Deep Reinforcement Learning for Visual Object Tracking in Videos

Da Zhang[1], Hamid Maei[2], Xin Wang[1], and Yuan-Fang Wang[1]

[1]Department of Computer Science, University of California at Santa Barbara
[2]Samsung Research America
{dazhang, xwang, yfwang}@cs.ucsb.edu hamid.maei@samsung.com

Abstract

In this paper we introduce a fully end-to-end approach for visual tracking in videos that learns to predict the bounding box locations of a target object at every frame. An important insight is that the tracking problem can be considered as a sequential decision-making process and historical semantics encode highly relevant information for future decisions. Based on this intuition, we formulate our model as a recurrent convolutional neural network agent that interacts with a video overtime, and our model can be trained with reinforcement learning (RL) algorithms to learn good tracking policies that pay attention to continuous, inter-frame correlation and maximize tracking performance in the long run. The proposed tracking algorithm achieves state-of-the-art performance in an existing tracking benchmark and operates at frame-rates faster than real-time. To the best of our knowledge, our tracker is the first neural-network tracker that combines convolutional and recurrent networks with RL algorithms.

1. Introduction

Given some object of interest marked in one frame of a video, the goal of *single-object tracking* is to locate this object in subsequent video frames, despite object movement, changes in the camera's viewpoint and other incidental environmental variations such as lighting and shadows. Single-object tracking finds immediate applications in many important scenarios such as autonomous driving, unmanned aerial vehicle, security surveillance, etc.

Despite the success of traditional trackers based on low-level, hand-crafted features [2, 8, 23]; models based on deep convolutional neural network (CNN) have dominated recent visual tracking research [20, 9, 3]. The success of all these models largely depends on the capability of CNN to learn a good feature representation for the tracking target. In order to predict the target location in a new frame, either a search-and-classify [20] or crop-and-regress [9, 3] approach is applied. In that sense, although the representation power of CNN is exploited to capture *spatial* features, only limited manual *temporal* constraints are added in these frameworks, *e.g.*, new target lies in the spatial vicinity of the previous prediction. Unfortunately, for a busy scene with multiple occluding objects, short-term cues of correlating temporally close objects can often fail to account for multiple targets and mutual occlusion. Hence, how to harness the power of deep-learning models to automatically learn both *spatial* and *temporal* constraints, especially with longer-term information aggregation and disambiguation, should be fully explored.

Inspired by the successful works that have applied recurrent neural network (RNN) on computer vision tasks such as video classification and visual attention [4, 19]. We explore and investigate a more general strategy to develop a novel visual tracking approach based on recurrent convolutional networks. The major intuition behind our method is that the historical visual semantics and tracking proposals encode pertinent information for future predictions and can be modeled as a recurrent convolutional network. However, unlike video classification or visual attention where only high-level semantic or single-step predictions are needed, visual tracking requires continuous and accurate predictions in both spatial and temporal domain over a long period of time, and thus, requires a novel network architecture design as well as proper training algorithms.

In this work, we formulate the visual tracking problem as a sequential decision-making process and propose a novel framework, referred to as *Deep RL Tracker (DRLT)*, which processes video frames as a whole and directly outputs location predictions of the target in each frame. Our model integrates convolutional network with recurrent network (Figure 1), and builds up a spatial-temporal representation of the video. It fuses past recurrent states with current visual features to make predictions of the target object's location over time. We describe an end-to-end RL algorithm that allows

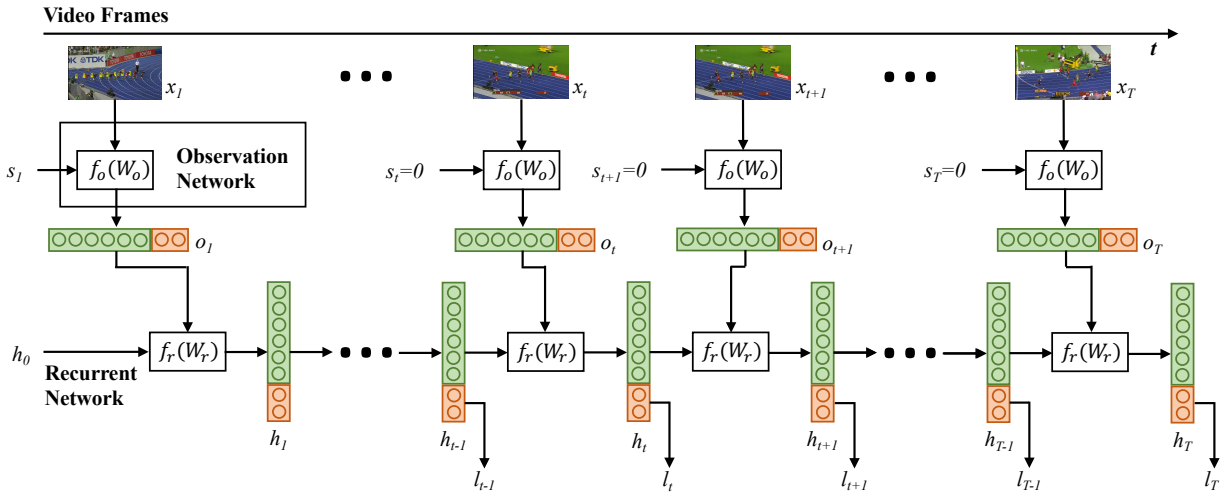

Figure 1: Overview of our Deep RL Tracker: At each timestep t, the **observation network** takes an image x_t and a location vector s_t as input and computes a feature representation o_t, where s_1 is the ground-truth location at the first frame and $s_t = 0$ otherwise. The **recurrent network** takes o_t as input, combining with the previous hidden state h_{t-1}, and generates new hidden state h_t. The predicted target location is directly extracted from h_t. During training, the agent will receive a reward signal r_t for each prediction. The basic RNN iteration is repeated for a variable number of steps and the cumulative rewards $R = \sum_{t=1}^{T} r_t$ will be used to update network parameters such that the long-term tracking performance is maximized.

the model to be trained to maximize tracking performance in the long run. This procedure uses backpropagation to train the nueral-network components and REINFORCE algorithm [28] to train the policy network.

Our algorithm augments traditional CNN with a recurrent convolutional model learning *spatial-temporal* representations and RL to maximize long-term tracking performance. The main contributions of our work are:

- We propose and develop a novel convolutional recurrent neural network model for visual tracking. The proposed method directly leverages the power of deep-learning models to automatically learn both spatial and temporal constraints.

- Our framework is trained end-to-end with deep RL algorithms, in which the model is optimized to maximize a tracking performance measure in the long run.

- Our model is trained fully off-line. When applied to online tracking, only a single forward pass is computed and no online fine-tuning is needed, allowing us to run at frame-rates beyond real-time.

- Our extensive experiments demonstrate the outstanding performance of our tracking algorithm compared to the state-of-the-art techniques in OTB [29] public tracking benchmark.

We claim that recurrent convolutional network plus RL algorithm is another useful deep-learning framework apart from CNN-based trackers. It has the potential of developing into a much robust and accurate tracker given that it pays explicit attention to temporal correlation and a long-term reward mechanism through RL.

The rest of the paper is organized as follows. We first review related work in Section 2, and discuss our RL approach for visual tracking in Section 3.1. Section 3.2 describes our end-to-end optimization algorithm, and Section 4 demonstrates the experimental results using a standard tracking benchmark.

2. Related Work

2.1. Visual Object Tracking

Visual Tracking is a fundamental problem in computer vision that has been actively studied for decades. Many methods have been proposed for single-object tracking. For a systematic review and comparison, we refer the readers to a recent benchmark and a tracking challenge report [29, 16].

Classification-based trackers. Trackers for generic object tracking often follows a tracking-by-classification methodology [14, 26]. A tracker will sample "foreground" patches near the target object and "background" patches farther away from the target. These patches are then used to train a foreground-background classifier, and this classifier is used to score potential patches in the next frame to es-

timate the new target location. Usually, the classifier is first trained off-line and fine-tuned during online tracking. Many neural-network trackers following this approach [12, 20, 25, 32] have surpassed traditional trackers [2, 8, 23], and achieved state-of-the-art performance [20, 16]. Unfortunately, these trackers are inefficient at run-time since neural networks are very slow to train in an online fashion. Another drawback of such a design is that it does not fully utilize all video information, particularly explicit temporal correlation.

Regression-based trackers. Some recent works [9, 3] have attempted to treat tracking as a regression instead of classification problem. David *et al.* [9] trained a CNN to regress directly from two images to the location in the second image of the object shown in the first image. Luca *et al.* [3] proposed a fully-convolutional siamese network to track objects in videos. These deep-learning methods can run at frame-rates beyond real time while maintaining state-of-the-art performance. However, they only extract features independently from each video frame and only perform comparison between two consecutive frames, prohibiting them from fully utilizing longer-term contextual and temporal information.

Recurrent-neural-network trackers. Several recent works [13, 6] have sought to train recurrent neural networks for the problem of visual tracking. Gan *et al.* [6] trained an RNN to predict the absolute position of the target in each frame and Kahou *et al.* [13] similarly trained an RNN for tracking using the attention mechanism. Although they brought good intuitions from RNN, these methods have not yet demonstrated competitive results on modern benchmark.

Another related work to ours is [21]. They proposed a spatially supervised recurrent convolutional neural network in which a YOLO network [22] is applied on each frame to produce object detections and a recurrent neural network is used to directly regress YOLO detections. Our framework does not need any supervision from other detection module and is more general and flexible.

2.2. Deep Reinforcement Learning

RL is a learning method based on trial and error, where an agent does not necessarily have *a prior* knowledge about what is the correct action to take. It learns interactively from rewards fed back from the environments. In order to maximize the expected rewards in the long term, the agent learns the best policy.

We draw inspiration from recent approaches that have used REINFORCE [28] to learn task-specific policies. Mnih *et al.* [19] and Ba *et al.* [1] learned spatial attention policies for image classification, and Xu *et al.* [31] for image caption generation. Our work is similar to the attention model described in [19], but we designed our own network architecture specially tailored for solving the visual tracking problem by combining CNN, RNN and RL algorithms.

Our proposed framework directly apply RNN on top of frame-level CNN features, paying direct attention to both *spatial* and *temporal* constraints, and the full framework is trained off-line with REINFORCE algorithm in an end-to-end manner. Due to its run-time simplicity, our tracker runs at frame-rates beyond real-time while maintaining state-of-the-art performance. We will describe our framework in detail in Section 3.

3. Deep RL Tracker (DRLT)

Our goal is to take a sequence of video frames and output target object locations at each frame. We formulate our tracking algorithm as a sequential decision-making process of a goal-oriented agent interacting with the visual environment. Figure 1 shows our model structure. At each point in time, the agent extracts representative features from a video frame, integrates information over time, and decides how to take actions accordingly. The agent receives a scalar reward signal at each timestep, and the goal of the agent is to maximize the total long-term rewards. Hence, it must learn to effectively utilize these temporal observations to reason on the moving trajectory of the object.

3.1. Architecture

The model consists of two major components: an observation network (Section 3.1.1), and a recurrent network (Section 3.1.2). The *observation network* encodes representations of video frames. The *recurrent network* integrates these observations over time and predicts the bounding box location in each frame. We now describe each of these in more detail. Later in Section 3.2, we explain how we use a combination of backpropagation and REINFORCE to train the model in an end-to-end fashion.

3.1.1 Observation Network

As shown in Figure 1, the observation network f_o, parameterized by W_o, observes a single video frame x_t at each timestep. It encodes the frame into a feature vector i_t, concatenates a location vector s_t and provides the feature and location combo (denoted as o_t) as input to the recurrent network.

The feature vector i_t is typically computed with a sequence of convolutional, pooling, and fully connected layers to encode information about *what* was seen in this frame. The importance of s_t are two folds: When the ground-truth bounding box location is known, such as the first frame in a given sequence, s_t is directly set to be the normalized location coordinate $(x, y, w, h) \in [0, 1]$, serving as a strong supervising guide for further inferences. Otherwise, s_t is

padded with zero and only the feature information i_t is incorporated by the recurrent network.

The concatenation of i_t and s_t allows the recurrent network to directly encode image features as well as location predictions, and it is also easier for location regression.

3.1.2 Recurrent Network

The recurrent network f_r, parameterized by W_r, forms the core of the learning agent. As can be seen in Figure 1, at one single timestep t, the observation feature vector o_t is fed into a recurrent network, and the recurrent network updates its internal hidden state h_t based on the previous hidden state h_{t-1} and the current observation feature vector o_t:

$$h_t = f_r(h_{t-1}, o_t; W_r) \tag{1}$$

where f_r is a recurrent transformation function and we use LSTM [11] in our network.

Importantly, the network's hidden state h_t models temporal hypotheses about target object locations. Since o_t is a concatenation of the image feature and the location signal, h_t directly encodes information about both *where* in the frame an object was located as well as *what* was seen.

As the agent reasons on a video, it outputs the location of target object l_t at each timestep t. $l_t = (x, y, w, h)$ where (x, y) represent the coordinates of the bounding box center relative to the width and height of the image, respectively. The width and height of the bounding box are also relative to those of the image, consequently, $(x, y, w, h) \in [0, 1]$.

The predicted location l_t is directly extracted from the last four elements of h_t denoted as μ_t, such that the agent's decision is a function of its past observations and their predicted locations. At training time, l_t is sampled from a multi-variate Gaussian distribution with a mean of μ_t and a fixed variance; at test time, the maximum a posteriori estimate is used.

Figure 1 further illustrates the roles of each component as well as the corresponding inputs and outputs with an example of a forward pass through the network.

3.2. Training

Training this network to maximize the overall tracking performance is a non-trivial task, and we leverage the REINFORCE algorithm [28] from the RL community to solve this problem.

3.2.1 Reward Function

During training, the agent will receive a reward signal r_t from the environment after executing an action at time t. In this work, we explore two different reward definitions in different training phases. One is

$$r_t = -avg(|l_t - g_t|) - max(|l_t - g_t|) \tag{2}$$

where l_t is the location outputted by the recurrent network, g_t is the target ground truth at time t, $avg(\cdot)$ and $max(\cdot)$ compute the pixel-wise mean and maximum. The other reward is

$$r_t = \frac{|l_t \cap g_t|}{|l_t \cup g_t|} \tag{3}$$

where the reward is computed as the intersection area divided by the union area (IoU) between l_t and g_t.

The training objective is to maximize the sum of the reward signals: $R = \sum_{t=1}^{T} r_t$. By definition, the reward in Equation 2 and Equation 3 both measure the closeness between predicted location l_t and ground-truth location g_t. We use the reward definition in Equation 2 in the early stage of training, while using the reward definition in Equation 3 in the late stage of training to directly maximize the IoU between the prediction l_t and ground-truth g_t.

3.2.2 Gradient Approximation

Our network is parameterized by $W = \{W_o, W_r\}$ and we aim to learn these parameters to maximize the total tracking reward the agent can expect in the long run. More specifically, the objective of the agent is to learn a policy function $\pi(l_t|z_{1:t}; W)$ with parameters W that, at each step t, maps the history of past interactions with the environment $z_{1:t} = x_1, l_1, x_2, l_2, ..., x_{t-1}, l_{t-1}, x_t$ (a sequence of past observations and actions taken by the agent) to a distribution over actions for the current timestep. Here, the policy π is defined by our neural network architecture, and the history of interactions $z_{1:t}$ is summarized in the hidden state h_t. For simplicity, we will use $Z_t = z_{1:t}$ to indicate all histories up to time t, thus, the policy function can be written as $\pi(l_t|Z_t; W)$.

To put it in a formal way, the policy of the agent $\pi(l_t|Z_t; W)$ induces a distribution over possible interactions Z_t and we aim to maximize the total reward R under this distribution, thus, the objective is defined as:

$$G(W) = E_{p(z_{1:T};W)}[\sum_{t=1}^{T} r_t] = E_{p(Z_T;W)}[R] \tag{4}$$

where $p(z_{1:T}; W)$ is the distribution over possible interactions parameterized by W.

This formulation involves an expectation over high-dimensional interactions which is hard to solve in traditional supervised manner. Here, we bring techniques from the RL community to solve this problem, as shown in [28], the gradient can be first simplified by taking the derivative over log-probability of the policy function π:

$$\nabla_W G = \sum_{t=1}^{T} E_{p(Z_T;W)}[\nabla_W \ln \pi(l_t|Z_t; W) R] \tag{5}$$

and the expectation can be further approximated by an episodic algorithm: since the action is drawn from probabilistic distributions, one can execute the same policy for many episodes and approximate expectation by taking the average, thus

$$\nabla_W G \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \nabla_W \ln \pi(l_t^i | Z_t^i; W) R^i \quad (6)$$

where R^is are cumulative rewards obtained by running the current policy π for N episodes, $i = 1...N$.

The above training rule is known as the episodic REINFORCE [28] algorithm, and it involves running the agent with its current policy to obtain samples of interactions and then updating parameters W of the agent such that the log-probability of chosen actions that have led to high overall rewards is increased.

In practice, although Equation 6 computes a good estimation of the gradient $\nabla_W G$, when applied to train the deep RL tracker, the training process is hard to converge due to the high variance of this gradient estimation. Thus, in order to obtain an unbiased low-variance gradient estimation, a common method is to subtract a reinforcement baseline from the cumulative rewards R:

$$\nabla_W G \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \nabla_W \ln \pi(l_t^i | Z_t^i; W)(R_t^i - b_t) \quad (7)$$

where b_t is called reinforcement baseline in the RL literature, it is natural to select $b_t = E_\pi[R_t]$, and this form of baseline is known as the value function [24]. This estimation maintains the same expectation with Equation 6 while sufficiently reduces the variance.

3.2.3 Training with Backpropagation

The only remaining part to compute the gradient in Equation 7 is to compute the gradient over log-probability of the policy function $\nabla_W \ln \pi(l_t | Z_t; W)$. To simplify notation, we focus on one single timestep and omit usual unit index subscript throughout. In our network design, the policy function π outputs the target location l which is drawn from a Gaussian distribution centered at μ with fixed variance σ, and μ is the output of the deep RL tracker parameterized by W. The density function g determining the output l on any single trial is given by:

$$g(l, \mu, \delta) = \frac{1}{(2\pi)^{\frac{1}{2}} \sigma} e^{-\frac{(l-\mu)^2}{2\sigma^2}} \quad (8)$$

Based on REINFORCE algorithm [28], the gradient of the policy function with respect to μ is given by the gradient of the density function:

$$\nabla_\mu \ln \pi = \frac{\partial \ln g}{\partial \mu} = \frac{l - \mu}{\sigma^2} \quad (9)$$

since μ is the output of deep RL tracker parameterized by W, the gradients with respect to network weights W can be easily computed by standard backpropagation.

3.2.4 Overall Procedure

The overall procedure of our training algorithm is presented in Algorithm 1. The network parameters W are first randomly initialized to define our initial policy. Then, we take first T frames from one training video to be the input of our network. We execute current policy N times, compute gradients and update network parameters. Next, we take consecutive T frames from the same video and apply the same training procedure. We repeat this for all training videos in our dataset, and we stop when we reach the maximum number of epochs or the cumulative reward ceases to increase.

Algorithm 1 Deep RL Tracker training algorithm

Input: Training videos $\{v_1, ..., v_M\}$ with ground-truth
Output: Network weights W

```
Randomly initialize weights W_o and W_r
Start from the first frame in training dataset
repeat
    Sequentially select T frames {x_1, ..., x_T}
    Extract features {o_1, ..., o_T}
    Generate hidden states {h_1, ..., h_T}
    Compute network output {μ_1, ..., μ_T}
    Randomly sample predictions for N episodes {l_1^{1:N}, ..., l_T^{1:N}} according to Equation 8
    Calculates rewards {r_1^{1:N}, ..., r_T^{1:N}} based on Equation 2 in early iterations or Equation 3 in late iterations
    b_t ← (1/N) Σ_{i=1}^{N} r_t^i, t = 1, ..., T
    Computes gradient according to Equation 7
    Update W using backpropagation
    Move on to next T frames
until reward doesn't increase
```

During testing, the network parameters W are fixed and no online fine-tuning is needed. The procedure at test time is as simple as computing one forward pass of our algorithm, *i.e.,* given a test video, the deep RL tracker predicts the location of target object in every single frame by sequentially processing the video data.

4. Experimental Results

We evaluated the proposed approach of visual object tracking on the Object Tracking Benchmark [30], and compared its performance with state-of-the-art trackers. Our algorithm was implemented in Python using TensorFlow toolbox[1], and ran at around 45 fps with an NVIDIA GTX 1080 GPU.

[1] https://www.tensorflow.org/

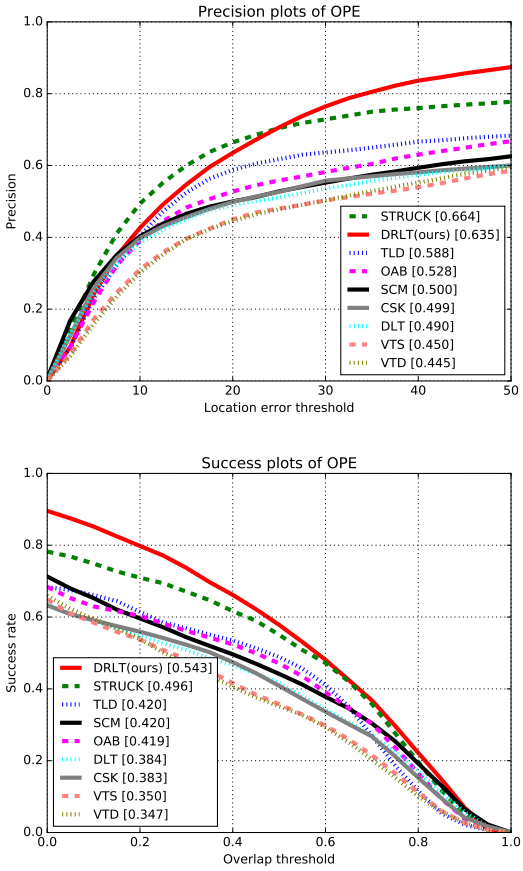

Figure 2: Precision and success plots on a subset of benchmark. The numbers in the legend indicate the representative precisions at 20 pixels for precision plot, and the AUC scores for success plots.

4.1. Evaluation Metrics

We followed the evaluation protocols in [29], where the performance of trackers was measured based on two different metrics: success rate and precision plots. In both metrics, the ratio of successfully tracked frames was measured by a set of thresholds, where bounding box overlap ratio and center location error were employed in success rate plot and precision plot, respectively. We ranked the tracking algorithms based on the Area-Under-Curve (AUC) for the success rate plot and center location error at 20 pixels for the precision plot, again, following [29]. We also compared the average bounding box overlap ratio for each tracking sequence, as well as run-time tracking speed.

4.2. Implementation Details

Here, we describe the design choices of our *observation network* and *recurrent network* as well as the *network learning* procedure in detail.

Observation network: We used a YOLO network [22] fine-tuned on the PASCAL VOC dataset [5] to extract visual features from observed video frames as YOLO was both accurate and time-efficient. The first Fc-layer features were extracted and concatenated with the location vector into a 5000-dimensional vector. Since the pre-trained YOLO weights were fixed during training, we added one more Fc-layer, with 5000 neurons on top of the concatenated vector, and provided the final observation vector as the input to the recurrent network.

Recurrent network: We used a 1-layer LSTM network with 5000 hidden units. At each timestep t, the last 4 digits were directly taken as the mean value μ of the location policy l. The location policy was sampled from a Gaussian distribution with mean μ and variance σ during training, and we found that $\sigma = 10^{-2}$ was good for both randomness and certainty in our experiment. During testing, we directly used the output mean value μ as prediction which was the same as setting $\sigma = 0$.

Network learning: The training algorithm was the same as Algorithm 1, we used $T = 10$ and $N = 5$ as these hyper-parameters provided the best tracking performance. We kept the pre-trained YOLO weights unchanged as they were proven to encode good information for both semantic prediction and localization, while the weights of the Fc-layer and the LSTM were updated using ADAM algorithm [15]. The initial learning rate was 10^{-5} and we exponentially annealed the learning rate from its initial value to 10^{-6} over the course of training. We trained the model up to 500 epochs, or until the cumulative tracking reward R stopped increasing. The first 300 epochs were trained with reward defined in Equation 2 while the last 200 epochs were trained with reward defined in Equation 3.

The trained model was directly applied to the test sequences with no online fine-tuning. During testing, only the video frames and the ground-truth location in the first frame were inputed to the network, and the predictions were directly generated through a single forward pass of the recurrent network.

4.3. Evaluation on benchmark

We have conducted extensive experiments on comparing the performance of our algorithm with eight other distinct trackers on a suite of 30 challenging and publicly available video sequences. Specifically, the one-pass evaluation (OPE) [29] was employed to compare our algorithm with seven top trackers included in the benchmark suite: STRUCK [8], TLD [14], CSK [10], OAB [7], VTD [17], VTS [18], SCM [33]. Note that DLT [27] was another tracking algorithm based on deep neural networks, which provided a baseline for tracking algorithms adopting deep learning. Since the YOLO weights were pre-trained on Im-

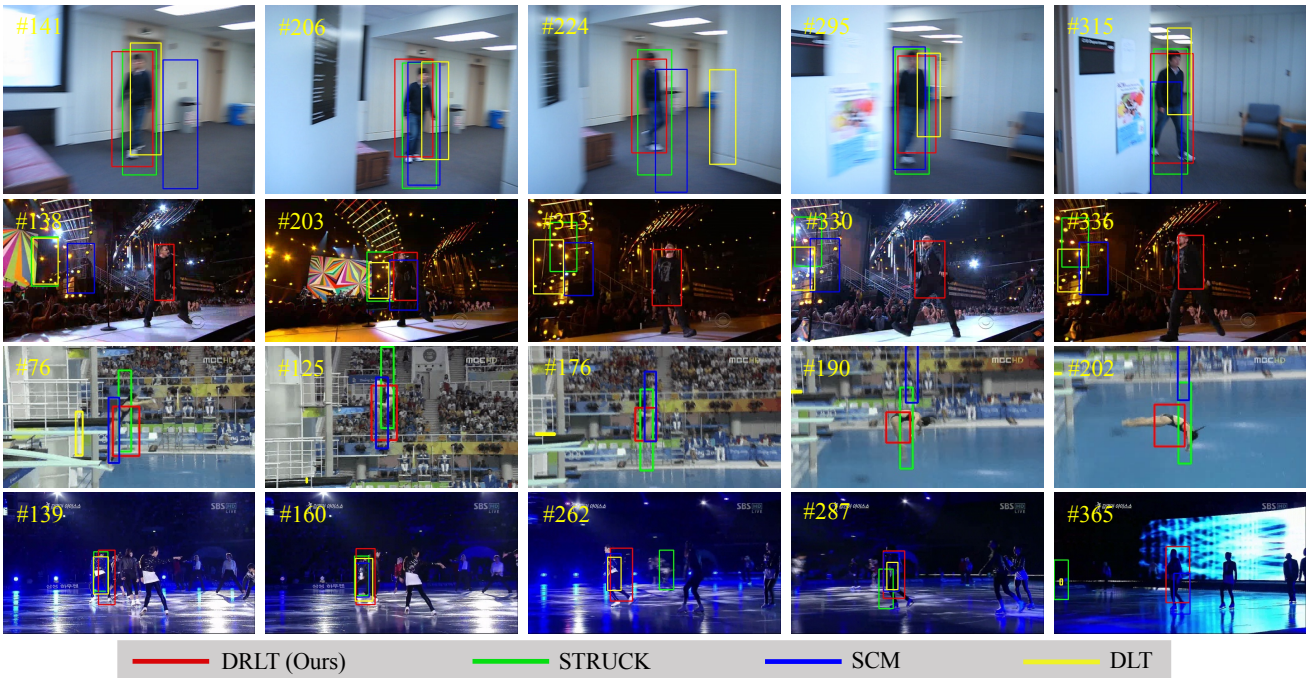

Figure 3: Qualitative results of the proposed method on some challenging sequences (*BlurBody*, *Singer2*, *Diving*, *Skating1*).

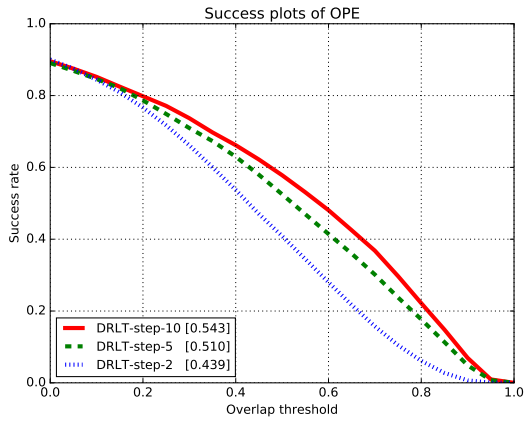

Figure 4: Success plots on a subset of benchmark for varying RNN step sizes. The numbers in the legend indicate the AUC scores.

ageNet dataset and finetuned on PASCAL VOC, capable of detecting objects of 20 classes, we picked a subset of 30 videos from the benchmark where the targets belonged to these classes (Table 2). According to our evaluation results, the difficulty of this subset was harder than that of the full benchmark.

As a generic object tracker, our algorithm was trained off-line and no online fine tuning mechanisms were applied.

Tracker	AUC	precision	speed (fps)
DLT [27]	0.384	0.490	8
STRUCK [8]	0.496	0.664	10
DRLT (ours)	0.543	0.635	45
DRLT-LSTM (ours)	0.543	0.635	270

Table 1: AUC, precision scores and run-time speed comparison between ours and two state-of-the-art trackers. DRLT-LSTM is our tracker with pre-computed YOLO features and only computing LSTM transitions during tracking. Our tracker is tested on a single NVIDIA GTX 1080 GPU.

Thus, training data with similar dynamics were needed to capture both categorical and motional information. We split the dataset and used first $1/3$ frames in each sequence with ground truth for off-line training, while the algorithm was tested on the whole sequence with unseen frames. This property made our algorithm especially useful in surveillance environments, where models could be trained off-line with pre-captured data.

Figure 2 illustrates the precision and success plots based on the center location error and bounding box overlap ratio, respectively. It clearly presented the superiority of our algorithm over other trackers. The higher success and precision scores indicated that our algorithm hardly missed targets while maintaining good tracking of tight bounding boxes to targets. The superior performance was probably because the CNN captured representative features for localization

Sequence	DRLT	STRUCK	SCM	DLT	VTS	VTD	OAB	CSK	TLD
Suv	0.621	0.519	0.725	0.743	0.468	0.431	0.619	0.517	0.660
Couple	0.493	0.484	0.100	0.237	0.057	0.068	0.346	0.074	0.761
Diving	0.540	0.235	0.272	0.141	0.213	0.210	0.214	0.235	0.180
Dudek	0.603	0.720	0.746	0.778	0.802	0.789	0.653	0.707	0.643
Human3	0.401	0.007	0.006	0.007	0.018	0.018	0.010	0.011	0.007
Human6	0.413	0.217	0.327	0.342	0.168	0.168	0.207	0.208	0.282
Human9	0.425	0.065	0.138	0.165	0.111	0.244	0.170	0.220	0.159
Jump	0.452	0.105	0.077	0.053	0.053	0.057	0.085	0.094	0.070
Jumping	0.651	0.664	0.116	0.598	0.149	0.116	0.069	0.050	0.662
Singer2	0.623	0.040	0.172	0.039	0.332	0.416	0.045	0.043	0.026
Skating1	0.457	0.285	0.444	0.422	0.482	0.492	0.368	0.478	0.184
Woman	0.479	0.693	0.651	0.595	0.132	0.145	0.466	0.194	0.129
Dancer	0.685	0.625	0.708	0.571	0.728	0.720	0.604	0.609	0.394
Liquor	0.532	0.408	0.311	0.342	0.427	0.478	0.457	0.253	0.456
BlurCar4	0.701	0.820	0.416	0.657	0.079	0.079	0.777	0.816	0.630
Human7	0.612	0.466	0.303	0.366	0.206	0.299	0.421	0.350	0.675
Human8	0.349	0.127	0.729	0.106	0.336	0.246	0.095	0.171	0.127
BlurCar2	0.693	0.743	0.185	0.732	0.108	0.108	0.100	0.743	0.726
Skater2	0.643	0.536	0.424	0.215	0.454	0.454	0.500	0.546	0.263
Bird2	0.473	0.565	0.756	0.221	0.328	0.214	0.648	0.580	0.570
Girl2	0.361	0.227	0.266	0.058	0.257	0.257	0.071	0.060	0.070
CarDark	0.548	0.872	0.844	0.582	0.717	0.521	0.765	0.744	0.423
CarScale	0.453	0.350	0.581	0.539	0.436	0.442	0.325	0.400	0.434
Car2	0.480	0.623	0.908	0.909	0.774	0.774	0.569	0.652	0.660
BlurCar3	0.680	0.780	0.220	0.205	0.188	0.188	0.720	0.430	0.639
Gym	0.641	0.350	0.214	0.178	0.359	0.367	0.069	0.219	0.276
BlurCar1	0.694	0.760	0.057	0.044	0.210	0.210	0.780	0.011	0.605
BlurBody	0.672	0.696	0.194	0.145	0.238	0.238	0.671	0.381	0.391
Skater	0.692	0.551	0.460	0.556	0.470	0.471	0.481	0.431	0.326
Dancer2	0.825	0.776	0.740	0.482	0.717	0.704	0.766	0.776	0.651
Average	0.562	0.477	0.403	0.368	0.334	0.331	0.402	0.367	0.403

Table 2: Average bounding box overlap ratio on individual sequence. Red: best, blue: second best.

and RNN was trained to force the long-term consistency of the tracking trajectory.

To gain more insights about the proposed algorithm, we evaluated the performance of trackers on individual sequences in the benchmark. Table 2 summarizes the average bounding box overlap ratio for each sequence. Our algorithm achieved best results for 12 sequences, and second best results for 4 sequences. We also achieved the best overall performance, beating the second best by almost 10% (0.562 vs. 0.477). Unlike other trackers where catastrophic failures were observed for certain sequences, our algorithm performed consistently well among all 30 sequences. This further illustrated that the spatial representations and temporal constraints learned by our algorithm were general, robust, and well-suited for tracking a large variety of targets.

We compared our tracker qualitatively with two distinct benchmark methods as well as DLT in Figure 3. It demonstrated that our tracker effectively handled different kinds of challenging situations that often required high-level semantic and temporal understanding such as motion blur, illumination variation, rotation, deformation, etc. Comparing with other trackers, our tracker hardly drifted to the background and predicted more accurate and reasonable bounding box locations.

To verify the importance of RNN in our algorithm, we did more experiments on varying RNN step sizes. Step size denoted the number of frames considered each time for training the network, referred to as T in Algorithm 1. Success plots of three different RNN step sizes were illustrated in Figure 4, and we found that larger step sizes allowed us to model longer and more complicated temporal constraints, thus resulting in better accuracy. This analysis demonstrated the importance of incorporating temporal information in tracking and the effectiveness of using our RL formulation.

Table 1 analyzed different trackers in terms of speed and accuracy. Our original model already operated at framerates beyond real-time by getting rid of online searching and fine-tuning mechanisms. Furthermore, pre-computing frame-level YOLO features off-line allowed us to only perform LSTM computation during online tracking, resulting in processing speed at 270 fps. In all, although our implementation was based on deep CNN and RNN, the proposed DRLT method was very efficient due to its extreme run-time simplicity, while preserving accurate tracking performance.

5. Conclusion

In this paper, we proposed a novel neural network tracking model based on a recurrent convolutional network trained with deep RL algorithm. To the best of our knowledge, we are the first to bring RL into CNN and RNN to solve the visual tracking problem. The entire network is end-to-end trainable off-line allowing it to run at frame-

rates faster than real-time. The deep RL algorithm directly optimizes a long-term tracking performance measure which depends on the whole tracking video sequence. Other than CNN-based trackers, our paper aims to develop a new paradigm for solving the visual tracking problem by bringing in RNN and RL to explicitly exploit temporal correlation in videos. We achieved state-of-the-art performance on OTB public tracking benchmark.

We believed that our initial work shed light on many potential research possibilities along this direction. Not only better training and design of recurrent convolutional network can further boost the efficiency and accuracy for visual tracking, but a broad new way of solving vision problem with artificial neural network and RL can be further explored.

References

[1] J. Ba, V. Mnih, and K. Kavukcuoglu. Multiple object recognition with visual attention. *arXiv preprint arXiv:1412.7755*, 2014. 3

[2] 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. 1, 3

[3] L. Bertinetto, J. Valmadre, J. F. Henriques, A. Vedaldi, and P. H. Torr. Fully-convolutional siamese networks for object tracking. In *European Conference on Computer Vision*, pages 850–865. Springer, 2016. 1, 3

[4] J. Donahue, L. Anne Hendricks, S. Guadarrama, M. Rohrbach, S. Venugopalan, K. Saenko, and T. Darrell. Long-term recurrent convolutional networks for visual recognition and description. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2625–2634, 2015. 1

[5] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Results. http://www.pascal-network.org/challenges/VOC/voc2012/workshop. 6

[6] Q. Gan, Q. Guo, Z. Zhang, and K. Cho. First step toward model-free, anonymous object tracking with recurrent neural networks. *arXiv preprint arXiv:1511.06425*, 2015. 3

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

[8] S. Hare, A. Saffari, and P. H. Torr. Struck: Structured output tracking with kernels. In *2011 International Conference on Computer Vision*, pages 263–270. IEEE, 2011. 1, 3, 6, 7

[9] D. Held, S. Thrun, and S. Savarese. Learning to track at 100 fps with deep regression networks. In *European Conference on Computer Vision*, pages 749–765. Springer, 2016. 1, 3

[10] J. F. Henriques, R. Caseiro, P. Martins, and J. Batista. Exploiting the circulant structure of tracking-by-detection with kernels. In *European conference on computer vision*, pages 702–715. Springer, 2012. 6

[11] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997. 4

[12] 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. 3

[13] S. E. Kahou, V. Michalski, and R. Memisevic. Ratm: Recurrent attentive tracking model. *arXiv preprint arXiv:1510.08660*, 2015. 3

[14] Z. Kalal, K. Mikolajczyk, and J. Matas. Tracking-learning-detection. *IEEE transactions on pattern analysis and machine intelligence*, 34(7):1409–1422, 2012. 2, 6

[15] D. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 6

[16] 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, 3

[17] J. Kwon and K. M. Lee. Visual tracking decomposition. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 1269–1276. IEEE, 2010. 6

[18] J. Kwon and K. M. Lee. Tracking by sampling trackers. In *Computer Vision (ICCV), 2011 IEEE International Conference on*, pages 1195–1202. IEEE, 2011. 6

[19] V. Mnih, N. Heess, A. Graves, et al. Recurrent models of visual attention. In *Advances in Neural Information Processing Systems*, pages 2204–2212, 2014. 1, 3

[20] H. Nam and B. Han. Learning multi-domain convolutional neural networks for visual tracking. *arXiv preprint arXiv:1510.07945*, 2015. 1, 3

[21] G. Ning, Z. Zhang, C. Huang, Z. He, X. Ren, and H. Wang. Spatially supervised recurrent convolutional neural networks for visual object tracking. *arXiv preprint arXiv:1607.05781*, 2016. 3

[22] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 779–788, 2016. 3, 6

[23] D. A. Ross, J. Lim, R.-S. Lin, and M.-H. Yang. Incremental learning for robust visual tracking. *International Journal of Computer Vision*, 77(1-3):125–141, 2008. 1, 3

[24] R. S. Sutton, D. A. McAllester, S. P. Singh, Y. Mansour, et al. Policy gradient methods for reinforcement learning with function approximation. 5

[25] 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. 3

[26] N. Wang, J. Shi, D.-Y. Yeung, and J. Jia. Understanding and diagnosing visual tracking systems. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 3101–3109, 2015. 2

[27] 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. 6, 7

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

[29] 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. 2, 6

[30] Y. Wu, J. Lim, and M.-H. Yang. Object tracking benchmark. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 37(9):1834–1848, 2015. 5

[31] K. Xu, J. Ba, R. Kiros, K. Cho, A. C. Courville, R. Salakhutdinov, R. S. Zemel, and Y. Bengio. Show, attend and tell: Neural image caption generation with visual attention. In *ICML*, volume 14, pages 77–81, 2015. 3

[32] K. Zhang, Q. Liu, Y. Wu, and M.-H. Yang. Robust visual tracking via convolutional networks. *arXiv preprint arXiv:1501.04505*, 2015. 3

[33] W. Zhong, H. Lu, and M.-H. Yang. Robust object tracking via sparsity-based collaborative model. In *Computer vision and pattern recognition (CVPR), 2012 IEEE Conference on*, pages 1838–1845. IEEE, 2012. 6