

# Hyperparameter Optimization for Tracking with Continuous Deep Q-Learning

Xingping Dong<sup>1</sup>, Jianbing Shen<sup>1,2</sup>, Wenguan Wang<sup>1</sup>, Yu, Liu<sup>1</sup>, Ling Shao<sup>2,3</sup>, and Fatih Porikli<sup>4</sup>

<sup>1</sup>Beijing Lab of Intelligent Information Technology, School of Computer Science, Beijing Institute of Technology, China

<sup>2</sup>Inception Institute of Artificial Intelligence, Abu Dhabi, UAE

<sup>3</sup>School of Computing Sciences, University of East Anglia, Norwich NR4 7TJ, U.K.

<sup>4</sup>Research School of Engineering, Australian National University, Australia

## Abstract

*Hyperparameters are numerical presets whose values are assigned prior to the commencement of the learning process. Selecting appropriate hyperparameters is critical for the accuracy of tracking algorithms, yet it is difficult to determine their optimal values, in particular, adaptive ones for each specific video sequence. Most hyperparameter optimization algorithms depend on searching a generic range and they are imposed blindly on all sequences. Here, we propose a novel hyperparameter optimization method that can find optimal hyperparameters for a given sequence using an action-prediction network leveraged on Continuous Deep Q-Learning. Since the common state-spaces for object tracking tasks are significantly more complex than the ones in traditional control problems, existing Continuous Deep Q-Learning algorithms cannot be directly applied. To overcome this challenge, we introduce an efficient heuristic to accelerate the convergence behavior. We evaluate our method on several tracking benchmarks and demonstrate its superior performance<sup>1</sup>.*

## 1. Introduction

Recent years witnessed a burst of object tracking solutions based on correlation filters and deep learning models. After the pioneering work of Henriques *et al.* [19], the kernelized correlation filter has evoked increasing attention in tracking applications thanks to its computational advantages, robust performance, and appealing theoretic

Corresponding author: Jianbing Shen (shenjianbing@bit.edu.cn). This work was supported in part by the Beijing Natural Science Foundation under Grant 4182056, the Australian Research Council's Discovery Projects funding scheme under Grant DP150104645, the Fok Ying Tung Education Foundation under Grant 141067, and the Specialized Fund for Joint Building Program of Beijing Municipal Education Commission.

<sup>1</sup>Our source code is available at [https://github.com/shenjianbing/dql\\_tracking](https://github.com/shenjianbing/dql_tracking).

KCF — SiamFc-3s — Siam-py — Ours —

Figure 1. Sample results of our hyperparameter optimization method on a real-time tracker: Siam-py [48]. By only optimizing the hyperparameters of Siam-py, we achieve a significant improvement. For instance, our method improves the distance precision [53] from 4.6% to an astounding score of 99.7% on *Bolt* (top row). It also outperforms the original versions of KCF [19] and SiamFc-3s [3].

cal framework. Many variants were devised to further improve its accuracy. For instance, Color Name [10] applies an alternative feature representation, DSST [9] estimates object scale using a multi-scale correlation filter, and LCT [34] and MUSTer [20] attempt to recover the target after a complete occlusion. In parallel, contemporary deep learning models have also been incorporated in object tracking tasks. Some approaches focus on accuracy, including DeepTrack [27, 28] and MDnet [37], which won the VOT 2015

challenge [26]. Several others aim at faster computational speeds such as deep regression networks [18], SiamFc [3], CFnet [48] and Siam-py [48].

Regardless of the tracking approach, hyperparameters need to be optimized for the desired performance. For example, Siam-py [48] adjusts its five hyperparameters by randomly grid searching 300 times in a validation dataset of 129 videos, which requires a lot of time. In most applications, hyperparameters are finetuned manually, which is impractical and time-consuming. Moreover, hyperparameters of many trackers are fixed and enforced to be identical across all test cases. Even if they may be suitable for some sequences, it is highly likely that the same hyperparameters may deteriorate the results for the others.

Our intuition is that by automatically selecting an optimal set of hyperparameters for each sequence, even for each frame, it is possible to improve the performance of a given baseline tracker. For instance, as shown in Fig. 1, the performance of Siam-py tracker [48] can be greatly enhanced by selecting a better set of hyperparameters. Most hyperparameter optimization algorithms including grid search, greedy search [1], tree search [22], Bayesian processes [44], and bandit-based scheme [29], concentrate on how to find the best hyperparameters on a validation set within a limited time and fix the obtained hyperparameters, which makes them invariant to test data.

Our ultimate goal is to optimize hyperparameters dynamically to accommodate different sequences. One idea is to employ a conventional neural network to predict the optimal hyperparameters for each sequence. However, we do not have the ground-truth values of the hyperparameters, thus we cannot provide target values for the network to regress them in a supervised fashion.

In reinforcement learning, an agent is trained to learn a policy to take a better action by giving a reward for its action according to the current state. The learning goal is to maximize the expected returns in a time sequence, where the return at time step  $t$  is defined as the accumulation of rewards from  $t$  to the end of the sequence. For our optimization task, we use a neural network to represent the agent and allow it choose the hyperparameters for each frame by regarding the choice as its action. By defining the reward as tracking accuracy, the goal of reinforcement learning becomes maximizing the expected cumulative tracking accuracies, which is consistent with the tracking evaluation.

Since the hyperparameters of most trackers are continuous, we apply reinforcement learning in continuous actions settings. Recently, [31] proposed Deep Deterministic Policy Gradient (DDPG) to solve the problem of continuous actions by adding an action-predict network into Deep Q-learning [36]. Following that, [15] simplified DDPG with Normalized Advantage Functions (NAF), which devises an imagination rollouts mechanism to accelerate the learning

process. NAF decomposes the Q-function (evaluation of the action taken at current state) into three independent deep networks: action-predict, value, and lower-triangular matrix nets, which are trained by using replay buffer for sampling and minimizing the Q-value’s temporal difference between samples. To reduce the training time, we opt for NAF as our base method. However, the convergence speed of a direct incorporation of NAF would still be too slow for object tracking due to two reasons. The first one is that our state-space (embodying observations) is much higher dimensional than the state-spaces of ordinary control problems. For example, in the task of ‘pendulum swing-up’, only a vector with three coefficients is sufficient to represent the state-space [43]. In object tracking, the dimensionality of the state vector may easily reach to hundreds or thousands to allow the extracted features to accurately represent video frames. The second factor is that computing the reward, i.e., tracking accuracy, in an object tracking setting would involve more intensive computations (tracker needs to be run) whereas in other applications of reinforcement learning it would take little computational time to get the reward.

To obtain an even further acceleration of NAF, we impose an efficient heuristic inspired by some recent studies [11, 4, 42, 46, 41, 32, 33]. Firstly, we use the original hyperparameters as the supervised labels to train the action-predict network to attain a reliable initial action (hyperparameters). Then, we freeze the action-predict network and learn parameters of the other value, and lower-triangle matrix networks. After that, we have a reliable initialization and apply it to train all three networks with the NAF method. Our heuristic is simple yet effective, which is demonstrated in our experiments. After the training, we only apply the action-predict network to adjust the hyperparameters of the tracker. Figure 2 shows a flowchart of our adjustment process. In our experiments, we use a small action-predict network (only with 3 convolutional layers and 3 fully-connected layers) to reduce the computations further. We verify our method on a recent real-time tracker, named Siam-py [48]. The results show that the adjusted tracker achieves a significant improvement in terms of the Area Under Curve (AUC) metric [53] (by increasing 5.4% from 0.597 to 0.629 on OTB-2013 [53]) while having slightly reduced speed (from 74 FPS to 69 FPS). Our hyperparameter optimization is suitable for all trackers that generate heat-maps, including the majority of the correlation filter based trackers and deep trackers [51, 3].

## 2. Related works

**Hyperparameter optimization:** Consequently, hyperparameters are set with brute-force methods such as random search and grid search. To develop more efficient search methods, some researchers [1, 22, 44] dominate the problem of hyperparameter optimization by using Bayesian Op-

Figure 2. The flow chart of online tracking with our action-predict network based adjustment process. Green arrows shows the original tracking method. A tracker is applied on search region to get a heat-map used to predict the object bounding box, and offer search region for next frame. To adjust the original tracker, we only add two parts: initialization (blue arrows) and action prediction (orange arrows). In initialization, we run the original tracker once on search region in frame 2 to get the initial heat-map. In action prediction, the heat-map is feed into Action Network to predict optimal hyperparameters to adjust the tracker.

timization methods to identify good hyperparameter configurations more quickly than standard baselines like random search. Existing works [47, 13, 45] have proven that these methods outperform random search in empirical experiments. To further accelerate Configuration Selection, Li *et al.* [29] define hyperparameter optimization as a pure-exploration non-stochastic infinitely many armed bandit problem and give some theoretical desirable guarantees. The aim of the above hyperparameter optimization methods is to find a fixed good set of hyperparameters, which is not suitable for our task since we need to search for dynamical hyperparameters for each sequence. A similar work [17] is proposed to control the learning rate on the gradient-based learning method by using deep Q-learning. In fact, it is only used to learn an optimization hyperparameter while our method can be applied to control multiple hyperparameters.

**Continuous Deep Q-learning:** Mnih *et al.* [36] propose a ‘Deep Q Network’ (DQN) algorithm to play many Atari video games and achieve human level performance, by combining advances in deep learning and reinforcement learning. DQN is effective for the task with discrete and low-dimensional action spaces, but it cannot be directly used for high-dimensional, continuous action spaces. To solve this problem, Lillicrap *et al.* [31] present a model-free, off-policy actor-critic algorithm based on deep approximate function and the Deterministic Policy Gradient (DPG) algorithm [43], called DDPG. They design an actor network to represent a continuous action and a critic Q network to evaluate its action. These networks are trained with a replay buffer similar in DQN. To accelerate DDPG, Gu *et al.* [15] propose two complementary techniques: they derive a continuous variant of the Q-learning algorithm called Normalized Advantage Functions (NAF) to simplify the DDPG; they use the iteratively refitted local linear models (imagination rollouts mechanism) to further

speed up the process. Existing empirical evidences have shown DDPG and NAF can learn competitive policies for the task using low-dimensional observations. However, for high-dimensional tasks like our hyperparameter optimization on tracking, they may fail or take long time to learn good policies.

**Tracker with heat-map:** Our hyperparameter framework can be directly applied to the tracker with heat-map, thus we give several real-time related works including some deep learning trackers [3, 48, 16] and recent correlation filter based trackers [5, 19, 9, 10, 34, 58, 2, 12, 52, 24, 14]. Here, we only introduce some trackers with high frame-rates (more than 50 FPS).

Bertinetto *et al.* [3] propose a end-to-end trained Siamese network with fully convolutional layers (SiamFc) for tracking, which is not updated during the tracking phase. So it achieves high frame-rates beyond real-time, nearly 86 FPS with GPU. In CFnet [48], a correlation filter layer is embedded into a Siamese network to enable learning deep features that are tightly coupled to the Correlation Filter (CF). It shows that 2 convolutional layers adding CF layer in Siamese network will achieve similar performance and speed (75 FPS) compared with SiamFc including 5 convolutional layers.

The early MOSSE [5] and improved Kernelized Correlation Filter (KCF) [19] can achieve the speed at 669 FPS and 292 FPS respectively. The following CN [10] tracker combined with adaptive color name feature can operate at 105 FPS. Another work Staple [2] tries to use multiple features containing color and HOG [8] to achieve robust performance and high speed (80 FPS). Recently, Wang *et al.* [52] accelerate the Structured output support vector machine (SVM) based tracker using the correlation filter algorithm and it can also run at 80 FPS.

**Tracker with reinforcement learning:** In computer

vision, reinforcement learning (RL) has been successfully used to some applications, such as object detection [6, 23]. For visual object tracking, there are several works [55, 7, 56, 21, 46] that use RL to learn good policies for tracking, e.g., [46] learns when to update the tracker and [56] learns an early decision policy for different frames to speed up the tracker. Our work is the first one to use RL to learn the optimal hyperparameters of a tracker for each frame.

### 3. Our approach

In this section, we first briefly introduce reinforcement learning and how to transfer it for continuous hyperparameter optimization. Then we describe a heuristic method to speedup continuous deep Q-learning for our task. For unified notation, we use the same math notation in [15] for reinforcement learning.

#### 3.1. Reinforcement learning for hyperparameter optimization

In reinforcement learning, the main abstract concepts include the agent and environment  $E$ , which are interacted for each other. At each time step  $t \in [1, T]$ , the agent take an action  $u_t$  from action space  $U$  according to its current policy ( $u_t|x_t$ ) and state  $x_t \in X$  in environment  $E$ . The environment gives a reward  $r(x_t, u_t)$  for this action and then update its states with a dynamic distribution  $p(x_{t+1}|x_t, u_t)$ . According to this new state, the agent will go into next time step and choose a new action. For each time step  $t$ , a return reward is defined as  $R_t = \sum_{i=t}^T \gamma^i r(x_i, u_i)$ , where  $\gamma \in [0, 1]$  is a discount factor that prioritizes earlier reward-s over later ones. The goal of reinforcement learning is to train an agent with policy  $\pi$  to maximize the expected sum of returns, defined as  $R = E_{r_{1:T}, x_{1:T} \sim E, u_{1:T} \sim \pi} [R_1]$ . To optimize the expected return  $R$ , a variant of model-free and model-based algorithms are proposed. In the next subsection, we will review the most recent deep Q-learning algorithm for continuous action space [15], which is the main learning method in our experiments. Now we explain how to transfer the reinforcement learning to hyperparameter optimization in object tracking.

In hyperparameter optimization for object tracking, the goal is find the optimal hyperparameters of the tracker for each frame in a sequence. An object tracking algorithm and its input sequence can be seen as the environment in reinforcement learning. Adjusting the hyperparameters can be seen as an action. Then the hyperparameter optimization is turned into learning a policy to choose the best action by training an adjusting agent. In object tracking, a lot of trackers like [3, 48] will produce a heat-map in the search region and then the location with the highest value is selected as the object location. This heat-map will capture the information of the tracker and the appearance of the object being tracked, and it will offer useful information for adjusting

hyperparameters. Thus, we can apply the heat-map  $h_{t-1}$  in the previous frame to construct the state  $x_t$ , written as

$$x_t = \{h_{t-1}, o_{t-1}, \text{tr}_{t-1}\} \quad (1)$$

where  $h_{t-1} = \text{HEAT}(o_{t-1} | \text{tr}_{t-1})$  is a spatial heat-map with size  $H \times W$ ,  $o_{t-1}$  is the appearance feature (such as the RGB-color or deep convolutional feature) of the search region with size  $h \times w \times f$ ,  $\text{HEAT}$  is a mapping function from  $R^{h \times w \times f}$  to  $R^{H \times W}$ , and  $\text{tr}_{t-1}$  is the parameters of the tracker (including hyperparameters). Given the state  $x_t$ , the adjusting agent will take an action i.e. choose a hyperparameter vector  $a_t \in R^{N_a}$ , where  $N_a$  is the number of hyperparameters to be adjusted. It is a user-defined variable and varies with different trackers. In this paper, we choose 5 key hyperparameters in Siam-py [48] containing scale step, scale penalty, scale learning rate, window weight, and template learning rate. After taking an action ( $u_t = a_t$ ), the tracker will be updated by replacing the hyperparameters with  $u_t$ , formulated as

$$\text{tr}_t = \text{UPDATE}(\text{tr}_{t-1}, u_t) \quad (2)$$

Then we use the updated tracker to predict the object box  $b_t$  in current frame  $t$ , defined as

$$b_t = \text{map}(o_t | \text{tr}_t) \quad (3)$$

where  $\text{map}$  is the mapping function of the tracker. As shown in equation (1), the updated tracker  $\text{tr}_t$  is also applied to predict the heat-map in frame  $t$ , and transfers the state from  $x_t$  to  $x_{t+1}$ . In the training phase, the reward function  $r(x_t, u_t)$  is defined according to the Intersection-over-Union (IoU) between the predicted box  $b_t$  and ground-truth box  $g_t$ . The IoU can be formally defined as

$$\text{IoU}(b_t, g_t) = \text{area}(b_t \cap g_t) / \text{area}(b_t \cup g_t). \quad (4)$$

Then the reward function  $r(x_t, u_t)$  is formulated as

$$r(x_t, u_t) = \begin{cases} \text{IoU}(b_t, g_t), & \text{stop} = 0 \\ -3, & \text{stop} = 1 \end{cases} \quad (5)$$

where  $\text{stop}$  is a flag deciding whether the agent stops to take an action. Assume the agent takes a policy to adjust the tracker but gets small IoUs (less than a threshold  $\text{thres}$ ) at consecutive  $K$  frames - it means tracking failure and this policy is not suitable for the current sequence. We need to give a negative reward to punish the agent and stop to adjust the tracker with this policy in the current sequence. Thus, we use the  $\text{stop}$  flag to represent the situation with tracking failure and define the reward function in equation (5). In our experiments, we set  $\text{thres} = 0.5$  and  $K = 5$ .



### 3.2. Continuous Deep Q-learning

To solve the reinforcement learning problem in a continuous action space, Lillicrap *et al.* [31] propose a simple model to enable deep Q-learning into the continuous action space and accelerate learning with new imagination rollouts. This method is suitable for the classical control tasks with a low dimensional state space. However, it cannot be applied directly to our hyperparameter optimization because of the huge dimension of our state space. Inspired by recent works with reinforcement learning for tracking [55, 46], we design a novel heuristic method to pre-train the networks. Firstly, we briefly introduce the continuous deep Q-learning [15], and then explain our heuristic method.

In Q-learning, the Q function  $Q(x_t, u_t)$  evaluating action  $u_t$ -value is defined as the expected return from  $x_t$  after taking action  $u_t$  following policy  $\mu$ . The formulation is

$$Q(x_t, u_t) = E_{r_{t+1}, x_{t+1} \sim E_{u_t}} [R_t | x_t, u_t]. \quad (6)$$

This function is applied to choose the best action  $\mu(x_t) = \arg \max_u Q(x_t, u_t)$ , which corresponds to  $(u_t | x_t) = (u_t = \mu(x_t))$ . Denote  $Q$  as the parameters of action-value function,  $\pi$  to be an arbitrary exploration policy, and  $\rho$  as the corresponding state distribution, then the learning goal is to minimize the Bellman error:

$$L(Q) = E_{x_t, u_t \sim \pi, r_t \sim E} [(Q(x_t, u_t) - y_t)^2] \quad (7)$$

$$y_t = r(x_t, u_t) + \gamma Q(x_{t+1}, \mu(x_{t+1})). \quad (8)$$

For a discrete action problem, we only use a deep network to represent the Q-function and easily take a best action  $\mu(x_t)$  by traversing the action space. However, it is difficult for a continuous action problem. To solve this problem, Gu *et al.* [15] propose a continuous Q-learning method with Normalized Advantage Functions (NAF). They decompose the Q-function into two terms: a value function  $V(x)$  and an advantage function  $A(x, u)$  containing the action-predict network to determine its maximum i.e.  $\mu(x) = \arg \max_u Q(x, u)$ . The advantage term is a quadratic function of nonlinear features of the state, written as:

$$A(x, u) = -\frac{1}{2} (u - \mu(x))^T P(x, u) (u - \mu(x)) \quad (9)$$

where  $P(x, u)$  is a state-dependent, positive-definite square matrix, which is decomposed as  $P(x, u) = L(x, u) L(x, u)^T$ , where  $L(x, u)$  is a lower-triangular matrix whose elements come from an output of a neural network, with the diagonal terms exponentiated. The final Q-function is defined as:

$$Q(x, u) = A(x, u) + V(x). \quad (10)$$

According to equations (9) and (10), three networks corresponding to  $\mu$ ,  $P$ , and  $V$  should be defined to construct

the Q-function. Then they train the combined network by using target networks, replay buffers and imagination rollouts mechanism.

As mentioned before, the NAF method cannot be applied directly to our hyperparameter optimization, since it is more complex than the classical controlling tasks. It is in common use to give a reliable initiation for a deep network before using it to reinforcement learning [4, 42, 55, 46]. Thus, we design a heuristic method to initialize the parameters of Q-function:  $\mu$ ,  $P$ , and  $V$ . Firstly, we train the action-predict network  $\mu$  with an off-line heuristic method by using supervised learning. The original hyperparameter vector  $\mu_0$  of a tracker is regarded as the target of the action-predict network and the loss function is defined with a mean squared error between them, written as:

$$L(\mu) = \frac{1}{N} \sum_i \|\mu(x_i) - \mu_0\|_2^2. \quad (11)$$

Usually  $\mu_0$  is manually adjusted to fit the tracker, so it is more reliable than the random initialization. Letting the prediction of network  $\mu$  be close to  $\mu_0$  will keep the initial adjusted tracker to be consistent with the original tracker in terms of tracking performance. The other two networks  $P$  and  $V$  cannot be initialized with supervised learning, since we cannot estimate the approximation of their output. Here, we use an on-line heuristic method by fixing the network  $\mu$  and learning the other networks  $P$  and  $V$  using the NAF method. Given the initialization of these three networks, we can learn the Q-function with the NAF method.

### 4. Implementation Details

**Q-function with three networks:** We define the Q-function containing  $\mu(x)$ ,  $L(x, u)$ , and  $V(x)$  with three simple deep neural networks, and each one includes three convolutional layers and several fully-connected layers. To process the input state  $x$  with size  $H \times W$ , we design three convolutional layers. The first one consists of  $5 \times 5 \times 128$  convolutions followed by ReLU and  $2 \times 2$  max pooling. The second one is constructed with  $3 \times 3 \times 64$  convolutions followed by ReLU and  $2 \times 2$  max pooling. The last one is similar with the second but replaces the max pooling with a flatten layer to output a vector. These three convolutional layers are applied to these three networks since all of them need the input state  $x$ . For  $\mu(x)$ , we add three fully-connected layers following the last convolutional layer. The first and second layer outputs 128 channels followed by ReLU. The last layer is a linear output layer with size  $N_a$ , where  $N_a$  is the dimension of the action space, i.e. the number of hyperparameters. In  $V(x)$ , we also add three similar fully-connected layers except for the different output numbers, which are defined as 64, 64, 1, respectively. To handle the other input ac-

tion  $u$  of  $L(x, u | P)$ , we enlarge the action with two fully-connected layers followed by ReLU, where the corresponding output number is 64 and 1024. The underlying goal is to achieve a similar size between the enlarged action  $u$  and the convolutional output of state  $x$ . Then we concatenate them and feed the output to a network with three fully-connected layers, which is similar with that in network  $\mu(x | P)$  except that the output numbers are 256, 256, and  $N_a(N_a + 1)/2$ .

**Base tracker:** Recently, Valmadre *et al.* [48] provide a Python version of their code implementing its baseline algorithm (Siam-py), which is a variant of SiamFc tracker [3]. They modify the 5 convolution layers of SiamFc to get a bigger heat-map with spatial size from  $17 \times 17$  to  $33 \times 33$ . We use this algorithm as our base tracker, since it is easy to be combined with the Python toolkit for reinforcement learning: Keras-rl [39]. In fact, Siam-py will produce a multi-scale heat-map for scale estimation. We only use the heat-map with the original scale as the input of our state. The hyperparameters, i.e. actions, are selected with 5 key hyperparameters in Siam-py containing scale step, scale penalty, scale learning rate, window weight, and template learning rate. These 5 hyperparameters are limited between the lower bound (1.02, 0.90, 0.40, 0.10, 0.00) and the upper bound (1.08, 1.00, 1.00, 0.50, 0.05). According this limit range, we normalize each hyperparameter to [0, 1] for training. This normalization will reduce the issue with different magnitudes of the 5 hyperparameters.

**Training details:** We use the NAF method in Keras-rl [39] to train the main three networks with ADAM [25] on a video detection benchmark of ILSVRC15 [40]. The learning rate is initialized with 0.001. Our training process consists of three phases: supervised learning, NAF learning with fixed action, and NAF learning. Firstly, in supervised learning, we only want to learn a coarse action-predict network. Thus to avoid over-fitting, we train it with mean squared loss in 10 epochs including 500 batches. In the last two phases, we train them in 200000 and 24000 steps, respectively. More steps are used in phase 2 since most parameters of networks are randomly initialized. The batch size in all phases is set as 128. During training, we randomly sample a video clip with 20 frames as an episode defined in reinforcement learning and the discount factor  $\gamma$  is set as 0.99. After training, the action-predict network is applied to adjust the corresponding tracker. In our experiments, the tracker Siam-py [48] with our hyperparameter optimization (HP) can run at average 69 FPS (original speed is 74 FPS) on OTB-2013 [53].

## 5. Experimental Results

In our experiments, we use the popular tracking benchmarks OTB-2013 [53], OTB-50, OTB-100 [54], and VOT-2015 [26] as test sets, which are used to compare our method with state-of-the-art trackers.

NAF — NAF+SL — NAF+SL+Mu (Ours) —

Figure 3. Self-comparison with NAF, NAF+SL, and NAF+SL+Mu (Ours) in terms of *loss*, *mean q*, *episode reward* and *mean IoU*. The *loss* may increase with the training since the  $Q$  estimates are away from true reward returns for the complex tasks. Still, they can be used to learn competent policies to obtain sufficiently good results. *mean q* is the average of estimated  $Q$  values of each episode.

**Evaluation metric:** The success and precision metrics [53] are used to evaluate all trackers on OTB-2013, OTB-50, and OTB-100. Success measures the intersection over union (IoU) of ground truth and predicted bounding boxes. The success plot shows the rate of bounding boxes whose IoU score is larger than a given threshold. Area Under the Curve (AUC) of success plots is applied to rank the trackers. The precision metric measures the percentage of frame locations within a certain threshold distance from those of the ground truth. The threshold distance is set as 20 for all trackers. For the VOT-2015 dataset, we evaluate tracking performance in terms of accuracy (overlap with the ground-truth) and robustness (failure rate) [26]. In VOT-2015, a tracker is restarted in the case of a failure, where there is no overlap between the predicted bounding box and ground truth.

### 5.1. Ablation Study

We evaluate our heuristic training method (NAF+SL+Mu) comparing the original NAF learning [15] and its variant initialized with the supervised action-predict network (NAF+SL). Fig. 3 shows the *loss*, *mean q* during our second training phase with 200000 steps and the *episode reward*, *mean IoU* on OTB-2013 in the last training phase with 24000 steps. The first two metrics are applied to analyze convergence and the last two is used to evaluate tracking accuracy. Our *loss* is lower than other two methods, which indicates our method can get faster convergence. The plot of *mean q* shows its value is proportional to the *loss* that means a too large



Figure 4. The results of OTB-2013 [53] benchmark. Success plots with AUC for OPE, TRE and SRE are one pass evaluation, temporal robustness, and spatial robustness evaluation, respectively. Only the top ten trackers are shown.

predicted  $Q$  value may cause slow convergence speed or even divergence. Our method achieves lower *mean  $q$*  since we fix the action-predict network ( $\mu$  net) to simplify the  $Q$  function. NAF gets similar *mean  $q$*  since its  $\mu$  net falls into local optimal values and outputs fixed values, which is also proved by its flat *mean IoU*. We have higher *episode reward* and *mean IoU*. This also demonstrates the effectiveness of our method.

## 5.2. Comparison on OTB-2013

On OTB-2013 including 50 videos, we compare our HP tracker against state-of-the-art trackers that can operate in real-time: Siam-py [48], SiamFc-3s [3], Staple [2], LCT [34], and KCF [19]. For reference, we also compare with recent trackers: DSST [9], MEEM [57], SAMF [30], DLSSVM [38] and 29 trackers from OTB-2013. AUC scores of all trackers are reported for OPE (one pass), SRE (spatial robustness) and TRE (temporal robustness) evaluations [53]. For OPE, the trackers run once with initialization from the ground truth position in the first frame. TRE starts at different frames and SRE uses different bounding boxes in the first frame for initialization. As shown in Fig. 4, our method outperforms recent state-of-the-art trackers in terms of OPE, TRE and SRE. Notice that Siam-py is our base tracker. Our tracker improves its AUC from 0.597 to 0.629 in terms of OPE and in robustness evaluations TRE and SRE, we also perform better than it.

## 5.3. Results on OTB-50 and OTB-100

OTB-100 benchmark improves OTB-2013 by adding more video sequences and selects 50 more challenging sequences to construct a small benchmark OTB-50. On these two benchmarks, we only compare the recent trackers mentioned on OTB-2013 comparison, since the performance of other 29 trackers is lower. Both precision and success metrics are reported for OPE. Fig. 5 shows that our tracker also

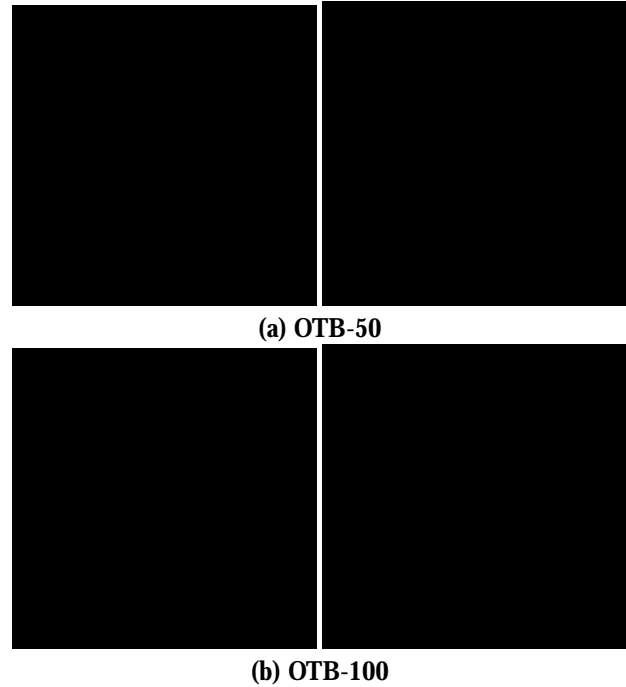


Figure 5. Precision and success plots with AUC for OPE on OTB-50 and OTB-100 [54] benchmark.

performs better than the other trackers in these two benchmarks in terms of precision and success metrics. On OTB-50, we have significant improvement in these two metrics. For example, our method achieves a promotion of 7.4% in the AUC metric compared with the second best tracker SiamFc-3s. On OTB-100, compared with the second best tracker, we also improve 1.5% and 3.3% in terms of precision and AUC, respectively.

**Attribute-based Performance Analysis.** On OTB-100, the sequences are annotated with 11 attributes for different challenging factors including Illumination Variation (IV),

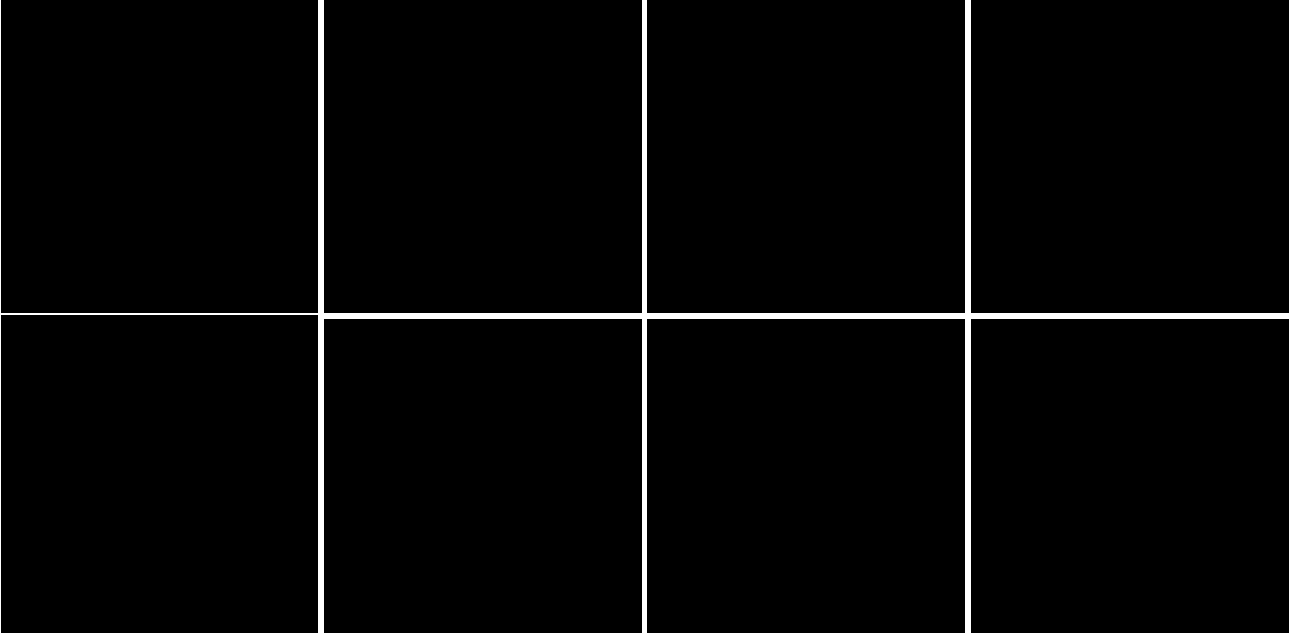


Figure 6. Overlap success plots of OPE with AUC for eight tracking challenges in terms of SV, OCC, DEF, MB, FM, IPR, OPR and OV.

Table 1. Evaluation on VOT2015 by the means of accuracy, robustness and speed. We run our method, Siam-py and SiamFc-3s, and report their speed (FPS). Otherwise (\*) we report the values from the VOT2015 results [26] in EFO units, which roughly correspond to FPS (e.g. the speed of the NCC tracker is 140 FPS with 160 EFO). The **first** and **second** best scores are highlighted in color.

	HP(ours)	Siam-py	SiamFc-3s	BDF	NCC	FOT	ASMS	FCT	matFlow	SKCF	PKLTF
Acc.	<b>0.578</b>	0.540	<b>0.549</b>	0.401	0.500	0.432	0.507	0.431	0.420	0.485	0.453
Rob.	<b>1.578</b>	<b>1.366</b>	1.818	3.106	11.345	4.360	1.846	3.338	3.121	2.681	2.721
FPS	69	74	78	<b>175*</b>	<b>140*</b>	126*	101*	73*	71*	58*	26*

Scale Variation (SV), Occlusion (OCC), Deformation (DEF), Motion Blur (MB), Fast Motion (FM), In-Plane Rotation (IPR), Out-of-Plane Rotation (OPR), Out-of-View (OV), Background Clutters (BC), and Low Resolution (LR). To evaluate the proposed method, we compare our method with other trackers in the subsets with different dominant attributes. Fig. 6 shows the results of 8 main challenging attributes evaluated by the overlap success plots of OPE. Our approach outperforms all others in 6 subsets: SV, OCC, MB, FM, IPR, and OPR, and achieves the second best performance in DEF and OV following Staple and SiamFc-3s, respectively. Compared with our baseline Siam-py, our tracker gets significant improvement in all 8 attributes.

#### 5.4. Evaluation on VOT-2015

**Fast speed:** We compare our tracker with Siam-py, SiamFc-3s and 8 top participants in the VOT-2015 in terms of speed, including BDF [35], FOT [49], ASMS [50], NCC, FCT, matFlow, SKCF, and PKLTF [26]. Table 1 shows that our tracker achieves the highest accuracy among the most accurate trackers with speed more than 20 FPS. Among all VOT2015 trackers, including the ones with less than 1 FPS speed, our tracker achieves the second highest accuracy

closely following MDnet [37] (accuracy: 0.603). Among the fast trackers, the highest robustness (1.366) belongs to Siam-py followed by ours HP (1.578). Our tracker significantly improves the accuracy and robustness of all participants with top speed in VOT2015 (BDF, FOT, ASMS, NCC, FCT, matFlow, SKCF, PKLTF) and SiamFc-3s.

## 6. Conclusion

In this paper, we proposed a hyperparameter optimization algorithm for object tracking by using continuous deep Q-learning. Unlike the general hyperparameter optimization that assigns fixed hyperparameters for all sequences, our method can adjust hyperparameters for different sequences to achieve the best accuracy. Also, we proposed a simple and effective heuristic to enable the continuous deep Q-learning to handle high-dimensional state spaces. We demonstrated the superior accuracy and competitive speed of our method compared to recent real-time CF-based and deep trackers over an extensive evaluation. Such a hyperparameter optimization method is appealing in that it is trained end-to-end and can be used to any tracker with key hyperparameters.



## References

- [1] J. S. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl. Algorithms for hyper-parameter optimization. In *NIPS*, pages 2546–2554, 2011. **2**
- [2] L. Bertinetto, J. Valmadre, S. Golodetz, O. Miksik, and P. H. Torr. Staple: Complementary learners for real-time tracking. In *IEEE CVPR*, pages 1401–1409, 2016. **3, 7**
- [3] L. Bertinetto, J. Valmadre, J. F. Henriques, A. Vedaldi, and P. H. Torr. Fully-convolutional siamese networks for object tracking. In *ECCV Workshops*, pages 850–865, 2016. **1, 2, 3, 4, 6, 7**
- [4] R. A. Bianchi, C. H. Ribeiro, and A. H. Costa. Heuristically accelerated reinforcement learning: Theoretical and experimental results. In *Proceedings of the 20th European Conference on Artificial Intelligence*, pages 169–174. IOS Press, 2012. **2, 5**
- [5] D. S. Bolme, J. R. Beveridge, B. A. Draper, and Y. M. Lui. Visual object tracking using adaptive correlation filters. In *IEEE CVPR*, pages 2544–2550, 2010. **3**
- [6] J. C. Caicedo and S. Lazebnik. Active object localization with deep reinforcement learning. In *IEEE ICCV*, pages 2488–2496, 2015. **4**
- [7] J. Choi, J. Kwon, and K. M. Lee. Visual tracking by reinforced decision making. *arXiv preprint arXiv:1702.06291*, 2017. **4**
- [8] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *IEEE CVPR*, volume 1, pages 886–893, 2005. **3**
- [9] M. Danelljan, G. Häger, F. Khan, and M. Felsberg. Accurate scale estimation for robust visual tracking. In *BMVC*, 2014. **1, 3, 7**
- [10] M. Danelljan, F. Shahbaz Khan, M. Felsberg, and J. Van de Weijer. Adaptive color attributes for real-time visual tracking. In *IEEE CVPR*, pages 1090–1097, 2014. **1, 3**
- [11] X. Dong, J. Shen, L. Shao, and M.-H. Yang. Interactive cosegmentation using global and local energy optimization. *IEEE Trans. on Image Processing*, 24(11):3966–3977, 2015. **2**
- [12] X. Dong, J. Shen, D. Yu, W. Wang, J. Liu, and H. Huang. Occlusion-aware real-time object tracking. *IEEE Trans. on Multimedia*, 19(4):763–771, 2017. **3**
- [13] K. Eggenberger, M. Feurer, F. Hutter, J. Bergstra, J. Snoek, H. Hoos, and K. Leyton-Brown. Towards an empirical foundation for assessing bayesian optimization of hyperparameters. In *NIPS workshop on Bayesian Optimization in Theory and Practice*, volume 10, 2013. **3**
- [14] H. Fan and H. Ling. Parallel tracking and verifying: A framework for real-time and high accuracy visual tracking. *arXiv preprint arXiv:1708.00153*, 2017. **3**
- [15] S. Gu, T. Lillicrap, I. Sutskever, and S. Levine. Continuous deep q-learning with model-based acceleration. In *ICML*, pages 2829–2838, 2016. **2, 3, 4, 5, 6**
- [16] Q. Guo, W. Feng, C. Zhou, R. Huang, L. Wan, and S. Wang. Learning dynamic siamese network for visual object tracking. In *IEEE ICCV*, pages 1–9, 2017. **3**
- [17] S. Hansen. Using deep q-learning to control optimization hyperparameters. *arXiv preprint arXiv:1602.04062*, 2016. **3**
- [18] D. Held, S. Thrun, and S. Savarese. Learning to track at 100 fps with deep regression networks. In *ECCV*, pages 749–765, 2016. **2**
- [19] J. F. Henriques, C. Rui, P. Martins, and J. Batista. High-speed tracking with kernelized correlation filters. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 37(3):583–596, 2015. **1, 3, 7**
- [20] 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 *IEEE CVPR*, pages 749–758, 2015. **1**
- [21] C. Huang, S. Lucey, and D. Ramanan. Learning policies for adaptive tracking with deep feature cascades. *arXiv preprint arXiv:1708.02973*, 2017. **4**
- [22] F. Hutter, H. H. Hoos, and K. Leyton-Brown. Sequential model-based optimization for general algorithm configuration. *LION*, 5:507–523, 2011. **2**
- [23] Z. Jie, X. Liang, J. Feng, X. Jin, W. Lu, and S. Yan. Tree-structured reinforcement learning for sequential object localization. In *NIPS*, pages 127–135, 2016. **4**
- [24] H. Kiani Galoogahi, A. Fagg, and S. Lucey. Learning background-aware correlation filters for visual tracking. In *IEEE CVPR*, pages 1135–1143, 2017. **3**
- [25] D. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. **6**
- [26] M. Kristan, J. Matas, A. Leonardis, M. Felsberg, L. Čehovin, and et al. The visual object tracking vot2015 challenge results. In *IEEE ICCV Workshops*, 2015. **2, 6, 8**
- [27] H. Li, Y. Li, and F. Porikli. DeepTrack: Learning discriminative feature representations by convolutional neural networks for visual tracking. In *BMVC*, 2014. **1**
- [28] H. Li, Y. Li, and F. Porikli. Convolutional neural net bagging for online visual tracking. *Computer Vision and Image Understanding*, 153:120 – 129, 2016. **1**
- [29] L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. *arXiv preprint arXiv:1603.06560*, 2016. **2, 3**
- [30] Y. Li and J. Zhu. A scale adaptive kernel correlation filter tracker with feature integration. In *ECCV Workshops*, pages 254–265, 2014. **7**
- [31] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015. **2, 3, 5**
- [32] B. Ma, L. Huang, J. Shen, and L. Shao. Discriminative tracking using tensor pooling. *IEEE Trans. on Cybernetics*, 46(11):2411–2422, 2016. **2**
- [33] B. Ma, J. Shen, Y. Liu, H. Hu, L. Shao, and X. Li. Visual tracking using strong classifier and structural local sparse descriptors. *IEEE Trans. on Multimedia*, 17(10):1818–1828, 2015. **2**
- [34] C. Ma, X. Yang, C. Zhang, and M.-H. Yang. Long-term correlation tracking. In *IEEE CVPR*, pages 5388–5396, 2015. **1, 3, 7**
- [35] M. E. Maresca and A. Petrosino. Clustering local motion estimates for robust and efficient object tracking. In *ECCV Workshops*, pages 244–253, 2014. **8**

- [36] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015. **2, 3**
- [37] H. Nam and B. Han. Learning multi-domain convolutional neural networks for visual tracking. In *IEEE CVPR*, 2016. **1, 8**
- [38] J. Ning, J. Yang, S. Jiang, L. Zhang, and M.-H. Yang. Object tracking via dual linear structured svm and explicit feature map. In *IEEE CVPR*, pages 4266–4274, 2016. **7**
- [39] M. Plappert. keras-rl. <https://github.com/matthiasplappert/keras-rl>, 2016. **6**
- [40] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015. **6**
- [41] J. Shen, D. Yu, L. Deng, and X. Dong. Fast online tracking with detection refinement. *IEEE Trans. on Intelligent Transportation Systems*, 19(1):162–173, 2018. **2**
- [42] 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, 5**
- [43] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller. Deterministic policy gradient algorithms. In *ICML*, pages 387–395, 2014. **2, 3**
- [44] J. Snoek, H. Larochelle, and R. P. Adams. Practical bayesian optimization of machine learning algorithms. In *NIPS*, pages 2951–2959, 2012. **2**
- [45] J. Snoek, O. Rippel, K. Swersky, R. Kiros, N. Satish, N. Sundaram, M. Patwary, M. Prabhat, and R. Adams. Scalable bayesian optimization using deep neural networks. In *ICML*, pages 2171–2180, 2015. **3**
- [46] J. Supancic III and D. Ramanan. Tracking as online decision-making: Learning a policy from streaming videos with reinforcement learning. *arXiv preprint arXiv:1707.04991*, 2017. **2, 4, 5**
- [47] C. Thornton, F. Hutter, H. H. Hoos, and K. Leyton-Brown. Auto-weka: Combined selection and hyperparameter optimization of classification algorithms. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 847–855. ACM, 2013. **3**
- [48] J. Valmadre, L. Bertinetto, J. F. Henriques, A. Vedaldi, and P. H. Torr. End-to-end representation learning for correlation filter based tracking. *arXiv preprint arXiv:1704.06036*, 2017. **1, 2, 3, 4, 6, 7**
- [49] T. Vojir and J. Matas. The enhanced flock of trackers. In *Registration and Recognition in Images and Videos*, pages 113–136. Springer, 2014. **8**
- [50] T. Vojir, J. Nuskova, and J. Matas. Robust scale-adaptive mean-shift for tracking. *Pattern Recognition Letters*, 49:250–258, 2014. **8**
- [51] L. Wang, W. Ouyang, X. Wang, and H. Lu. Visual tracking with fully convolutional networks. In *IEEE ICCV*, pages 3119–3127, 2015. **2**
- [52] M. Wang, Y. Liu, and Z. Huang. Large margin object tracking with circulant feature maps. *arXiv preprint arXiv:1703.05020*, 2017. **3**
- [53] Y. Wu, J. Lim, and M.-H. Yang. Online object tracking: A benchmark. In *IEEE CVPR*, pages 2411–2418, 2013. **1, 2, 6, 7**
- [54] W. Yi, L. Jongwoo, and M.-H. Yang. Object tracking benchmark. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 37(9):1834–1848, 2015. **6, 7**
- [55] S. Y. J. C. Y. Yoo, K. Yun, and J. Y. Choi. Action-decision networks for visual tracking with deep reinforcement learning. In *IEEE CVPR*, 2017. **4, 5**
- [56] D. Zhang, H. Maei, X. Wang, and Y.-F. Wang. Deep reinforcement learning for visual object tracking in videos. *arXiv preprint arXiv:1701.08936*, 2017. **4**
- [57] J. Zhang, S. Ma, and S. Sclaroff. Meem: Robust tracking via multiple experts using entropy minimization. In *ECCV*, pages 188–203, 2014. **7**
- [58] W. Zuo, X. Wu, L. Lin, L. Zhang, and M.-H. Yang. Learning support correlation filters for visual tracking. *arXiv preprint arXiv:1601.06032*, 2016. **3**