Action-Decision Networks for Visual Tracking with Deep Reinforcement Learning

Sangdoo Yun[1] Jongwon Choi[1] Youngjoon Yoo[2] Kimin Yun[3] and Jin Young Choi[1]
[1]ASRI, Dept. of Electrical and Computer Eng., Seoul National University, South Korea
[2]Graduate School of Convergence Science and Technology, Seoul National University, South Korea
[3]Electronics and Telecommunications Research Institute (ETRI), South Korea
{yunsd101, i0you200, jychoi}@snu.ac.kr, jwchoi.pil@gmail.com, kimin.yun@etri.re.kr

Abstract

This paper proposes a novel tracker which is controlled by sequentially pursuing actions learned by deep reinforcement learning. In contrast to the existing trackers using deep networks, the proposed tracker is designed to achieve a light computation as well as satisfactory tracking accuracy in both location and scale. The deep network to control actions is pre-trained using various training sequences and fine-tuned during tracking for online adaptation to target and background changes. The pre-training is done by utilizing deep reinforcement learning as well as supervised learning. The use of reinforcement learning enables even partially labeled data to be successfully utilized for semi-supervised learning. Through evaluation of the OTB dataset, the proposed tracker is validated to achieve a competitive performance that is three times faster than state-of-the-art, deep network–based trackers. The fast version of the proposed method, which operates in real-time on GPU, outperforms the state-of-the-art real-time trackers.

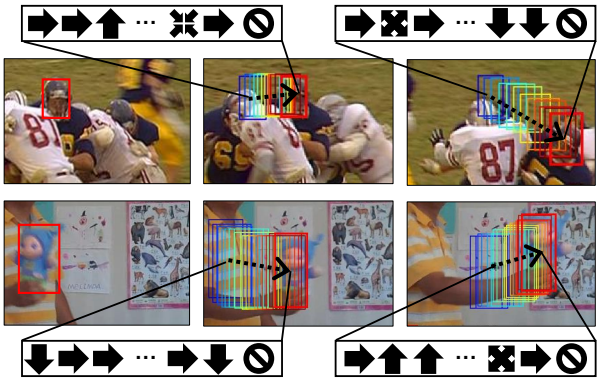


Figure 1: The concept of the proposed visual tracking controlled by sequential actions. The first column shows the initial location of the target, and the second and third columns show the iterative action flow to find the target bounding box in each frame. The sequential actions selected by the proposed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

1. Introduction

Visual tracking is one of the fundamental problems in the computer vision field. Finding the location of the target object is difficult because of several tracking obstacles such as motion blur, occlusion, illumination change, and background clutter. Conventional tracking methods [17, 42, 7, 15, 13] follow target objects using a low-level handcrafted feature. Although they achieve computational efficiency and comparable tracking performance, they are still limited in solving the above-mentioned obstacles because of their insufficient feature representation.

Recently, tracking methods [35, 14, 24] using convolutional neural networks (CNNs) have been proposed for robust tracking and vastly improved tracking performance with the help of rich feature representation. Several algorithms [35, 14] utilize pre-trained CNNs on a large-scale classification dataset such as ImageNet [26]. However, due to the gap between classification and tracking problem, the pre-trained CNN is not sufficient to solve the difficult tracking issues. Nam *et al*. [24] proposed a tracking-by-detection algorithm with CNNs trained with tracking video datasets such as [40, 20] and achieved the better performance compared to the traditional trackers. However, this approach usually focuses on improving the ability to distinguish the target and background using the appearance model and may thus overlook the following problems: (1) inefficient search algorithms that explore the region of interest and select the best candidate by matching with the tracking model, and (2) the need for a large amount of labeled tracking sequences for training and the inability to utilize unlabeled frames in a semi-supervised case.

In this work, to deal with the issues raised above, we propose a novel tracker to pursue the change of target by repetitive actions controlled by the proposed action-decision network (ADNet). The basic concept of the proposed visual tracking is depicted in Figure 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 convolutional neural network [4], 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 [31, 24]. In addition, since our method can precisely localize the target by selecting actions, post-processing such as bounding box regression [24] is not necessary.

We also propose a combined learning algorithm of supervised learning (SL) and reinforcement learning (RL) to train the ADNet. 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 pre-trained network in the SL stage is used as an initial network. We perform RL via tracking simulation using training sequences composed of sampled states, actions, and rewards. The network is trained with deep reinforcement learning based on policy gradient [38], using the rewards obtained during the tracking simulation. Even in the case where training frames are partially labeled (semi-supervised case), the proposed framework successfully learns the unlabeled frames by assigning the rewards according to the results of tracking simulation.

2. Related Work

2.1. Visual Object Tracking

As surveyed in [41, 29], various trackers have shown their performance and effectiveness on various tracking benchmarks [20, 18, 40]. The approach based on Tracking-by-Detection [10, 11, 1, 17] aims to build a discriminative classifier that distinguishes the target from the surrounding background. Typically these methods capture the target position by detecting most matching position using the classifier. Online boosting methods [10, 11] were proposed to update the discriminative model in online manner. Multiple instance learning (MIL) [1] and tracking-learning-detection (TLD) [17] methods were proposed to update tracking model robust to the noise.

Tracking methods based on correlation filter [2, 7, 13, 15, 5] 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*. [2] proposed a minimum output sum of squared error (MOSSE) filter and Henriques *et al*. [13] proposed kernelized correlation filters (KCF) with multi-channel features. Hong *et al*. [15] proposed the combined system employing short-term correlation tracker and long-term memory stores. Choi *et al*. [5, 6] proposed the integrated tracking system to handle the various type of correlation filters with attentional mechanism. To overcome the insufficient representation of the hand-crafted features, deep convolutional features are utilized in the correlation filters [8, 9] which have achieved the state-of-the-art performance. However, since they need to train various scale-wise filters to deal with the scale change and compute the deep features, they are much slower than the traditional correlation filter based methods.

Recently, CNN-based methods [36, 21, 22, 34, 14, 35, 31, 24, 12] have been proposed to learn the tracking models. Early attempts [36, 21, 22] were suffering from the data deficiency problem for training their networks. To solve the insufficient data problem, transferring methods [14, 35] were proposed by utilizing the pre-trained CNNs on a large-scale classification dataset such as ImageNet [26]. However, these methods still have a limitation due to the gap between the object classification and tracking domain. Recently proposed methods [31, 24, 12] overcome the gap by training their network with a large amount of tracking video datasets [40, 20, 29]. Held *et al*. [12] proposed the tracking algorithm which capture the target's location with deep regression networks. However, this method has difficulties to track the target when the target is moving too quickly or occlusion is happened since it has no online updating procedure. Tao *et al*. [31] and Nam *et al*. [24] proposed Tracking-by-Detection approach that distinguishes the target and the surrounding background using the trained CNNs and successfully achieved the state-of-the-art performance. However, these methods [31, 24] need computationally inefficient search algorithms, such as sliding window or candidate sampling.

2.2. Deep Reinforcement Learning

The goal of reinforcement learning (RL) is to learn a policy that decides sequential actions by maximizing the cumulative future rewards [30]. Recent trends [23, 32, 28, 27] in RL field is to combine the deep neural networks with RL algorithms by representing RL models such as value function or policy. By resorting of the deep features, many difficult problems such as playing Atari games [23] or Go [27] can be successfully solved in semi-supervised setting. Also, several methods were proposed to solve the computer vision problems, such as object localization [3] or action recognition [16], by employing the deep RL algorithms.

There are two popular approaches in deep RL algo-

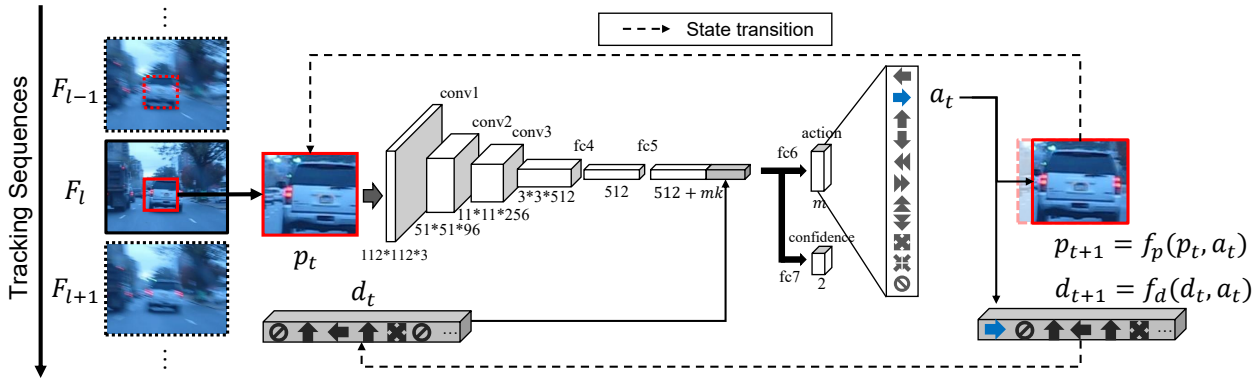


Figure 2: Architecture of the proposed network. The dashed lines indicate the state transition. In this example, the 'move right' action is selected to capture the target object. This action-decision process is repeated until finalize the location of the target in each frame.

rithms: Deep Q Networks (DQN) and policy gradient. DQN 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 [23]. Based on the DQN algorithm, various network architectures such as Double DQN [32] and DDQN [37] were proposed to improve performance and keep stability.

Policy gradient methods directly learn the policy by optimizing the deep policy networks with respect to the expected future reward using gradient descent. Williams *et al*. [38] proposed REINFORCE algorithm simply using the immediate reward to estimate the value of the policy. Silver *et al*. [28] proposed a deterministic algorithm to improve the performance and effectiveness of the policy gradient in high-dimensional action space. In the work of Silver *et al*. [27], it is shown that pre-training the policy networks with supervised learning before employing policy gradient can improve the performance. In tracking problem, we train the proposed network with supervised learning to learn the appearance characteristics of the target objects, and train action dynamics of the tracking target with reinforcement learning using policy gradient method.

3. Tracking Scheme Controlled by Actions

3.1. 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 Action-Decision networks (ADNets) shown in Figure 2 (Details are in Section 3.2). The proposed networks predict the action to chase the target from the position of the current tracker. The tracker is moved by the predicted action from current state, and then the next action is predicted from the moved position. By repeating this process over the test sequence, we solve the object tracking problem. The ADNet is pretrained by supervised learning (Section 4.1) as well as reinforcement learning (Section 4.2). During actual tracking, online adaptation (Section 4.3) is conducted.

3.2. Problem Settings

Basically our tracking strategy follows Markov Decision Process (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, the tracker is defined as an agent of which 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 bounding box location and size in each frame.

In every frame, the agent decides sequential actions until finalizing the target's position, and then, goes to the next frame. The state representation includes the appearance information at the bounding box of the target and the previous actions. The agent receives a reward for the final state of the frame l by deciding whether the agent succeed to track the object or not. The state and action are represented as $s_{t,l}$ and $a_{t,l}$ respectively, for $t = 1, ..., T_l$ and $l = 1, ..., L$ 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 l-th frame is transferred to the next frame, i.e., $s_{1,l+1} := s_{T_l,l}$. In the following except the sections 4.2 and 4.3, we omit the subscript l when we describe MDP in each frame for simplicity.

Action. The action space $\mathcal{A}$ consists of eleven types of actions including translation moves, scale changes, and stopping action as shown in Figure 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-dimensional vector with one-hot form.

Figure 3: The defined actions in our method.

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 $d_t \in \mathbb{R}^{110}$ represents the dynamics of actions denoted by a vector (called by "action dynamics vector" in the following) containing the previous k actions at t-th iteration. The patch p_t is pointed by 4-dimensional 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 pre-processing 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 concatenated past k action vectors. We store past k actions in the action dynamics vector $d_t = [\psi(a_{t-1}), ..., \psi(a_{t-k})]$, where $\psi(\cdot)$ denotes one-hot encoding function. Letting $k = 10$, d_t has 110 dimension since each action vector has 11 dimension.

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

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

where α is 0.03 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 represent the transition of action history. When the *'stop'* action is selected, we finalize the patch position for the target in the current frame, the agent will receive the reward, and then the resulting state is transferred to the initial state of the next frame.

Reward. The reward function is defined as $r(s)$ since the agent 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 $IoU(b_T, G)$ denotes overlap ratio of the terminal patch position b_T and the ground truth G of the target with intersection-over-union criterion. The tracking score z_t is defined as the terminal reward, $z_t = r(s_T)$, which will be used to update model in reinforcement learning.

3.3. Action-Decision Network

The pre-trained VGG-M model [4] is used to initialize our network. Small CNN models such as VGG-M [4] are more effective in the visual tracking problem than deep models [24]. As illustrated in Figure 2, our network has three convolutional layers $\{\text{conv1}, \text{conv2}, \text{conv3}\}$, which are identical to the convolutional layers of VGG-M networks. The next two fully connected layers $\{\text{fc4}, \text{fc5}\}$ are combined with the 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 $\{\text{fc6}, \text{fc7}\}$ predict the action probabilities and confidence score of the given state respectively. The parameter of the i-th layer is denoted by w_i and the whole network parameter by W.

The fc6 layer has 11 output units and is combined with softmax layer, which represent 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 Figure 2, the proposed network iteratively pursues the target position. The agent selects actions sequentially and updates states until finalizing the position of target. The final state is reached by selecting stop action or falling in the oscillation case. The oscillation case occurs, for example, when the sequential actions are obtained as $\{$*left*, *right*, *left*$\}$, which means the agent is coming back to the previous state. The confidence layer (fc7) with two output units produces the probability of target and background class for the given state s_t. The target probability $p(\text{target}|s_t; W)$ is used as the confidence score of the tracker at s_t. The confidence score is utilized for the online adaptation during tracking (Section 4.3).

4. Training of ADNet

In this section, we describe the training frameworks for ADNet. First, in offline manner, ADNet is pretrained by supervised learning (Section 4.1) and reinforcement learning (Section 4.2) using the training videos with the purpose of learning to track general objects. In supervised learning, the proposed network is trained to predict a proper action to a given state. In reinforcement learning, the proposed network is updated by performing tracking simulation on the

training sequence and utilizing action dynamics. After pre-training ADNet, online adaptation (Section 4.3) is applied to the network to accommodate the appearance changes or deformation of the target during tracking test sequences. In Section 4.4, the implementation details for training ADNet are described.

4.1. Training ADNet with Supervised Learning

In the supervised learning stage, the network parameters W_{SL}, $\{w_1, ..., w_7\}$, 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 $\{o_j^{(act)}\}$, class labels $\{o_j^{(cls)}\}$. 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 datasets provide video frames and the ground truth patch position and size. A sample patch p_j is obtained by adding Gaussian noise to the ground truth and the corresponding action label $o_j^{(act)}$ is assigned by,

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

where G is the ground truth patch and $\bar{f}(p, a)$ denotes the moved patch from p by the action a. The class label $o_j^{(cls)}$ corresponding to p_j is defined as the following,

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

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

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

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

4.2. Training ADNet with Reinforcement Learning

In the reinforcement learning stage, network parameters W_{RL}, ($\{w_1, ..., w_6\}$), except fc7 layer are trained. Training ADNet with RL in this section aims to improve the network by policy gradient approach [38]. 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-come-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.

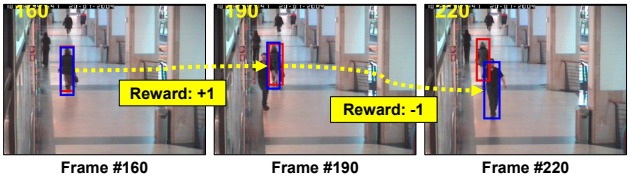


Figure 4: Illustration of the tracking simulation in semi-supervised case on *Walking2* sequence. Red boxes and blue boxes denote the ground truths and the predicted target's positions respectively. In this example, only the frames #160, #190, and #220 are annotated. Through the sequential actions, the agent 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.

The detail algorithm to train ADNet with RL is described in the supplementary material. During the training iterations, we first randomly pick a piece of training sequence $\{F_l\}_{l=1}^{\mathcal{L}}$ and the ground truths $\{G_l\}_{l=1}^{\mathcal{L}}$. We then perform the reinforcement learning via tracking simulation with the training image sequences annotated by ground truth. 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 $t = 1, ..., T_l$ and frame indices $l = 1, ..., \mathcal{L}$. The action $a_{t,l}$ for the state $s_{t,l}$ is assigned by,

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

where $p(a_{t,l}|s_{t,l})$ denotes the conditional action probability. When the tracking simulation is done, tracking scores $\{z_{t,l}\}$ are computed with the ground truths $\{G_l\}$. 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, which is defined as Eq. (3). By utilizing the tracking scores, the network parameters W_{RL} are updated by stochastic gradient ascent [38] to maximize the expected tracking scores as follows,

$$\Delta W_{RL} \propto \sum_{l}^{L}\sum_{t}^{T_l} \frac{\partial \log p(a_{t,l}|s_{t,l}; W_{RL})}{\partial W_{RL}} z_{t,l}. \tag{8}$$

Our framework can train ADNet even if the ground truths $\{G_l\}$ are partially given, which means the semi-supervised setting as shown in Figure 4. The supervised learning framework cannot learn the information of the unlabeled frames, however, the reinforcement learning can utilize the unlabeled frames in semi-supervised manner. In order to train ADNet in RL, the tracking scores $\{z_{t,l}\}$ should be determined, however, the tracking scores in the unlabeled sequences cannot be immediately determined. Instead, we assign the tracking scores to the reward obtained from the result of tracking simulation. In other works, 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 Figure 4.

4.3. Online Adaptation in Tracking

The proposed network is updated in online manner during tracking. This online adaptation can make the tracking algorithm more robust against the appearance changes or deformation. When updating ADNet, we fix the convolutional filters $\{w_1, w_2, w_3\}$ and fine-tune the fully-connected layers $\{w_4, ..., w_7\}$ because the convolutional layers would have generic tracking information whereas the fully-connected layers would have the video-specific knowledge. The detailed procedure of the proposed tracking and online adaptation scheme is described in the supplementary material. The tracking is performed by deciding sequential actions with the state-action probability $p(a|s; W)$. We adopt the online update adaptation of [24]. The online adaptation is done by fine-tuning of ADNet with a supervised learning using the temporal training samples generated during the tracking process. For the supervised learning, training samples with labels are required. For the labeling, we have to determine the ground truth. The tracked patch position determined by the network is used for the temporal ground truth. As similar to SL (Section 4.1), 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 $\{o_i^{act}\}$ and class labels $\{o_i^{cls}\}$. The labels are obtained via Eq. (4) and Eq. (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 tracker miss the target, then the *re-detection* 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 re-detected target position b^* is selected by

$$b^* = arg\max_{\tilde{b}_i} c(\tilde{b}_i), \tag{9}$$

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

4.4. Implementation Details

Pretraining ADNet. In each frame, we generated the training samples by adding Gaussian noise, whose mean is zero and covariance matrix is diag$((0.3w)^2, (0.3h)^2, (0.1w)^2, (0.1h)^2)$, to the ground truth position $G(= [x, y, w, h])$. When pretraining ADNet, we draw 250 samples in each frame. We set the learning rate to 0.0001 for convolutional layers (fc1-3) and 0.001 for fully-connected layers (fc4-7) [24], momentum to 0.9, weight decay to 0.0005, and mini-batch size to 128. For pretraining ADNet with $\mathcal{K}$ training videos, the number of training iteration is set to 300 for each video. In each iteration of the reinforcement learning, we randomly select the sequence of length $\mathcal{L}(= 10)$ for the tracking simulation.

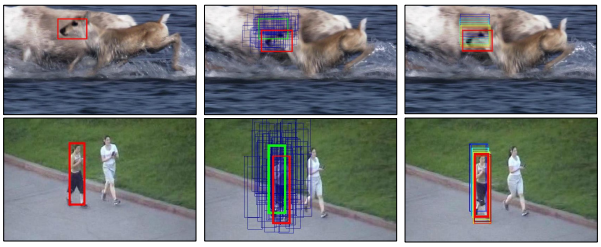
Online adaptation during tracking. For online adaptation, we only train the fully-connected layers (fc4-7) with the learning rate 0.001. We fine-tune ADNet with $\mathcal{T}_I(= 300)$ iterations at the first frame and $\mathcal{T}_O(= 30)$ iterations for the online adaptations. The online training is conducted for every $\mathcal{I}(= 10)$ frames and the training data are sampled from the past $\mathcal{J}(= 20)$ frames. For the *re-detection*, we draw $\mathcal{N}_{\text{det}}(= 256)$ target position candidates. In the online adaptation, $\mathcal{N}_{\text{I}}(= 3000)$ samples are generated in the first frame, and $\mathcal{N}_{\text{O}}(= 250)$ samples are generated in the frame whose confidence is above 0.5 during tracking. In addition, 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". 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 endures 3% performance degradation but achieves a real time speed around 4 times faster than the standard version of ADNet.

5. Experiments

We evaluated our method on the popular visual tracking benchmarks, Object Tracking Benchmark (OTB) [39, 40], comparing 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 [33]. In our settings, ADNet and ADNet-fast run at 3 fps and 15 fps on the GPU, respectively. The program and benchmark results were uploaded online[1].

We evaluated our method on two OTB datasets: OTB-50 [39], which has 50 video sequences, and OTB-100 [40], which has 100 video sequences including OTB-50. In order to pre-train ADNet, we used 360 training videos from VOT2013 [19], VOT2014 [20], VOT2015 [18], and ALOV300 [29], 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 [39]. The center location error measures the distance

[1] https://sites.google.com/view/cvpr2017-adnet



(a) Previous frame (b) Tracking-by-Detection method (c) Our method

Figure 5: Searching strategy comparison of the searching strategy between the existing tracking-by-detection approach [24] (second column) and the proposed method (third column) on the *Deer* and *Jogging-1* sequences. Detail explanation is in Section. 5.1

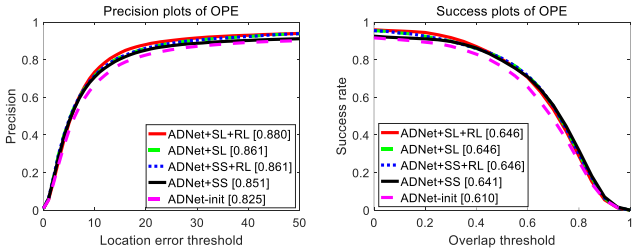
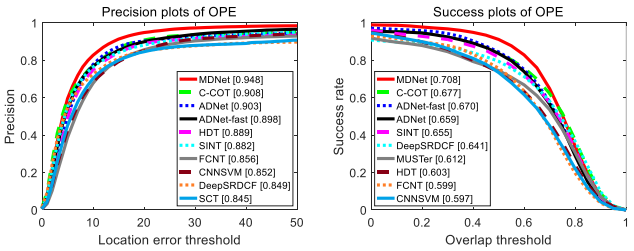


Figure 6: Self-comparison results of experiments on OTB-100. The scores in the legend indicate the mean precisions when the location error threshold is 20 pixel for the precision plots and area-under-curve (AUC) for the success plots.

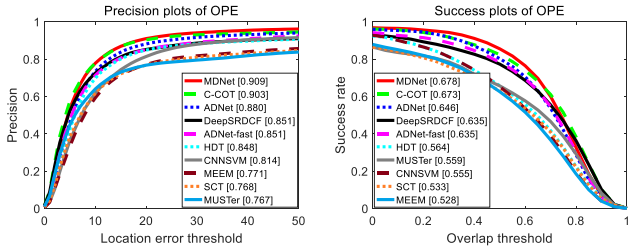
between the center of the tracked frame and the ground truth and the bounding box overlap ratio measures the Intersection-over-Union (IOU) ratio between the tracked bounding box and the ground truth.

5.1. Analysis

Self-comparison. To verify the effectiveness of the components of ADNet, we conducted four variants of ADNet and evaluated them using OTB-100. We first conducted the baseline ‘’‘ADNet-init”, which is not pre-trained and simply uses the initial parameters. In ADNet-init, the parameters of convolutional networks (conv1-3) are initialized with the VGG-M [4] model, and the fully-connected layers (fc4-7) are initialized with random noises. “ADNet+SL” is the pre-trained models with supervised learning using fully labeled frames of the training sequences. “ADNet+SS” is trained using partially labeled data in the semi-supervised (SS) settings. In the training of ADNet+SS, the ground truth annotation is provided only every 10 frames. Then we conducted “ADNet+SL+RL” and “ADNet+SS+RL” by training ADNet+SL and ADNet+SS using reinforcement learning (RL), respectively. ADNet+SL+RL is the final version of the proposed method. The precision and success rate of the self-comparisons are illustrated in Figure 6. By conducting



(a) OTB-50



(b) OTB-100

Figure 7: Precision and success plots on OTB-50 [39] and OTB-100 [40]. Only the top 10 trackers are presented.

Table 1: 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 [24]	90.9%	0.678	< 1	O
	C-COT [9]	90.3%	0.673	< 1	O
	DeepSRDCF [8]	85.1%	0.635	< 1	O
	HDT [25]	84.8%	0.564	5.8	O
	MUSTer [15]	76.7%	0.528	3.9	X
Real-time	MEEM [42]	77.1%	0.528	19.5	X
	SCT [5]	76.8%	0.533	40.0	X
	KCF [13]	69.7%	0.479	223	X
	DSST [7]	69.3%	0.520	25.4	X
	GOTURN [12]	56.5%	0.425	125	O

SL, ADNet+SL and ADNet+SS achieved 3.6% and 2.6% improvement in precision performance improvement, respectively, compared to ADNet-init. In the semi-supervised case, the precision is 1.0% 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 semi-supervised case but also the supervised case.

Analysis on the actions. In the experiment, the ratio of the frames using re-detection to the whole frames was around 9%, and the ratio of the frames requiring more than five actions to capture the target to the whole frames was only around 4%, that is, most of the frames require fewer than five actions to pursue the target in each frame. Figure 5 illustrates the efficiency of the proposed method pursuing the target by sequential actions, compared to the existing tracker based on tracking-by-detection strategy [24]. In Fig-

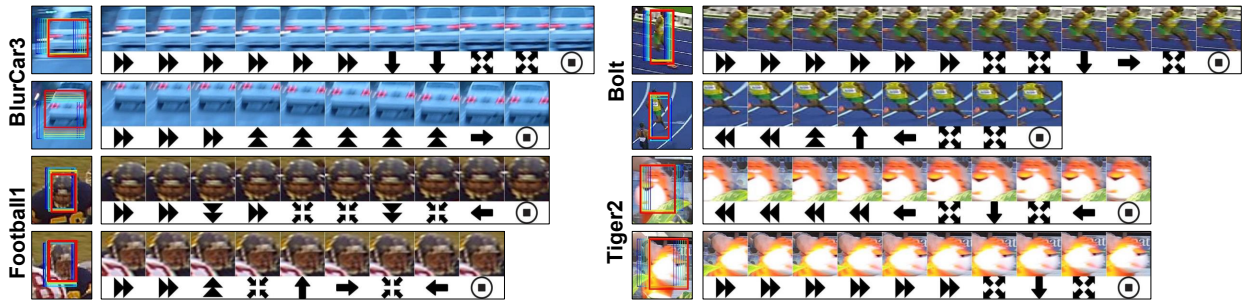


Figure 8: Examples of sequential actions selected by ADNet on *BlurCar3*, *Bolt*, *Football1*, and *Tiger2* sequences. The state transitions in the proposed tracking algorithm are presented with the image patches and the corresponding actions.

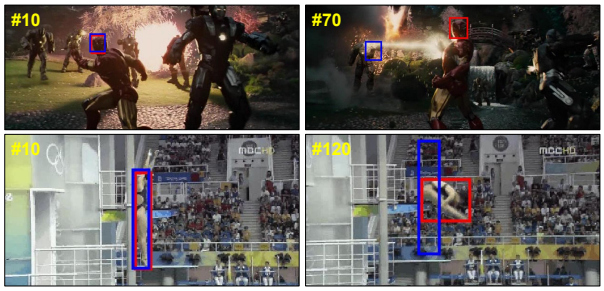


Figure 9: Failure cases of the proposed method on *Ironman* and *Diving* sequences. Blue and red bounding boxes indicate the ground truths and the tracking results of ADNet, respectively.

ure 5 (b), the green, red, and blue boxes denote the previous target position, the current target position, and the generated target candidates respectively. Figure 5 (c) shows the tracking process of ADNet by selecting sequential actions. The average number of searching steps including the required actions and the candidates by re-detection is 28.26 per frame, which is much smaller than that of state-of-the-art trackers such as MDNet [24](= 256 per frame).

5.2. State-of-the-art Comparison

We compared ADNet in a comprehensive comparison with 13 state-of-the-art trackers including MDNet [24], C-COT [9], GOTURN[2] [12], HDT [25], DeepSRDCF [8], SINT [31], FCNT [34], SCT [5], MUSTer [15], CNN-SVM [14], MEEM [42], DSST [7], and KCF [13]. Figure 7 shows the plots of precision and success rate based on center location error and overlap ratio respectively and Table 1 summarizes the comparison of tracking performance with computational speed (fps). The proposed method shows comparable performance with the state-of-the-art trackers, MDNet [24] and C-COT [9] in both precision and success rate. The proposed method is much efficient in computation, where ADNet is about three times faster than MDNet and C-COT. ADNet-fast, the fast version of ADNet, has a 3% performance degradation but runs in real-time (15 fps) and shows performance comparable with those of other CNN-based trackers such as DeepSRDCF [8] and HDT [25]. As shown in Table. 1, ADNet-fast achieved the best performance among the real-time tracking algorithms. Figure 8 illustrates examples of the sequential actions selected by ADNet. The bounding box flow from the initial position (blue) to the captured target position (red) is shown in the left-most columns and the sequential transitions of the state are represented by the image patches and the selected actions. Figure 9 presents a few failure cases of the proposed method. ADnet failed to follow the abrupt movement of the target in *Ironman* sequence, and the proposed actions could not adapt to the dramatic aspect ratio change in *Diving* sequence.

[2]We evaluated the GOTURN [12] on the OTB dataset using the author's code.

6. Conclusion

In this paper, we have proposed a novel tracker controlled by action-decision network (ADNet), which pursues the target object by sequential actions iteratively. To the best of our knowledge, it is the first attempt to adopt the tracking strategy controlled by pursuing actions trained by deep reinforcement learning. Action-based tracking makes a significant contribution to the reduction of computation complexity in tracking. In addition, reinforcement learning makes the use of partially labeled data possible, which could greatly benefit actual applications. According to the evaluation results, the proposed tracker achieves a state-of-the-art performance in 3 fps, which is three times faster than the existing deep network-based trackers adopting a tracking-by-detection strategy. Furthermore, the fast version of the proposed tracker achieves a real-time speed (15 fps) with an accuracy that outperforms state-of-the-art real-time trackers.

Acknowledgement. This work was partly supported by the ICT R&D program of MSIP/IITP (No.B0101-15-0552, Development of Predictive Visual Intelligence Technology and No.B0101-15-0266, Development of High Performance Visual BigData Discovery Platform), the SNU-Samsung Smart Campus Research Center at Seoul National University, and Brain Korea 21 Plus Project.

References

[1] B. Babenko, M.-H. Yang, and S. Belongie. Robust object tracking with online multiple instance learning. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(8):1619–1632, 2011. 2

[2] D. S. Bolme, J. R. Beveridge, B. A. Draper, and Y. M. Lui. Visual object tracking using adaptive correlation filters. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 2544–2550. IEEE, 2010. 2

[3] J. C. Caicedo and S. Lazebnik. Active object localization with deep reinforcement learning. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2488–2496, 2015. 2

[4] K. Chatfield, K. Simonyan, A. Vedaldi, and A. Zisserman. Return of the devil in the details: Delving deep into convolutional nets. *arXiv preprint arXiv:1405.3531*, 2014. 2, 4, 7

[5] J. Choi, H. Jin Chang, J. Jeong, Y. Demiris, and J. Young Choi. Visual tracking using attention-modulated disintegration and integration. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4321–4330, 2016. 2, 7, 8

[6] J. Choi, H. Jin Chang, S. Yun, T. Fischer, Y. Demiris, and J. Young Choi. Attentional correlation filter network for adaptive visual tracking. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017. 2

[7] M. Danelljan, G. Häger, F. Khan, and M. Felsberg. Accurate scale estimation for robust visual tracking. In *British Machine Vision Conference, Nottingham, September 1-5, 2014*. BMVA Press, 2014. 1, 2, 7, 8

[8] M. Danelljan, G. Hager, F. Shahbaz Khan, and M. Felsberg. Convolutional features for correlation filter based visual tracking. In *Proceedings of the IEEE International Conference on Computer Vision Workshops*, pages 58–66, 2015. 2, 7, 8

[9] M. Danelljan, A. Robinson, F. Shahbaz Khan, and M. Felsberg. Beyond correlation filters: Learning continuous convolution operators for visual tracking. In *ECCV*, 2016. 2, 7, 8

[10] H. Grabner, M. Grabner, and H. Bischof. Real-time tracking via on-line boosting. In *BMVC*, volume 1, page 6, 2006. 2

[11] H. Grabner, C. Leistner, and H. Bischof. Semi-supervised on-line boosting for robust tracking. In *European conference on computer vision*, pages 234–247. Springer, 2008. 2

[12] D. Held, S. Thrun, and S. Savarese. Learning to track at 100 fps with deep regression networks. *arXiv preprint arXiv:1604.01802*, 2016. 2, 7, 8

[13] J. F. Henriques, R. Caseiro, P. Martins, and J. Batista. High-speed tracking with kernelized correlation filters. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 37(3):583–596, 2015. 1, 2, 7, 8

[14] S. Hong, T. You, S. Kwak, and B. Han. Online tracking by learning discriminative saliency map with convolutional neural network. *arXiv preprint arXiv:1502.06796*, 2015. 1, 2, 8

[15] 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 *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 749–758, 2015. 1, 2, 7, 8

[16] D. Jayaraman and K. Grauman. Look-ahead before you leap: end-to-end active recognition by forecasting the effect of motion. *arXiv preprint arXiv:1605.00164*, 2016. 2

[17] Z. Kalal, K. Mikolajczyk, and J. Matas. Tracking-learning-detection. *IEEE transactions on pattern analysis and machine intelligence*, 34(7):1409–1422, 2012. 1, 2

[18] M. Kristan, J. Matas, A. Leonardis, M. Felsberg, L. Cehovin, G. Fernandez, T. Vojir, G. Hager, G. Nebehay, and R. Pflugfelder. The visual object tracking vot2015 challenge results. In *Proceedings of the IEEE International Conference on Computer Vision Workshops*, pages 1–23, 2015. 2, 6

[19] M. Kristan, R. Pflugfelder, A. Leonardis, J. Matas, F. Porikli, L. Cehovin, G. Nebehay, G. Fernandez, T. Vojir, A. Gatt, et al. The visual object tracking vot2013 challenge results. In *Proceedings of the IEEE International Conference on Computer Vision Workshops*, pages 98–111, 2013. 6

[20] M. Kristan, R. Pflugfelder, A. Leonardis, J. Matas, L. Čehovin, G. Nebehay, T. Vojir, G. Fernandez, and A. Lukežič. The visual object tracking vot2014 challenge results. In *Visual Object Tracking Workshop 2014 at ECCV2014*, 2014. 1, 2, 6

[21] H. Li, Y. Li, and F. Porikli. Robust online visual tracking with a single convolutional neural network. In *Asian Conference on Computer Vision*, pages 194–209. Springer, 2014. 2

[22] H. Li, Y. Li, and F. Porikli. Deeptrack: Learning discriminative feature representations online for robust visual tracking. *IEEE Transactions on Image Processing*, 25(4):1834–1848, 2016. 2

[23] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013. 2, 3

[24] H. Nam and B. Han. Learning multi-domain convolutional neural networks for visual tracking. *arXiv preprint arXiv:1510.07945*, 2015. 1, 2, 4, 6, 7, 8

[25] Y. Qi, S. Zhang, L. Qin, H. Yao, Q. Huang, and J. L. M.-H. Yang. Hedged deep tracking. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, 2016. 7, 8

[26] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, et al. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015. 1, 2

[27] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016. 2, 3

[28] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller. Deterministic policy gradient algorithms. In *ICML*, 2014. 2, 3

[29] A. W. Smeulders, D. M. Chu, R. Cucchiara, S. Calderara, A. Dehghan, and M. Shah. Visual tracking: An experimental survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(7):1442–1468, 2014. 2, 6

[30] R. S. Sutton. *Introduction to reinforcement learning*, volume 135. 2

[31] R. Tao, E. Gavves, and A. W. Smeulders. Siamese instance search for tracking. *arXiv preprint arXiv:1605.05863*, 2016. 2, 8

[32] H. Van Hasselt, A. Guez, and D. Silver. Deep reinforcement learning with double q-learning. *CoRR, abs/1509.06461*, 2015. 2, 3

[33] A. Vedaldi and K. Lenc. Matconvnet – convolutional neural networks for matlab. In *Proceeding of the ACM Int. Conf. on Multimedia*, 2015. 6

[34] L. Wang, W. Ouyang, X. Wang, and H. Lu. Visual tracking with fully convolutional networks. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 3119–3127, 2015. 2, 8

[35] N. Wang, S. Li, A. Gupta, and D.-Y. Yeung. Transferring rich feature hierarchies for robust visual tracking. *arXiv preprint arXiv:1501.04587*, 2015. 1, 2

[36] N. Wang and D.-Y. Yeung. Learning a deep compact image representation for visual tracking. In *Advances in neural information processing systems*, pages 809–817, 2013. 2

[37] Z. Wang, N. de Freitas, and M. Lanctot. Dueling network architectures for deep reinforcement learning. *arXiv preprint arXiv:1511.06581*, 2015. 3

[38] R. J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992. 2, 3, 5

[39] Y. Wu, J. Lim, and M.-H. Yang. Online object tracking: A benchmark. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2411–2418, 2013. 6, 7

[40] Y. Wu, J. Lim, and M.-H. Yang. Object tracking benchmark. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 37(9):1834–1848, 2015. 1, 2, 6, 7

[41] H. Yang, L. Shao, F. Zheng, L. Wang, and Z. Song. Recent advances and trends in visual tracking: A review. *Neurocomputing*, 74(18):3823–3831, 2011. 2

[42] J. Zhang, S. Ma, and S. Sclaroff. Meem: robust tracking via multiple experts using entropy minimization. In *European Conference on Computer Vision*, pages 188–203. Springer, 2014. 1, 7, 8