcement learning for visual object detection," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2016, pp. 2894–2902.
[36] Y. Xiang, A. Alahi, and S. Savarese, "Learning to track: Online multi-object tracking by decision making," in *Proc. IEEE Int. Conf. Comput. Vis.*, Dec. 2015, pp. 4705–4713.
[37] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, "The PASCAL visual object classes (VOC) challenge," *Int. J. Comput. Vis.*, vol. 88, no. 2, pp. 303–338, Jun. 2010.
[38] A. Barto and R. S. Sutton, *Introduction to Reinforcement Learning*, vol. 135. Cambridge, MA, USA: MIT Press, 1998.
[39] V. Mnih *et al.* (2013). "Playing Atari with deep reinforcement learning." [Online]. Available: https://arxiv.org/abs/1312.5602
[40] V. Mnih *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, pp. 529–533, Feb. 2015.
[41] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," in *Proc. AAAI*, vol. 16. Feb. 2016, pp. 2094–2100.
[42] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, "Deterministic policy gradient algorithms," in *Proc. ICML*, 2014.
[43] D. Silver *et al.*, "Mastering the game of Go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.
[44] Z. Wang, T. Schaul, M. Hessel, H. van Hasselt, M. Lanctot, and N. de Freitas. (2015). "Dueling network architectures for deep reinforcement learning." [Online]. Available: https://arxiv.org/abs/1511.06581
[45] S. Gu, T. Lillicrap, I. Sutskever, and S. Levine. (2016). "Continuous deep Q-learning with model-based acceleration." [Online]. Available: https://arxiv.org/abs/1603.00748
[46] T. P. Lillicrap *et al.* (2015). "Continuous control with deep reinforcement learning." [Online]. Available: https://arxiv.org/abs/1509.02971
[47] D. Jayaraman and K. Grauman. (2016). "Look-ahead before you leap: End-to-end active recognition by forecasting the effect of motion." [Online]. Available: https://arxiv.org/abs/1605.00164
[48] D. Zhang, H. Maei, X. Wang, and Y.-F. Wang. (2017). "Deep reinforcement learning for visual object tracking in videos." [Online]. Available: https://arxiv.org/abs/1701.08936
[49] Y. Wu, J. Lim, and M.-H. Yang, "Online object tracking: A benchmark," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2013, pp. 2411–2418.
[50] A. Vedaldi and K. Lenc, "MatConvNet: Convolutional neural networks for MATLAB," in *Proc. ACM Int. Conf. Multimedia*, 2015, pp. 689–692.
[51] M. Kristan *et al.*, "The visual object tracking VOT2013 challenge results," in *Proc. IEEE Int. Conf. Comput. Vis. Workshops*, Dec. 2013, pp. 98–111.
[52] Y. Qi *et al.*, "Hedged deep tracking," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2016, pp. 4303–4311.
[53] J. Zhang, S. Ma, and S. Sclaroff, "MEEM: Robust tracking via multiple experts using entropy minimization," in *Proc. Eur. Conf. Comput. Vis.*, 2014, pp. 188–203.

Sangdoo Yun (S'17) received the B.S., M.S., and Ph.D. degrees in electrical engineering and computer science from Seoul National University, Seoul, South Korea, in 2010, 2013, and 2017, respectively.

He is currently a Post-Doctoral Researcher with the Automation and Systems Research Institute, Seoul National University. His current research interests include visual tracking, deep learning, and reinforcement learning.

Jongwon Choi (S'17) received the B.S. and M.S. degrees in electronic and electrical engineering from the Korea Advanced Institute of Science and Technology, Daejeon, South Korea, in 2012 and 2014, respectively. He is currently pursuing the Ph.D. degree in electrical and computer engineering with Seoul National University, Seoul, South Korea.

His current research interests include visual tracking, deep learning, and surveillance vision algorithm.

Youngjoon Yoo (S'17) received the B.S degree in electrical engineering and computer science from Seoul National University, Seoul, South Korea, in 2011, and the Ph.D. degree in electrical engineering and computer science from Seoul National University, Seoul, South Korea, in 2017.

He is currently a Research Engineer with CLOVA AI Research, Naver, Seoungnam-si, South Korea. His current research interests include machine learning, computer vision, deep generative model, and probabilistic inference.

Kimin Yun received the B.S. degree and the unified M.S. and Ph.D. degree in electrical engineering and computer science from Seoul National University, Seoul, South Korea, in 2010 and 2017, respectively.

He joined the SW & Contentes Research Laboratory, Electronics and Telecommunications Research Institute (ETRI), Daejeon, South Korea. His current research interests include machine learning, computer vision, visual event analysis, moving object detection in a moving camera, and video analysis.

Jin Young Choi (M'17) received the B.S., M.S., and Ph.D. degrees in control and instrumentation engineering from Seoul National University, Seoul, South Korea, in 1982, 1984, and 1993, respectively.

From 1984 to 1989, he joined the project of TDX switching system at the Electronics and Telecommunications Research Institute (ETRI), Daejeon, South Korea. From 1992 to 1994, he was with the Basic Research Department, ETRI, where he was a Senior Member of Technical Staff involved in the neural information processing system. Since 1994, he has been with Seoul National University, where he is currently a Professor with the School of Electrical Engineering. He is also with the Automation and Systems Research Institute, the Engineering Research Center for Advanced Control and Instrumentation, and the Automatic Control Research Center, Seoul National University. From 1998 to 1999, he was a Visiting Professor with the University of California at Riverside, Riverside, CA, USA. His current research interests include adaptive and learning systems, visual surveillance, motion pattern analysis, object detection and tracking, and pattern learning and recognition.