Received July 15, 2021, accepted August 23, 2021, date of publication August 30, 2021, date of current version September 8, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3108623

On the Use of Deep Reinforcement Learning for Visual Tracking: A Survey

GIORGIO CRUCIATA, LILIANA LO PRESTI, AND MARCO LA CASCIA
Department of Engineering, University of Palermo, 90128 Palermo, Italy

Corresponding author: Liliana Lo Presti (liliana.lopresti@unipa.it)

This work was supported in part by the Italian PRIN 2017 I-MALL Grant 2017BH297_004, and in part by the PON IDEHA Grant ARS01_00421.

ABSTRACT This paper aims at highlighting cutting-edge research results in the field of visual tracking by deep reinforcement learning. Deep reinforcement learning (DRL) is an emerging area combining recent progress in deep and reinforcement learning. It is showing interesting results in the computer vision field and, recently, it has been applied to the visual tracking problem yielding to the rapid development of novel tracking strategies. After providing an introduction to reinforcement learning, this paper compares recent visual tracking approaches based on deep reinforcement learning. Analysis of the state-of-the-art suggests that reinforcement learning allows modeling varying parts of the tracking system including target bounding box regression, appearance model selection, and tracking hyper-parameter optimization. The DRL framework is elegant and intriguing, and most of the DRL-based trackers achieve state-of-the-art results.

INDEX TERMS Computer vision, machine learning, video-surveillance, deep reinforcement learning, visual tracking.

I. INTRODUCTION

In the last decade, deep convolutional neural networks (CNNs) [1], [2] have consistently shown impressive performance in many vision tasks, especially object detection and recognition [3]–[5]. The success of such models seems to be ascribable to their capability of extracting higher-level features through their multiple-layer structure [6]. Despite the lack of suitable error bounds and convergence guarantees [7], [8], such approaches have greatly outperformed hand-crafted features and relieved the expert from the burden of designing ad-hoc features depending on the problem at hand. In turn, this is permitting to reconsider well-known problems in a new light (i.e., object detection [9], visual tracking [10], pose estimation [11]).

In the field of visual tracking, the use of deep learning (DL) allows achieving much higher performance than in the past, as detailed in former surveys/reviews of the state-of-the-art [12]–[15]. Indeed, the adoption of deep models has allowed learning more discriminative target feature representation. Nonetheless, it is still unclear which deep architecture can be effective for tracking [16], [17].

The associate editor coordinating the review of this manuscript and approving it for publication was Chao Shen.

Another issue is the selection of the most suitable training strategy to use. The deep model can be retrained at test time (eventually only partly as in [16]) by adapting the model parameters to the most recent target appearance during tracking. Such a self-training strategy uses uncertain labels and can result in drifting of the tracker. Hence, approaches in [10], [18]–[20] avoided the model parameters adaptation by training a Siamese network to locate the target within a search region ideally centered in the last estimated target location. In the latter case, the model parameters do not change at test time.

All above (representative) tracking methods are based on deep models trained in a supervised way on large training datasets in which the exact annotation of the target bounding box is provided at each frame. There is a trend to make the system learn through trial-and-error approaches able to reinforce the effective system abilities and correct the wrong attitude. The main paradigm implementing such a mechanism is reinforcement learning (RL) [21]. In RL, the system is generally called the agent. The agent is characterized by a state, and it interacts with the environment by taking decisions, often called actions. Each action has consequences on the environment and the agent's state. As a consequence of the selected action, the state transits into another one, and the agent receives a reward indicating how good is the new reached state. In RL, the goal is letting the agent learn a policy, namely a function, to decide, given a state, what action is better to take to follow the optimal trajectory in the state space or, equivalently, such to maximize the expected future reward. The triplet (state, action, reward) is a sample. RL takes advantage of simulations during which the agent uses its current policy to solve the problem by taking a sequence of actions that let its state changing over time. Hence, the simulation generates a set of samples used to gradually refine the agent's policy. Under this point of view, RL is a self-learning approach. In a discrete state-action setting, the policy is a table.

The recent combination of deep and reinforcement learning yields deep reinforcement learning (DRL) [22], which has been used to solve problems where the state is continuous-value. In such a case, the learned policy is a function represented by a deep model. DRL has been applied in several computer vision problems such as visual tracking [23], activity localization [24], object detection [9], video recognition [25], and segmentation [26]. In particular, DRL-based visual tracking has grown rapidly in recent years, and different methods have been proposed for target location prediction, tracking hyper-parameter optimization, or appearance model selection. In the most common application of DRL in visual tracking, the agent has to predict the target location (bounding box) in a frame (see Fig. 1) by iteratively selecting suitable actions. The state is generally an image representing the content of the current bounding box. Actions represent transformations to the current bounding box coordinates. The agent's goal is to select a sequence of shifts and scaling transformations to center the target in the frame. The reward is modeled based on the changes to the Intersection-over-Union (IoU) of the current bounding box and the ground-truth one (the selected action can increase or decrease the IoU value).

This is not the only way in which DRL has been employed in visual tracking. In this paper, our goal is to summarize the many ways in which DRL has been used to implement novel visual object tracking strategies.

In Sec.II, we formulate the visual tracking problem and provide background information on the most adopted tracking paradigms, including recent trends in the development of deep tracking methods. In Sec. III, we explain how the tracking problem can be cast into a DRL one; In Sec. IV, we summarize RL algorithms whose evolution yielded to the current DRL frameworks. In particular, three categories of DRL algorithms are presented: value-based algorithms, policy gradient methods, and actor-critic approaches. Sec. V reviews tracking papers based on RL and DRL. Sec. VI compares the reviewed approaches by their published results on publicly available datasets, Sec. VII attempts a theoretical comparison by focusing on the main components of the surveyed works, and Sec. VIII focuses on the limitations of the analyzed algorithms and suggests future directions. Finally, Sec. IX draws conclusions by remarking on the pros and cons of using DRL in visual tracking.

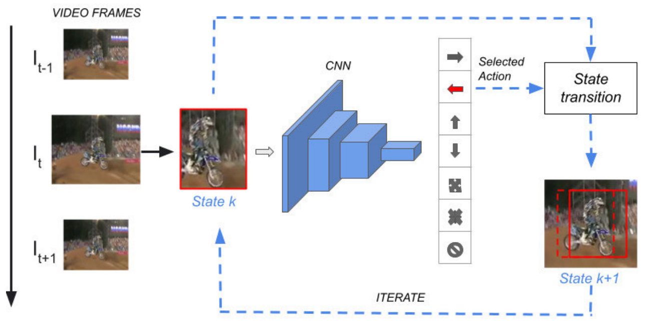

FIGURE 1. **In most visual tracking approaches based on DRL, a deep model takes in input a patch of the image, named state, and predicts a transformation of the bounding box (such as a shift to the left/right/up/down or re-scale), named action. The selected action is used to modify the bounding box and yields to a new state, which is then processed by the model. The process aims at centering the target and is iterated until a stop action is selected. The resulting bounding box will be the initial state to locate the target in the next frame.**

II. VISUAL TRACKING FORMULATION

Visual tracking has a long history that has been summarized in former surveys/reviews of the state-of-the-art [12]–[15]. In this section, we only provide basic knowledge by mostly highlighting the challenges to consider when designing a tracking algorithm. We limit our attention to methods devised for 2D visual object tracking (VOT) with a single camera.

Visual tracking is the problem of analyzing a video, namely a sequence of N images $v = \{I_1, I_2, \ldots, I_N\}$, and detecting the location l on the image plane of a target moving in the environment over time or, more formally, estimating $l_i = (x_i, y_i)$ with $i \in [1, N]$ indexing the image frames.

Visual tracking methods strongly rely on two main features: target motion and appearance. The problem is made harder in the case of moving cameras because the target motion is coupled with the camera ego-motion resulting in higher uncertainty of the target location.

Target motion dynamics have been modeled by Kalman filter [27], Extended/ Unscented Kalman Filter [28], [29], and particle filter [30] in [31]–[34]. In such approaches, appearance models were used as correction tools or to estimate the observational likelihood.

A. TRACKING-BY-DETECTION AND BY-CORRELATION

Visual tracking was profoundly changed by the development of the tracking-by-detection paradigm [35], [36], which has the merit of unifying the field of tracking in static and moving cameras formerly modeled separately. In tracking-by-detection (see Fig. 2), the goal is that of training online (meaning at test time) a detector specialized in recognizing the target in the scene. It turns the visual tracking problem into a binary classification one whose aim is that of classifying images into target/non-target. In the most simplistic form, its main ingredients are a strong feature representation, a classifier, an online procedure to re-train the classifier at

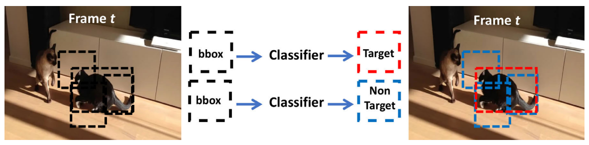

FIGURE 2. In tracking-by-detection, feature representations of image patches sampled around the last known target location are extracted, then the classifier is used to discriminate between target/non-target.

test time, a strategy to assemble an appropriate training set of positive and negative samples collected at test time. Tracking-by-detection algorithms suffered from the problem of defining suitable feature representations to discriminate between target and background. Furthermore, the self-training strategy used to re-train the classifier with uncertain labels generally results in the drifting of the tracker.

Another approach to visual tracking is that of learning the template that allows locating the target in the training images. This idea is at the basis of the correlation filters (CFs). CFs are trained in a discriminative way to estimate a template that, when correlated to the image, responds to the target rather than to the background [37]. Learning may be done in the Fourier space where correlation turns into an element-wise product (see Fig. 3), while supervision is provided in terms of heatmaps whose peaks are centered on the object of interest (Average of Synthetic Exact Filters (ASEF) [37]). Minimum Output Sum of Squared Error (MOSSE) [38] and spatially regularized discriminative CF (SRDCF) [39] were proposed to learn ASEF-like filters with limited samples to be specifically used in visual tracking.

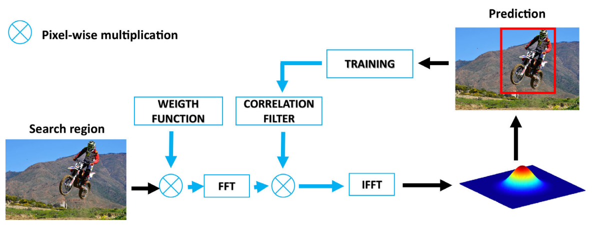

FIGURE 3. Training of a CF. The search region is element-wise multiplied with a 2D weight function, then FFT is computed and element-wise multiplied with the CF. The IFFT allows the computing of a heatmap whose peak indicates the target location. Based on such target location, an image sample set is assembled to retrain the CF.

B. DEEP VISUAL TRACKING

In recent years, deep learning techniques have been widely used in visual tracking. Several architectures, such as CNNs [40], fully convolutional neural networks [41], autoencoders [42], Siamese networks [43], Recurrent Neural Networks (RNNs) [44], [45], generative adversarial networks (GAN) [46], have all been adopted in tracking (see Fig. 4). The interested reader can refer to the following excellent surveys for more details [12]–[15]. Learning of the model parameters is, in general, done in a supervised way.

Recently, tracking models have also been trained through meta-learning [47], which aims at training a model on a variety of learning tasks. The learned models are then used to solve a novel learning task given a reduced set of training samples. The idea has been adopted to the training of recurrent networks [48]. Recently, a model-agnostic meta-learning approach (MAML) [49] has been introduced to extend meta-learning to models other than RNN.

The time required for training deep models is, in general, very high, which conflicts with the need to adapt the deep model in visual object tracking at test time. Under this point of view, deep tracking techniques have used either test-time model adaptation strategies or pre-trained models for which parameters are never updated at test time.

1) TRACKING BY TEST-TIME MODEL ADAPTATION

First attempts to adapt deep models to the target appearance changes were re-training the model (or part of it) from time to time on positive/negative patches cropped from each frame around the target location, [50], [51].

In multi-domain tracking (MDNet) [16], fully-connected (FC) layers are re-trained at test-time to discriminate between target/non-target and extract target-specific features. Convolutional layers are pre-trained on multiple videos to extract shared, general feature representations. In real-time MDNet (RT-MDNet) [52], the tracking process is speed-up by including a RoiAlign layer [5] to improve target localization during tracking.

In [53], discriminative CFs are trained on convolutional features from a pre-trained VGG model. In [54], CFs are adaptively trained on the outputs of each CNN layer to use semantics and fine-grained details for handling large appearance variations.

Recently, in ATOM [55], three modules are used. The target estimation module is trained offline to predict the IoU overlap with the target. From this output, target-specific appearance information is extracted. The IoU predictor module receives such target features and proposal bounding boxes in the test frame and estimates the IoU for each input box. Finally, the target classification module is trained, at test time, to output target confidences in a fully convolutional way.

In Vital [56], adversarial learning is used to augment data in the feature space rather than in the image space. The main goal is that of exploiting robust features over time to track the target. At test time, the model is incrementally updated frame-by-frame.

Meta-learning has been applied to the visual tracking problem in [57]–[60]. The work in [61] (DiMP) proposes a two branches network; one of the branches predicts a model of the target appearance in terms of weights of a convolutional layer. The method has been improved in [62] by integrating a regression formulation into both branches of DiMP. The work in [60] adopts MAML [49] to turn a general-purpose object detector into a target-specific one by training on a single image at test time.

2) TRACKING BY AD-HOC PRE-TRAINED MODELS

Approaches in [10], [18]–[20] have pre-trained a Siamese network to locate the target within a search region. Once trained, the model is not fine-tuned on the target appearance.

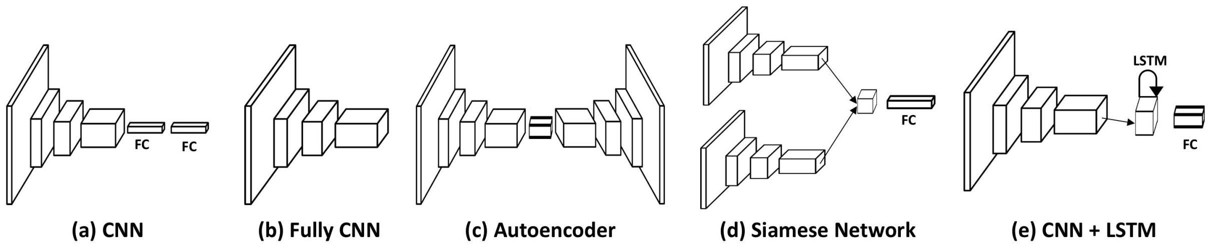

FIGURE 4. The most commonly adopted architectures for tracking. (a) CNN model where convolutional layers are followed by fully connected (FC) ones. (b) Fully convolutional model can output feature maps. (c) Autoencoder can encode and decode its input. (d) In the Siamese network, the two branches share the same architecture and parameters. A layer is used to merge the branch outputs. Additional FC layers can be also included. (e) LSTM works like a memory of its inputs (generally from a CNN). Additional FC layers can be added to the model.

At test time, the network can predict the target's bounding box by ranking proposed candidate boxes [18], regressing it directly from images [10], or estimating its center position and scale [19], [20].

YCNN [20] combines shallow features (from the first convolutional layer) with deeper features. The Siamese network returns a prediction map in which each point indicates how likely the target appears in the search image.

In the Siamese region proposal network (Siamese-RPN) [63], a template branch and a detection branch are trained end-to-end on a large dataset of image pairs. The template branch encodes the target appearance information into the RPN [64] feature map. At test time, meta-learning is used by reinterpreting the output of the template branch as parameters to predict the detection kernels.

In [65] (Siam-RPN++), it is noted that the performance of the Siamese network-based tracking algorithm can improve when using deeper networks if some precautions are taken in their development. In particular, a novel sampling strategy is proposed, and multi-branches features are extracted from various layers of the network to infer the target location.

Instead of using a Siamese net, the work in [66] uses a conv-LSTM on top of a fully convolutional CNN. The network takes in input a cropped target image, and the LSTM produces a target-specialized filter. The target location is predicted by convolving the filter to the feature map of the next frame to estimate the target response map.

III. MODELING VISUAL TRACKING BY DRL

Reinforcement Learning (RL) is an approach focused on goal-directed learning from interactions. Problems solved by RL are modeled as Markov decision processes (MDPs), which are discrete-time stochastic processes modeling sequential decision making.

In a typical RL setting, shown in Fig. 5, there is an agent that interacts with the environment at discrete time steps, $t = 0, 1, 2, \ldots$. At each time t, the agent decides on the action to perform, A_t. The decision depends on the state S_t encoding the agent's perceived environment (whatever information is available to the agent about the environment). During training, once the action has been taken, the agent receives a signal called reward, R_{t+1} measuring the goodness of the taken decision in state S_t. Furthermore, the action changes the

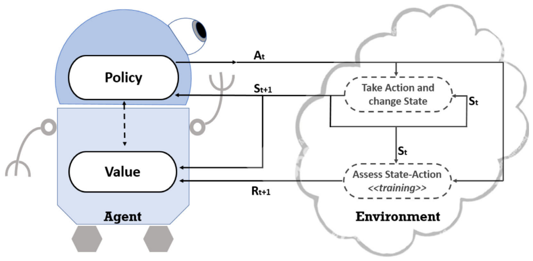
Policy
Value
Agent
A_t
S_{t+1}
R_{t+1}
Take Action and change State
S_t
S_t
Assess State-Action <<training>>
Environment

FIGURE 5. In RL, an agent interacts with the environment. Information about the environment at time t is encoded in the state S_t. Based on the learned policy, the agent takes the action A_t, which will have an impact on the environment and will determine the new state S_{t+1}. During training, the agent receives a reward R_{t+1} depending on A_t and S_t. The reward is used to modify the agent's value function and the policy.

environment, which transits in a new state encoded by S_{t+1}. In such a setting, the future state depends only on the current state S_t and the taken action A_t (Markov property).

As shown in Fig. 6, RL can be used to model the visual tracking problem in several ways. In most cases, the tracker itself is a RL-agent predicting the future target location (specifically the bounding box); in others, the agent solves auxiliary tasks to improve the tracking strategy or takes decisions affecting the target appearance model. All RL approaches for visual tracking differ mostly in the definition of the goal and, consequently, of the state and the action set.

In most of the approaches, the state is an image patch extracted based on the hypothetical target bounding box, i.e. the estimated target location at the previous frame. The discrete action set represents possible changes to the bounding box coordinates through horizontal/vertical shifts and scale adjustments. At each frame, iteratively, the bounding box is refined based on the sequence of actions taken by the agent until a stop action is selected, and the final estimated target location is used to process the next frame. The action may also assume continuous values correlated with the number of pixels or scale factor to be used to refine the bounding box of the target. In this case, each frame is processed only once, namely the sequence of actions taken by the agent has a length equals to the number of frames in the video. The state can be augmented by also considering a template of the target, which is useful in Siamese network-based methods.

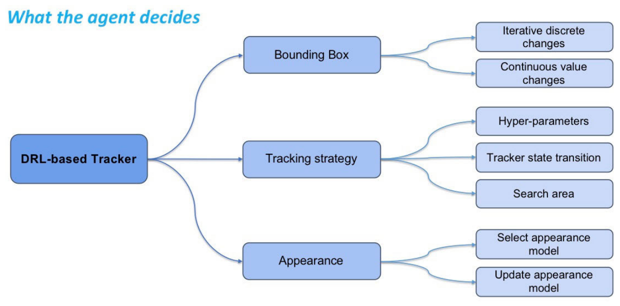

FIGURE 6. The agent can make decisions about the target bounding box, the tracking strategy to adopt, or the target appearance model. The bounding box can be refined by iterative discrete changes or by continuous values changes. Agents can also select the tracking hyper-parameters, the search region where to locate the target, or, especially in multi-object tracking, the tracker state, namely re-initialize, update or delete the tracker from the pool of active ones. Finally, the agent can decide when to update the appearance model or what model to select from a pool of available ones.

Another way to use DRL for tracking is letting the agent accounting for the target appearance changes. In this case, the action set represents either the selection of the target template to be fed in input to the model or of the correlation filter to use from a pool of pre-existing filters or of a specific tracker from an ensemble.

RL-agent can also be used to decide the most suitable tracking strategy. For instance, the agent can decide to track or re-detect the target or to enlarge the search area to improve target localization. There are also interesting approaches where the agent has to decide the (continuous value) hyper-parameters (i.e., scale step, learning rate, window weight, etc.) on which the tracking results greatly depend.

In all the above methods, it is necessary to extract features from an image, and a deep model is adopted (generally a Siamese network or a CNN sometimes coupled to a recurrent model). The extracted features are fed in input to some other (generally small) network representing the policy of the agent. During training, the agent receives a reward generally defined based on the intersection-over-union of the estimated bounding box and the true bounding box. As we will discuss later, this means that even if the agent is not trained to regress the bounding box, still the bounding box annotation may be necessary to compute the reward. However, the reward function can be learned from data.

A deeper analysis of DRL-based tracking approaches has highlighted some implementation details that can be decisive for the success of these algorithms, such as the augmentation of the state with the sequence of actions selected by the agent in previous iterations. In the following, we will describe the above-mentioned and other tracking approaches more in detail. To fully describe how such methods have been implemented, we first provide an introduction to the most adopted RL algorithms and to their modification in presence of deep models defining the agent's policy.

IV. FROM RL TO DRL

In RL, the sequential decision problem is cast into an optimization one where the agent learns to take actions by maximizing the expected discounted reward obtained for its future decisions:

$$G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{1}$$

where γ is a discount rate $0 \leq \gamma \leq 1$. Since each reward depends on the state and the selected action, the goal of the agent is to determine a *policy* π, namely a mapping from the perceived state to the action to take when in the given state. Hence, the policy fully defines the agent's behavior. When states and actions are both discrete, the policy can be designed as a lookup table. Whenever states are continuous values, the policy can be defined in terms of continuous variable functions.

Let S and $A(s)$ be the sets of states and actions that the agent can take in the state $s \in S$ respectively. The *value function* (V-function) is defined as

$$\begin{aligned} v_\pi(s) &= \mathbb{E}_\pi[G_t|S_t = s] \\ &= \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}|S_t = s\right] \\ &= \mathbb{E}_\pi\left[R_{t+1} + \gamma G_{t+1}|S_t = s\right], \forall s \in S. \end{aligned} \tag{2}$$

Such a function depends on the immediate reward for the taken action and on the discounted value of future rewards. In other words, the value function $v_\pi(s)$ measures how good is for the agent being in state s.

The *action-value function*, denoted by $q_\pi(s, a)$ and called also *Q-function*, is the agent's expected return when, after selecting action a while in the state s, it starts following the policy π thereafter:

$$\begin{aligned} q_\pi(a, s) &= \mathbb{E}_\pi\left[G_t|S_t = s, A_t = a\right] \\ &= \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}|S_t = s, A_t = a\right], \\ &\quad \forall s \in S, a \in A(s). \end{aligned} \tag{3}$$

Value functions are used to establish if a policy is optimal. When following an optimal policy, we can write the optimal value function q_* in terms of the optimal value function v_* as follows:

$$q_*(s, a) = \mathbb{E}\left[R_{t+1} + \gamma v_*(S_{t+1})|S_t = s, A_t = a\right]. \tag{4}$$

The optimal value function in the finite MDP satisfies the Bellman equation, and dynamic programming-based or heuristic search approaches can be devised to learn the optimal value function from experience [21], [67]. Such methods are called model-based approaches. Model-free methods do not rely on a model of the environment and are explicitly trial-and-error learners generally based on Monte Carlo and temporal-difference methods.

A. REWARDING THE AGENT

In RL, the reward is independent of what the agent's correct action should be. The correct action is, in general, unknown, and the reward assesses the agent's progress in achieving its goal. Designing the reward function is crucial for the success of RL algorithms, and sometimes a learning acceleration is achieved by providing an initial guess for the value function or by "behavioral shaping" where the reward function is refined with the agent's learning progresses [21], [68]. The reward can also be assigned by comparing the agent's decisions to those of an "expert", which can be represented by either another agent or a trained system. This strategy is often called imitation learning, learning from demonstration, and apprenticeship learning.

In the simple tracking strategy represented in Fig.1, the reward should measure the effect of the action taken by the agent on the tracking results. Often, a function of the IoU of the new bounding box and the ground-truth one is used.

Finally, *inverse reinforcement learning (IRL)* learns an unknown reward function under which the expert's behavior is optimal. Eventually, using direct reinforcement learning and the learned reward function, it is possible to learn an optimal policy for the RL agent. However, IRL problems are in general ill-posed as every policy is optimal for the null reward. Furthermore, there might be many reward functions under which the experts' behavior is optimal.

B. POLICY LEARNING

Three families of model-free approaches can be identified: value-based methods, policy gradient methods, and actor-critic approaches.

If the tracking strategy in Fig. 1 implements a value-based approach, then, given the image patch corresponding to the current bounding box, the deep model would return the q-values associated with each possible action, that is the expected discounted future reward. The agent would select the action (bounding box transformation) yielding the highest q-value. If a policy gradient approach is adopted, the deep model returns a probability distribution over the actions. The agent would select the action with the highest probability. Finally, if an actor-critic approach is used, the actor would return a probability distribution over the actions. The agent would select the action with the highest probability. The image patch corresponding to the new transformed bounding box can be assessed by the critic to get a confidence score or the V-function value.

1) VALUE-BASED APPROACHES

Value-based approaches aim at learning iteratively the policy by estimating the Q-function. Given the current estimate of the Q-function and a state s, the action a is chosen based on a policy.[1] Once a has been taken, the Q-function is updated.

[1] The action a might be selected by an ϵ-greedy policy: randomly from the set of possible actions with probability ϵ or $a = \arg\max_a Q(s, a)$ otherwise. Q-functions different than the learned one can be adopted as well. This strategy helps the agent to trade-off between the need of exploiting its experiences and exploring new possible solutions.

Learning the policy can be achieved by adopting Monte Carlo-based approaches; However, the resulting learning process can be slow because the Q-function is updated after the whole sequence of decisions is observed. In contrast, in Temporal-Difference (TD) learning, updates are done at every step (namely, after an action is taken) without waiting for the whole sequence of decisions to be observed. A popular TD-learning approach is the Q-learning algorithm [69].

In Q-learning, the Q-function is updated independently of the actual future action a' selected in s':

$$Q(s, a) = Q(s, a) + \alpha \left[r + \gamma \max_{a'} Q(s', a') - Q(s, a) \right]. \quad (5)$$

However, since action selection and Q-function updating are both based on the maximum of the Q-function itself, the learning process can led to a biased estimation. To make the method more robust, in *double Q-learning* algorithms, two independent estimates of the Q-function, Q_1 and Q_2, are maintained. Actions can be selected via the Q_1 function and assessed by the Q_2 function.

The former techniques can be generalized by replacing the tabular functions with functions parameterized by a weight vector $w \in R^d$, such as deep models. This yields to Deep Q-Learning (DQL). Given a policy π, if the true action-value function $q_\pi(s, a)$ is known, then the approximate action-value function $\hat{q}(s, a, w)$ could be learned by minimizing the mean squared value error defined as

$$L(w) = \sum_{s,a} \mu(s, a)[q_\pi(s, a) - \hat{q}(s, a, w)]^2 \quad (6)$$

where $\mu(s, a)$ represents the state-action distribution. Given a training set in which state-action pairs appear with a given distribution μ, the above error can be minimized by stochastic gradient descent (SGD). The target action-value $q_\pi(s_t, a_t)$ can be substituted with the discounted expected return G_t or with the target value $y_t = \mathbb{E}[r + \gamma \max_{a'} \hat{q}(s', a', w_t)]$, which is its approximation. The advantage of the former model is that it accommodates for continuous value states, while actions are, in general, still discrete values.

In practice, the above training strategy fails because the sequential states are strongly correlated, and the target value is changing during training. This leads to a divergence of the Q-function. To account for such issues, in [70], it has been proposed to endure new updates while also exploiting previous experience. This mechanism is known as *experience replay*, where updates are not done sample-by-sample but through batches of transition experiences collected in the form of (s, a, r, s'). The work in [22] applies DRL to train networks end-to-end directly from visual data by sampling mini-batches of tuples (s, a, r, s') from the replay memory. As an alternative to experience replay, multiple agents were asynchronously executed in parallel to decorrelate the data used to update the model weights in several RL approaches, including deep Q-learning [71].

As for the changing target value y_t from iteration to iteration, a target network different than the optimized one can be used [22] and updated by the learned network parameters every C steps [72]. Later on, in [73], *Double DQL* implements the concept of double Q-learning to avoid that action-values get overestimated.

Dueling DQL (DDQL) [74] proposes to decompose the Q-value in the sum $Q(s, a) = v(s) + A(s, a)$, where $v(s)$ is the state value while $A(s, a)$ measures the advantage of taking the action a in the state s. Finally, in [75], a variant of the Q-learning algorithm, *Normalized Advantage Function (NAF)*, is proposed to tackle with continuous actions.

2) POLICY GRADIENT METHODS

Policy gradient methods approximate the policy with a function depending on a learnable weight vector $\theta \in R^d$. Once θ has been learned, the policy function provides for each action a the probability $\pi(a|s, \theta)$ of taking action a when in state s. Parameters θ are learned by maximizing a performance measure $J(\theta)$ such that, by the gradient ascent algorithm, the parameters can be iteratively updated by:

$$\theta_{t+1} = \theta_t + \alpha \nabla J(\theta_t). \tag{7}$$

When a finite sequence of decisions is given, such that we know $s_0, a_0, r_1, s_1, a_1, \ldots, s_{T-1}, a_{T-1}, r_T, s_T$ with the last state s_T being a terminal state (this sequence is often called episode or trial), then the function $J(\theta)$ can be defined in terms of $v_{\pi(\theta)}(s_0)$, which is the expected return in state s_0 for all future decisions. In such a case, with $\mu(s) = \sum_a \mu(s, a)$, the policy gradient theorem [21] establishes that:

$$\begin{aligned} \nabla J(\theta) &\propto \sum_s \mu(s) \sum_a [q_\pi(s, a) - b(s)] \nabla \pi(a|s, \theta) \\ &= \mathbb{E}_\pi \left[\sum_a [q_\pi(s_t, a) - b(s_t)] \nabla \pi(a|s_t, \theta) \right] \end{aligned} \tag{8}$$

where $b(s)$ is an arbitrary *baseline* that does not depend on the action a and contributes to reduce variance of the learning approaches. The Eq. 8 is used to develop the most widely adopted algorithm for policy learning: the *REINFORCE* algorithm [21], [76]. In REINFORCE, by taking advantage of Eq. 3 and remembering that the baseline does not depend on the policy π, the gradient of the performance measure $J(\theta)$ is computed as:

$$\begin{aligned} \nabla J(\theta) &= \mathbb{E}_\pi \left[\sum_a \pi(a|s_t, \theta) [q_\pi([s_t, a) - b(s_t)] \frac{\nabla \pi(a|s_t, \theta)}{\pi(a|s_t, \theta)} \right] \\ &= \mathbb{E}_\pi \left[[G_t - b(s_t)] \nabla \ln \pi(a|s_t, \theta) \right] \end{aligned}$$

and the parameters θ are updated based on the following equation:

$$\theta_{t+1} = \theta_t + \alpha [G_t - b(s_t)] \nabla \ln \pi(a|s_t, \theta_t). \tag{9}$$

A possible choice for the baseline is that of using $b(s) = \hat{v}(s, w)$, a functional learnable approximation of the value function.

3) ACTOR-CRITIC METHODS

Actor-critic (AC) methods jointly learn approximation functions for both the policy and the value function. The 'actor' learns the policy function, and the 'critic' assesses the actor's decisions by estimating the value function $\hat{v}(s, w)$ (or, in other implementations of the method, the Q-function $\hat{q}(s, a, w)$).

A variant of AC methods is the *Deterministic Policy Gradient algorithm (DPG)* [77], where the policy is deterministic and approximated by a function $\pi(\cdot, \theta)$ parameterized such that, given the state s, action a is determined through the policy $a = \pi(s, \theta)$. DPG is the limiting case, as policy variance tends to zero, of the stochastic policy gradient [77].

In the deep versions of the AC methods, both actor and critic are typically represented by CNNs. Several variants have been proposed in the past years. The *Asynchronous Advantage Actor-Critic (A3C)* algorithm [71] maintains a policy $\pi(a_t, s_t, \theta)$ and a value function $v(s_t, w)$. Multiple agents, in parallel environments, independently follow and update the policy. This helps to break correlations among samples. A2C is a variant of the A3C algorithm that does not take advantage of asynchronous updating.

Deep Deterministic Policy Gradient (DDPG) [78] adapts DPG and the main ideas in DQL, such as using an experience replay memory and a target network, to the learning of a policy for the continuous action domain. *Twin Delayed DDPG (TD3)* [79] improves over DDPG by including double Q-learning, and delaying policy updates with respect to the Q-function updates (the Q-function is updated more frequently than the policy). In *Trust Region Policy Optimization (TRPO)* [80], a surrogate objective function, defined in terms of the advantage function and the gain of the new policy with respect to the old one, is maximized by constraining the scale of the Kullback-Leibler divergence between the old and the new policy. In contrast, in *Proximal Policy Optimization (PPO)* [81], the KL-divergence penalty/constraint is not used and the minimum between the objective function and its clipped version is maximized instead. The clip function aims at simplifying the learning process.

V. DRL-BASED VISUAL TRACKING

First attempts to use RL for visual tracking have focused on feature/appearance model selection [82], [83], PTZ camera parameter estimation [84], tracking strategy selection [85], [86], or tracking hyper-parameters estimation [87]. They have mostly adopted the Q-learning algorithm and have greatly suffered from the difficulty of representing properly the state.

An attempt to use IRL is proposed in [88], where the life of a target is modeled as a MDP. Target state values are: active, lost, tracked, inactive. State transitions are deterministic and pre-defined; actions represent the switch from a state to another. The trained reward function is the confidence returned by a classifier/detector and depends on the current target state. A similar idea is employed in [89], where the agent also decides whether or not to update the discriminative Correlation Filter (CF) used to detect the target. Hierarchical discriminative CF (HCF) [54] are learned from convolutional features computed by a pre-trained CNN. The merit of these works is that of modeling the visual tracking problem as a sequential decision one. However, the use of RL for tracking has become popular after the introduction of DRL.

In the following, we summarize DRL-based visual tracking approaches based on the categorization shown in Fig. 6. When summarizing such works, we also refer to the categorization in Fig. 7 where the same works are analyzed under the DRL algorithm used to implement the method. As shown in the figure, methods can take advantage of some form of adaptation at test time (re-training of part of the network or of the employed CFs) or can use models pre-trained for tracking purposes without parameters adaptation at test-time (blue and yellow circles respectively). The methods differ for the adopted RL framework: deep Q-learning (DQL), deep policy gradient (D-PG) and actor-critic (AC) approaches.

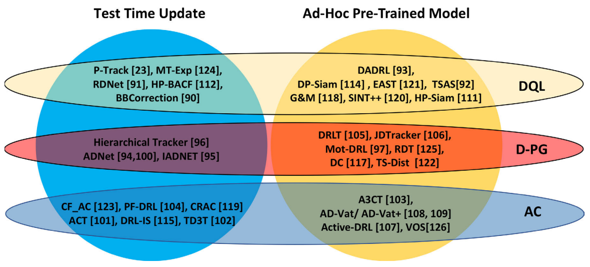
Test Time Update
Ad-Hoc Pre-Trained Model
P-Track [23], MT-Exp [124], RDNet [91], HP-BACF [112], BBCorrection [90]
DADRL [93], DP-Siam [114] , EAST [121], TSAS[92] G&M [118], SINT++ [120], HP-Siam [111]
DQL
Hierarchical Tracker [96] ADNet [94,100], IADNET [95]
DRLT [105], JDTracker [106], Mot-DRL [97], RDT [125], DC [117], TS-Dist [122]
D-PG
CF_AC [123], PF-DRL [104], CRAC [119] ACT [101], DRL-IS [115], TD3T [102]
A3CT [103], AD-Vat/ AD-Vat+ [108, 109] Active-DRL [107], VOS[126]
AC

FIGURE 7. Categorization of DRL-based visual tracking approaches. Methods can take advantage of some form of test time adaptation or can be ad-hoc pre-trained (blue and yellow circles respectively). The methods differ for the adopted RL framework: deep Q-Learning (DQL), deep policy gradient (D-PG), and deep Actor-Critic (AC) methods.

Where not diversely specified, the agent's reward is defined as a function of the Intersection-over-Union (IoU) of the predicted and annotated target bounding boxes. Sometimes, the improvement of the IoU measured before and after taking action a_t is used. In some approaches, pre-training of the neural networks is done in a supervised form. In this case, the instantaneously optimal action to select is derived based on the ground truth available with the training data.

A. DECIDING BOUNDING BOX ADJUSTMENTS

The target bounding box can be adjusted frame-by-frame by a sequence of discrete actions representing bounding-box shifts/scaling, or by a single continuous-value action representing changes to the bounding box coordinates.

1) ITERATIVE, DISCRETE BOUNDING BOX ADJUSTMENTS

When iterative, discrete bounding box adjustments are operated, the agent's action determines the direction (left/right/top/down) of the bounding box shift and/or the scale factor (enlarge/shrink). Actions are applied to the bounding box on the same frame until a stop action is taken.

The methods in [90]–[93] adopt DQL [22] to train the agent, while [94]–[97] use the REINFORCE algorithm [76] (D-PG). The methods take all advantage of test-time updates of the model with the only exception of [92], [93], [97] that use an ad-hoc pre-trained tracking model.

In [90], the method, which we name *BBCorrection*, adapts the work in [9] for object detection to visual tracking. Given an initial detection of the target, a pre-trained deep network (derived from the VGG-16 model [98]) is used to extract features from the bounding box. These features, together with a memory of the last 10 actions, represent the state; the agent is trained to shift and scale the bounding box to the target location by multiple interactions at a pixel level; Whenever the agent gets stacked into a local optimum (i.e., the agent selects pairs of actions canceling each other), the bounding box is randomly moved.

The work in [91], *RDNet*, adopts two agents: a movement and a scaling agent, both implemented as a Siamese network receiving in input the current state and a search region. The agents are individually trained by DDQN [74]. Motivated by the way humans would accomplish the same task, first, the bounding box is moved to center the target by a sequence of shifting actions of the movement agent. Later, the bounding box is re-scaled by the scaling agent. The movement agent is rewarded based on the improvement of the Euclidean distance before and after the selected action is performed. Differently from other approaches using the Siamese network, where inputs are cropped from different frames, RDNet takes inputs from the current frame.

The method in [93], *DADRL*, takes advantage of the cooperation of two different agents for deformable face tracking: the agent is implemented by a CNN and aims at predicting horizontal/vertical shifts and scale changes to the current bounding box for centering the target face. The alignment agent estimates the face landmarks and is based on the hourglass network [99] for pose estimation. It takes the decision of continuing to adjust the face region or stopping. The landmark detection accuracy is used as a reward.

The work *TSAS* [92] uses two networks in cascade to predict how to shift the target bounding box. The first network predicts the best discrete action to take to locate the target; this network is trained in a supervised way. The second network models a Q-function and is trained by RL to assess the action value. Finally, a regressor is used to refine the estimation.

In [94], [100], an action-decision network (*ADNet*) is used to predict the action probability distribution. The state is represented by the cropped target image and the history of the last 10 selected actions represented by a one-hot encoding. The policy network is pre-trained in a supervised way. The network also estimates a confidence score, which is used to decide when the tracking has to be re-initialized. At test time, a supervised adaptation of the latest fully connected layers is performed to make the model more robust to appearance changes. ADNet was later improved in *IADNet* [95]. At training time, the action is sampled from the estimated action distribution to ensure higher exploration. During training, a multi-domain learning strategy [16] is used. An online adaptive update strategy based on meta-learning is used to estimate the most appropriate model parameters such that the parameters are closer to the optimal ones.

The work in [96] proposes a *hierarchical tracker* composed of an ad-hoc pre-trained motion model (CNN and LSTM), and an appearance model based on HCF [54] that is updated at test time. After a sequence of actions, the resulting target image is processed by the HCF to get a coarse-to-fine target verification. The peak in the correlation map is chosen as the new target location. In this work, only the HCF is adapted at test time. In their ablation study, the authors show that the RL-based motion model contributes to the tracking performance.

In [97], a set of single-target trackers are used (*Mot-DRL*) to iteratively adjust the target bounding box. The state is represented by a cropped image and the last 10 selected actions.

2) CONTINUOUS-VALUE BOUNDING BOX ADJUSTMENTS

The works in [101]–[104] adopt the Actor-Critic framework to handle continuous-value action prediction, while [105], [106] adopt the REINFORCE algorithm [76]. Among the former works, only the methods in [101], [102], [104] take advantage of test-time adaptation of the model.

In the Actor-Critic Tracker (ACT) [101] and in TD3T [102], the state is defined as the cropped target image based on the bounding box detected at the previous frame. The actor-network provides continuous actions to update the past bounding box to the new target location in the current frame. The critic-network takes in input the state and the selected action and provides the Q-values and a verification score representing the decision reliability. During tracking, the actor is fine-tuned in a supervised way through the annotation in the first frame. The critic is updated any 10 frames based on the collected target detections. The whole framework is trained by DDPG [78]. In *TD3T* [102], the policy is learned by means of the TD3 algorithm [79], which aims at getting more accurate Q-values estimation by using double Q-learning. Hence, during training, two critic networks assess the actor's decisions; at test time, one of the two critics is used to estimate the bounding box confidence. The training and updating strategies in TD3T are similar to those of ACT [101].

In [104], a motion-aware RL agent guides the particle sampling within the particle filter framework (*PF-DRL*). In this sense, the work revisits the MDNet model [16], which tracks adaptively the target by evaluating a number of samples drawn around the past target location. The motion-aware network takes in input the cropped target image and the action history and provides in output a continuous action representing parameters to estimate the mean and covariance matrix of a Gaussian distribution to be used to sample the particles. The critic network estimates the Q-value of the selected action. The model is trained by DDPG [78].

Also *A3CT* in [103] proposes to train the RL agent under the guidance of an expert tracker. A Siamese network and a LSTM fed the actor and the critic, which estimate continuous value actions to update the target bounding box and the V-function respectively. The expert tracker (SiamFC [19]) performs tracking by generating a sequence of states (estimated target locations) and actions (changes to the target locations). The agent learns to behave better than the expert. As a variant, in A3CTD the tracker takes advantage of the expert tracker also at test time. The interesting aspect of this work is that the rewards are computed independently of the ground truth.

In [105], a DRL-based Tracking algorithm (*DRLT*) adopting a recurrent CNN is proposed. Given in input the whole frame, the CNN extracts a feature representation. Such representation is augmented with the target location and fed in input to the RNN. The RNN approximates the policy function and predicts the new target location. The main advantages of this approach are: the tracker can use contextual information since the whole frame is processed; the tracker predicts the current bounding box without any iterative procedure; no action history is needed since dynamics are embedded in the RNN. However, training RNN requires a large dataset.

The work in [106] proposes *JDTracker*, a policy-based jump-diffusion process for visual tracking. Stochastic jump-diffusion processes are used to sample a probabilistic distribution over a mixture of subspaces of varying dimensions. The jump allows moving from a subspace to another. In JDTracker, the state decomposes into a discrete subspace encoding the target visibility (target visible, occluded, or invisible), and a continuous subspace encoding the target location. The method learns two sub-policies (two CNNs), the first designed on the discrete subspace and the other on the continuous sub-space. Each CNN estimates a distribution probability over the discrete actions (i.e., switching into a new discrete state) or the continuous value changes to the bounding box (shift and scale values). For discrete actions, the reward is 1 if the visibility state equals the ground-truth, 0 otherwise. Training is performed by a variant of the REINFORCE algorithm [76].

3) CAMERA PARAMETERS ADJUSTMENTS

There are also works [107], [108] on active tracking that model actions as continuous value camera parameters changes. They both employ the actor-critic framework and do not adapt the model at test time.

In [107], [109], the tracker (*Active-DRL*) consists of a CNN, a LSTM, and an actor-critic network to derive the agent's action and estimate the state value (V-function). Given the location and orientation of the target on the image plane, the reward function is designed such that its maximum value is achieved when the target stands in front of the camera within a predefined distance and no rotation intervenes. The method is tested only in a virtual environment, and the model is trained from scratch without any supervision.

Virtual environments are also used in Asymmetric Dueling Visual Active Tracking (*AD-Vat*) [108], [110], which proposes an adversarial RL method involving two agents: the tracker and the target agents playing the role of opponents during the training. Thanks to the adopted adversarial learning strategy, the target attempts to find the weakness of the tracker, and the tracker becomes stronger. The reward function is 0 near range (when target location and tracker estimate are close) and non zero otherwise to penalize the target agent when it runs too far from the tracker estimation.

B. DECIDING THE TRACKING STRATEGY

In this section, we discuss works where the agent decides the tracking hyper-parameters, the search region where to locate the target, or the tracker state transition. The latter may imply reinitializing, updating or deleting the tracker.

1) TRACKING HYPER-PARAMETERS ADJUSTMENTS

In [111] (*HP-Siam*), it is noted that hyper-parameters, such as scale step, scale penalty, scale learning rate, window weight, and template learning rate, also play a crucial role in tracking processes, and dynamically changing such hyper-parameters at a frame level may lead to a great improvement of the tracking results. Ground-truth values of hyper-parameters are not available. A policy network estimates the hyper-parameter values, and another network assesses the Q-function value. Learning is accelerated by using NAF [75]. The work is extended in [112] (*HP-BACF*) where the proposed network is combined with a real-time correlation filter-based tracker [113].

A hybrid approach that allows estimating both the new target location and the tracking hyper-parameters is proposed in [114], *DP-Siam*. A Siamese network produces a heatmap representing the per-pixel likelihood of finding the target in the search area. The agent network predicts the new target position in the search area. Such continuous action produces a state change represented as a novel target bounding box. The environment network approximates the Q-function and estimates the Q-value of the new state and a confidence value. The Q-network also provides the tracking hyper-parameters (scale penalty, scale learning rate, window weight, template learning rate). Learning is done by a modified Q-learning approach that uses alternate training of the agent and environment networks.

2) TRACKER STATE TRANSITION

Methods in [23], [115] learn a policy to decide when to track the target/reinitialize the tracking, and if to update the target appearance model. The former method is based on DQL [22], the latter is based on the AC framework. Both the methods take advantage of model updates at test time.

In [23], *P-Track*, a memory of past actions (action history) is maintained and processed by the model. Training of the network is interactive, with manually annotated frames in strides of 50. The base tracker is a fully convolutional network tracker (FCNT) [116] whose parameters are adapted at test time, while the Q-function is approximated by a small network initialized by heuristically guided Q-learning through the minimization of a supervised loss function.

In the method [115], DRL with iterative shift (*DRL-IS*), three networks are used: the prediction network, the actor, and the critic. These networks share all the convolutional layers. Appearance features of the target are explicitly maintained and online updated. The prediction network iteratively adjusts the target location (shifting and scaling the target bounding box). The actor-network makes decisions on the tracking status, whether or not to update the target representation and the prediction network, or even restart tracking. The critic returns a value assessing the state-action pair. The prediction network is pre-trained in an end-to-end manner. The actor-critic model is trained considering different rewards based on the selected action. Finally, in [117], a decision controller (*DC*) implemented by a Siamese network to choose between two trackers' results is proposed. The controller, trained by REINFORCE [76], takes in input the patches corresponding to the trackers' predictions. The reward is the difference between the IoU of the selected prediction and the IoU of the other one. Another decision controller is proposed whose goal is that of deciding whether to track the target (by Siamese-FC [19]) or to detect it (by SSD [4]), namely to re-initialize the target tracker whenever drifting occurs.

3) SEARCH AREA ADJUSTMENTS

While works in [118], [119] aim at adjusting the search area through discrete shift/scale actions, the work in [120] proposes a data augmentation technique whose goal is determining the area to occlude to generate hard samples. The methods in [118], [120] adopt the DQL framework [22], while CRAC [119] uses the actor-critic one coupled with GAN [46]. This latter method takes advantage of model updating at test-time.

G&M [118] proposes to estimate the optimal search area to feed into a Siamese network-based tracker. Two modules are available: the matching and the guessing modules. The first one is a Siamese network that provides the heatmap of the new target location by cross-correlation. The guessing module provides a rough estimation of the target location. The agent has to decide how to move the bounding box such that the matching module will be able to detect the target within it.

CRAC [119] has been designed for unsupervised vehicle tracking in drone videos. The actor takes in input observations from an image and the action history. It determines the window size of the search area (the contextual region around the target) by selecting actions such as enlarge, shrink, or terminate. The critic assesses the action-state pair according to the tracking score of the newly generated images. The actor-critic model is trained by A3C [71] on the ground-view dataset.

Differently from the above approaches, *SINT++* in [120] couples reinforcement and adversarial learning to augment the training set with hard positive samples and improve the robustness of the SINT algorithm [18]. For each video, a variational autoencoder is trained and used to generate samples of the target that did not occur in training data. A hard positive transformation network allows adding occlusions on the target objects by using image patches extracted from the background. The latter module is trained by DQL [22], and the agent's actions entail the movements of the patch over the target.

C. DECIDING APPEARANCE MODEL SELECTION/UPDATE

The agent can also make decisions concerning the target appearance model management. In particular, it can select the appearance model to use from a pool of available models, can decide whether to update or not the appearance model or can decide which tracking result to consider when a pool of expert trackers is available. All these methods use discrete actions. Whenever the appearance model is represented in terms of CF [38], then the methods take advantage of the appearance model update at test time.

The EArly Stopping Tracker (*EAST*) [121], aims at speeding up deep Siamese trackers in an adaptive way by choosing the output of the optimal layer to use for tracking. It is based on the intuition that complex scenarios may require deeper features, while simpler scenarios may need the output of the first layer. The agent decides whether using the representation of the current layer or moving to the next one. The maximal number of iterations depends on the network depth, which makes the iteration number a-priori limited by the architecture structure. The method is based on DQL [22]. Somehow related is the work in [122] (*TS-Dist*), which aims at distilling a small, fast Siamese tracker from a large one by transferring knowledge from a teacher to multiple student networks. The first module extracts a reduced network representing the ''dull'' student and learns a policy to sequentially shrink the Siamese network layers. The reward function measures the compression rate and tracking accuracy. Learning is based on policy gradient methods.

The works in [123], [124] adopt CF as target appearance models, the work in [125] is concerned with the choice of the best heatmap to use in tracking, and [126] represents the target appearance in terms of template images.

In [124], *MT-Exp*, a DQL-based expert selects the best tracker to use from a pool of CFs by taking their response maps in input to estimate the reliability of each tracker (value function). The network is pre-trained in a supervised way by letting the network regress the IoU on a single heatmap. Then, RL is used to refine the network parameters.

In [123] (*CF-AC*), it is noted that older CFs may yield better results and it is proposed to maintain several past CFs. It uses the actor-critic framework to learn a policy for selecting the optimal appearance model (i.e., CF) to locate the target. The policy network takes in input all response maps and provides a beta probability distribution over all available filters. This differs from MT-Exp [124], where the state is the response map of a single CF and multiple evaluations are required. In CF-AC, the critic is used to assess the policy decision. The policy network is trained by PPO [81]. The pool of CFs includes the initial (never updated) CF, several updated CFs, and an accumulated (always updated) CF.

In *RDT* [125], policy gradient is used to select the target template from a pool of past observations. The matching between the target template and the new frame is established by a Siamese network, which produces a heatmap. The agent has to decide, for each template, if the estimated heatmap is reliable or no. The most reliable heatmap is used to locate the target. In practice, the model helps to maintain an updated and effective appearance model represented in terms of ensemble of templates.

The method in [126] (*VOS*) considers the problem of video object tracking and segmentation. Given a pool of candidate target detection obtained through an instance segmentation network, such as YOLACT [127] or Mask R-CNN [5], the AC-based agent has to decide whether to update or not the target template. The choice of which matching strategy to use between a fast one (IoU-based) and a slow but accurate one (appearance-based) is taken based on the agent's action history.

VI. QUANTITATIVE METHOD COMPARISON

We compare DRL-based visual-tracking methods by considering their reported results on publicly available datasets regarding single-object tracking.

In table 1, we group the papers based on the categorization in Fig. 6, and, for each method, we report the adopted DRL-method (DQL, D-PG, AC) described in Sec.IV, the adopted training dataset, and the achieved results on public benchmarks.

A. DATASETS AND PERFORMANCE MEASUREMENTS

Here, we provide a brief overview of the most adopted public benchmarks for single object tracking. We also summarize the experimental protocols suggested when using these benchmarks.

1) PUBLIC BENCHMARKS

Two main benchmarks are adopted to validate tracking algorithms: the Object Tracking benchmark (OTB-2013) [132], and the Visual Object Tracking (VOT) dataset [133] of which several extensions/variants are provided each year, being the dataset part of an annual challenge.

The OTB-2013 includes 50 fully annotated videos characterized by several attributes (illumination variation, scale variation, occlusion, motion blur, etc.). The benchmark was expanded in [134] to include further 50 annotated trajectories (some videos have more than one annotated object). The 100 annotated trajectories are indicated with the name OTB-100; a subset of such trajectories, named OTB-50, includes the more challenging videos and differs from the set of videos in OTB-2013.

The VOT challenges [133] started in 2013 and the publicly available dataset grew over time (25 videos for short-term tracking in 2014, 60 since 2015, additional 35 sequences for long-term tracking in 2018). Based on the final reports of each competition, we have found that: VOT 2016 used the same videos as in VOT 2015 but with a more accurate

TABLE 1. Comparison on OTB-13/15 and VOT-16/18. The column Goal reports the kind of problem solved by the method (BBOX = target Bounding Box prediction, TS = Tracking Strategy, App = Appearance model selection/update) as presented in Sec. V. The column Method reports the algorithm category (AC = Actor-Critic, D-PG = Deep Policy Gradient, DQL = Deep Q-Learning, DT = Deep Tracker, ML = Meta-learning, AL = Adversarial Learning, CF = Correlation Filter). DT and CF methods are reported as baseline methods and grouped as well based on the goal pursued by the tracker. The column Trained on reports the training dataset. AUC stands for area under the curve, P for precision, EAO for expected average overlap, FPS is the frame per second, and GPU indicates the GPU architecture reported in the published paper. In bold are highlighted the best-achieved results per column. *Results reported in [128].All GPUs are Nvidia. N.A. stands for not available.**

Goal	Method	Tracker	Trained on	OTB-13		OTB-50		OTB-100		VOT-16	VOT-18	FPS	GPU**
				AUC	P	AUC	P	AUC	P	EAO	EAO		
BBox	DQL	RDNET [91]	ILSVRC;ALOV	-	-	0.676	0.901	0.673	0.903	-	0.273	4	Titan Xp
BBox	DQL	DP-Siam [114]	ILSVRC	0.686	0.918	0.621	0.839	0.677	0.883	0.38	0.387	82	Titan Xp
BBox	DQL	TSAS [92]	ILSVRC	0.692	0.912	-	-	0.651	0.861	-	-	20	Titan Xp
BBox	D-PG	ADNet [94], [100]	VOT13-15; ALOV300	-	-	0.659	0.903	0.646	0.88	-	-	2.9	Titan X
BBox	D-PG	ADNet-Fast [94], [100]	VOT13-15; ALOV300	-	-	0.67	0.898	0.635	0.851	-	-	15	Titan X
BBox	D-PG	Hier. T. [96]	VOT-16	-	-	**0.681**	0.921	0.651	0.894	-	-	6.3	GTX-1060
BBox	D-PG,ML	IADNET [95]	VOT13-15	-	-	-	-	0.651	0.868	-	-	3.9	Titan X
BBox	AC	A3CTD [103]	ILSVRC, GOT-10k	-	-	-	-	0.535	0.717	-	0.1847	50	four Titan V and Titan Xp
BBox	AC	ACT [101]	ILSVRC	0.657	0.905	-	-	0.625	0.859	0.2746	-	30	Titan X
BBox	AC	TD3T [102]	ILSVRC	0.651	0.867	-	-	0.616	0.821	-	-	23	GTX 1080
BBox	AC	PF-DRL [104]	ILSVRC	-	-	-	-	0.666	0.898	-	-	-	GTX 1080 Ti
BBox	DT,ML	FCOS-MAML [60]	TrackingNet, COCO, GOT10k,LaSOT	0.714	-	0.665	-	0.704	-	-	0.392	40	P100
BBox	DT	ATOM [55]	TrackingNet, COCO,LaSOT	-	-	-	-	-	-	-	0.401	40	Titan X
BBox	DT	DiMP [61]	TrackingNet, COCO, GOT10k,LaSOT	-	-	-	-	0.684	-	-	**0.44**	30	GT-1080
BBox	DT	MDNet [16]	ILSVRC	-	-	0.678	**0.948**	**0.708**	0.909	-	-	46	Tesla K20m
BBox	DT	VITAL [56]	-	0.710	0.950	-	-	0.682	0.917	0.323	-	1.5	Tesla K40c
BBox	DT	SIAM-RPN [63]	Youtube-BB	-	-	-	-	0.637	0.851	0.344	0.243	**160**	GTX 1060
BBox	DT	SIAM-RPN++ [65]	COCO, ImageNet, ImageNet,YouTube-BB	-	-	-	-	0.696	0.914	-	0.414	35	Titan Xp
TS	DQL	HP [111]	ILSVRC	0.629	-	0.554	0.745	0.601	0.796	0.2726	0.201	69	N.A.
TS	DQL	HP-BACF [112]	ILSVRC	-	-	-	-	0.634	0.833	-	0.113	12	GTX 1080
TS	DQL	G&M [118]	ILSVRC	0.667	0.883	0.564	0.763	0.613	0.81	-	0.2138	68	Titan X
TS	DQL,AL	SINT++ [120]	VOT13-16	-	-	0.624	0.839	0.574	0.768	-	-	-	GTX1080
TS	D-PG	DC [117]	TinyTLP; NFS	-	-	-	-		-	**0.3942**	0.311	2	Titan X
TS	D-PG	DC-RT [117]	TinyTLP; NFS	-	-	-	-	-	-	0.3516	0.252	50	Titan X
TS	DT	C-RPN [128]	LaSOT, VID, YT-BB	0.675	0.902	-	-	0.663	0.882	0.363	0.289	36	N.A.
TS	DT	SA-SiamR-IE [129]	ILSVRC, COCO, GOT10k	0.689	0.887	0.631	0.83	0.682	0.883	-	0.311	36	N.A.
TS	CF	ECO* [130]	-	0.709	0.93	-	-	0.691	0.91	0.374	0.28	12.5	N.A.
App	AC	DRL-IS [115]	ILSVRC, VOT13-15	-	-	-	-	0.671	0.909	-	-	10.2	GTX 1080 Ti
App	DQL	MT-Exp [124]	TC-128; UAV123	0.706	0.931	-	-	0.688	0.919	-	0.293	-	Titan X
App	AC	CF-AC [123]	VID	0.652	0.851	-	-	0.623	0.812	-	-	-	Titan X
App	CF	GFS-DCF [131]	-	**0.722**	**0.965**	-	-	0.693	**0.932**	-	0.397	7.8	Titan X

annotation; VOT 2017 replaced the least challenging videos in VOT 2016 with newer sequences and further improved the annotation; videos/annotations for short-term tracking in VOT 2018/ 2019 are the same as in 2017. In the following, we summarize the results on VOT-16 and VOT-18.

Other more recent benchmarks could have been used to assess DRL-based methods, such as LaSOT [135], TrackingNet [136], TC [137], and GOT-10k [138]. Such benchmarks would have allowed validating the methods on long video sequences. The analysis of the reviewed papers revealed that G&M [118] is validated on TrackingNet and LaSot, A3CTD [103] is validated on LaSOT and Got-10k, DRL-IS [115], and PF-DRL [104] are validated on TC. Since these evaluation results are too sparse, they do not allow drawing conclusions on the effectiveness of DRL-based trackers in long videos.

For training purposes, ImageNet videos (ILSCRC) [139], ALOV300+ [140] and VOT are often used. As reported in [18], 12 sequences in the ALOV300++ overlap with the OTB dataset, and overlapping sequences are also in VOT-16. Most of the reviewed papers clearly claim to have excluded the overlapping sequences from the training set [52], [94]–[96], [115], [120]). Other adopted datasets for training purposes are reported in Table 1.

2) EXPERIMENTAL PROTOCOLS

OTB and VOT benchmarks have different evaluation protocols. For the mathematical formulation of the evaluation metrics, we refer the reader to the papers [132]–[134]. Here, we provide a high-level description of the methodology to use for assessing tracking algorithms on these two benchmarks.

In OTB, each tracker starts from an initial annotated bounding box and runs till the end of the video without re-initialization in case of tracking failure. This methodology is referred to as one-pass evaluation (OPE). Other methodologies (temporal robustness evaluation and spatial robustness evaluation) are proposed in [132] but rarely adopted in practice. In VOT, whenever a tracking failure occurs, the tracker is re-initialized.

In the OTB benchmark, two performance measures are suggested: IoU to measure tracking accuracy, and RMSE of the target center location to measure the tracker precision. The two measures are used to draw the success and precision plots respectively. The success plot represents the percentage of frames with $IoU > \kappa$ for varying thresholds $\kappa \in [0, 1]$. The area under the curve (AUC) of the success plot serves to rank the algorithms. In [141], it is proved that this AUC is in fact the average overlap (AO) over the sequence.

The precision plot shows the percentage of frames in which the target distance to the ground-truth location is below a varying threshold. The precision score with a threshold equals 20 pixels is used to rank the trackers.

In VOT, stochastic trackers are run 15 times on each sequence. Three measurements are computed [133]: the expected average overlap (EAO), accuracy (A), and robustness (R). Robustness metric evaluates the tracking failure rate; a failure occurs when the IoU is not greater than 0. In such cases, the tracker is re-initialized 5 frames after the failure by using the ground-truth, and 10 frames after the re-initialization are ignored when measuring the performances. The accuracy metric measures the AO over all successfully tracked frames. Some works report the accuracy and robustness raw scores, while others report the average ranking of the method over a set of trackers. Due to these discrepancies, we only report the EAO values.

EAO estimates the tracker accuracy by taking into account how long the tracker can successfully follow the target independently than the length of the video sequence [133]. Based on the probability density function over the sequence lengths in the dataset (computed by kernel density estimate), two length boundaries, t_i and t_f, are found such that the integral of the pdf within this range equals 0.5. Since the tracker is reset in case of failure, the tracking sequence is split into fragments (based on the frames where a failure has been detected). Fragments with a length below the video length N and not terminating in a failure are discarded. The remaining fragments are padded with zeros to have length N. The fragments are then per-frame averaged and the per-sequence average in the range $[t_i, t_f]$ yields to the EAO.

Despite both OTB and VOT benchmarks suggest to perform attribute-based analysis (i.e. measuring performance in case of illumination changes, occlusions, etc.), only very few DRL-based papers present this analysis (namely [95], [101], [111], [112], [123]). For this reason, an attribute-based comparison of the trackers cannot be done.

B. COMPARING DRL-BASED TRACKING APPROACHES

Table 1 summarizes the results of the selected papers on the OTB and VOT benchmarks. It also reports the frame rate of the algorithms and the specification of the used GPU. Whenever the paper indicated that the experiments were run on OTB-2015, we assumed the authors used OTB-100. Where the use of VOT 2017 was reported, we have indicated VOT-18 since the two datasets are the same as reported on the challenge website.

Papers not included in this table, such as [90], [93], [106]–[110], [142], are not comparable because experiments are run on different datasets or in simulation. Whenever results of variants of the algorithm are reported, we consider the best-achieved results unless the modification led to high differences in the scores. For the OTB dataset, we only include papers reporting both the AUC and precision P values (hence, we excluded [23], [121], [125]). The table does not report VOT-15 where only 3 of the papers have reported results: DP-Siam [114], HP [111], and EAST [121] whose EAO scores are 0.39, 0.242, and 0.34 respectively.

The method DRLT [105] is not included in the table because the training procedure largely differs from that of the other methods. DRLT has been trained/tested on 30 videos from the OTB-100 in cross-validation. Values of AUC, precision, and fps are 0.543, 0.635, and 270 respectively. TS-Dist [122] is not included because DRL is only used to find the student architecture but not to transfer knowledge.

In the experiments on the OTB benchmark, the work TD3T [102] reports that the threshold for the success of the tracker is 0.5 but the OTB evaluation protocol requires that the AO is reported.

We first compare DRL-based trackers independently on the pursued goal. Then we compare methods within the same category. By examining the table, especially the more complete results reported on the OTB-100, the best accuracy values are achieved by MT-Exp [124] and DP-Siam [114]. However, on the VOT-18 dataset, DP-Siam achieves much higher EAO than the MT-Exp method [124], which might indicate a lower number of tracking failures. Overall DP-Siam seems to offer a good trade-off between accuracy and tracking speed. MT-Exp [124] reports very good results on OTB but does not specify the achieved frame rate. Looking at the results, there is no clear advantage in preferring a deep model over another. DP-Siam [114] adopts a Siamese network as the base tracker and is not adapted at test time. On the contrary, MT-Exp [124] uses a CNN whose inputs are computed by re-training a set of CFs at test time. Both the methods take advantage of DQL algorithms [22], as well as RDNet [91] that achieves the highest results on the OTB-50 at the cost of a lower frame rate. Good results are achieved on the OTB-50/100 by D-PG and AC methods, namely ADNet [94] and DRL-IS [115], both online adapted.

Among the DRL-based approaches aiming at predicting the target bounding box, the most performant method on the OTB-13 is TSAS [92], on OTB-50 is the Hierarchical Tracker [96], and on OTB-100 are RDNET [91] and DP-Siam [114].

The best performing method among those dealing with the tracking strategy, on OTB-100, is HP-BACF [112], which learns the tracking hyper-parameters and adopt also CFs. On the VOT benchmark, the best results are achieved by DC [117].

As for the methods dealing with the target appearance selection/updating, the best results on OTB-100 are achieved by MT-Exp [124].

If we compare the results across categories, methods dealing with the target appearance seem to perform better and are competitive or superior to those dealing with the target bounding box prediction. On the OTB benchmark, methods dealing with the tracking strategy are the ones achieving the worst results. However, this is not confirmed by the results achieved on the VOT benchmark.

C. COMPARISON TO THE STATE-OF-THE-ART

Finally, we also compare to deep tracking methods trained in a supervised way. We have included in the table the widely known MDNet [16] as a baseline and very recent works end-to-end trainable or adopting discriminative correlation filters (CF), adversarial learning, and meta-learning.

As the table shows, on OTB-100 the majority of DRL-based trackers perform worst than MDNet. However,

TABLE 2. Theoretical comparison of DRL-based trackers grouped based on the adopted DRL algorithm. The column DRL specifies the algorithm category (DQL, D-PG and AC). The column Decisions describes the action space. The column D/C stands for Discrete or Continuous-values actions. AH stands for Action History. RNN and Siam indicate the adoption of RNN or Siamese network, respectively. The column SL refers to the possibility of pre-training the policy by Supervised Learning. The column OU indicates that, at test time, the model parameters or the CF are updated.CF stands for Correlation Filter. Finally, the column Reward describes the adopted reward function.

DRL	Tracker	Decisions	D/C	AH	RNN	Siam	SL	OU	CF	Reward
DQL	BBCorrection [90]	Shift/Scale BBox	D	✓	-	-	-	✓	-	$f(\Delta\text{IoU}(b_t, g))$
DQL	RDNET [91]	Shift/Scale BBox	D	-	-	✓	-	✓	-	$f(\Delta\text{IoU}(b_t, g), \Delta\text{Eu}(b_t, g))$
DQL	TSAS [92]	Shift BBox	D	-	-	-	✓	-	-	$f(\text{IoU}(b_t, g))$
DQL	DADRL [93]	Shift/Scale BBox & align	D	-	✓	-	✓	-	-	$f(\text{Eu}(\text{landmarks}_t, g))$
DQL	DP-Siam [114]	Predict BBox, hyper-parameters	C	-	-	✓	-	-	-	$f(\text{IoU}(b_t, g))$
DQL	EAST [121]	Scale BBOx, Use next layer	D	✓	-	✓	-	-	-	$f(\Delta\text{IoU}(b_t, g))$
DQL	G&M [118]	Shift Search Area	D	✓	-	✓	-	-	-	$f(\Delta\text{IoU}(b_t, g))$
DQL	SINT++ [120]	Shift Area to occlude	D	-	-	✓	-	-	-	$f(\Delta\text{Score}(b_t, g))$
DQL	P-Track [23]	Track/Init/Update	D	✓	-	-	✓	✓	-	$f(\text{IoU}(b_t, g))$
DQL	MT-Exp [124]	Select CF	D	-	-	-	✓	✓	✓	$f(\text{IoU}(b_t, g))$
DQL	HP-Siam [111]	Hyper-parameters	C	-	-	✓	✓	-	-	$f(\text{IoU}(b_t, g))$
DQL	HP-BACF [112]	Hyper-parameters	C	-	-	-	✓	✓	✓	$f(\text{IoU}(b_t, g))$
D-PG	ADNet [94], [100]	Shift/Scale BBox	D	✓	-	-	✓	✓	-	$f(\text{IoU}(b_t, g))$
D-PG	IADNET [95]	Shift/Scale BBox	D	✓	-	-	✓	✓	-	$f(\Delta\text{IoU}(b_t, g))$
D-PG	Hier. T. [96]	Shift/Scale BBox	D	-	✓	-	-	✓	✓	$f(\text{IoU}(b_t, g))$
D-PG	DRLT [105]	Predict BBox	C	-	✓	-	-	-	-	$f(\text{D}(b_t, g)), f(\text{IoU}(b_t, g))$
D-PG	JDTracker [106]	Predict BBox & visibility	D&C	✓	-	-	✓	-	-	$f(\text{IoU}(b_t, g)), f(\text{visibility}, g_{vis})$
D-PG	RDT [125]	Select heatmap	D	-	-	✓	-	-	-	$f(\text{IoU}(b_t, g))$
D-PG	DC [117]	Select tracking results	D	-	-	✓	-	-	-	$f(\Delta_{Tr}\text{IoU}(b_t, g))$
AC	ACT [101]	Shift/Scale BBox	C	-	-	-	✓	✓	-	$f(\text{IoU}(b_t, g))$
AC	TD3T [102]	Shift/Scale BBox	C	-	-	-	✓	✓	-	$f(\text{IoU}(b_t, g))$
AC	A3CT [103]	Shift/Scale BBox	C	-	✓	✓	-	-	-	$f(\text{IoU}(b_t, b_t^{exp}))$
AC	CRAC [119]	Scale Search Area	D	✓	-	-	-	✓	-	$f(\Delta\text{Score}_{GAN})$
AC	PF-DRL [104]	Shift/Scale Expected BBox	C	✓	-	-	✓	✓	-	$f(\text{IoU}(b_t, g))$
AC	CF-AC [123]	Select CF	D	-	-	-	-	✓	✓	$f(\text{IoU}(b_t, g))$
AC	DRL-IS [115]	Track/Init/Update	D	-	-	-	✓	✓	-	$f(a,\text{IoU}(b_t, g), \Delta\text{IoU}(b_t, g))$
AC	Active_DRL [107]	Camera parameters	C	-	✓	-	-	-	-	$f(loc_{target})$
AC	ADVat [108]	Camera parameters	D	-	✓	-	-	-	-	$f(loc_{target})$
AC	VOS [126]	Update Template	D	-	-	✓	-	-	-	$f(\text{IoU}(Mask_t, g))$

this is true also for several recent baseline methods using supervised learning (SL). The method achieving the highest tracking results is a CF-based, namely GFS-DCF [128]. The best DRL-based trackers MT-Exp [124] and DP-Siam [114] achieve slightly inferior results with respect to GFS-DCF but comparable to VITAL [56].

Comparing the DRL-methods dealing with the bounding box prediction with the corresponding baseline methods on the OTB-100, there are DRL-methods [91], [104], [114] that are competitive with deep trackers.

VII. DISCUSSION

Table 2 summarizes the main characteristics of the analyzed DRL-based tracking approaches. The column *DRL* reports the type of DRL algorithm used to train the network, namely DQL, D-PG, and AC. The column *Decisions* describes the kind of actions that the agent has to take while the column *D/C* stands for Discrete or Continuous-values actions. *AH* stands for Action History and highlights the models taking in input the sequence of past actions selected by the agent. *RNN* indicates the adoption of recurrent neural network (such as LSTM) while *Siam* indicates that the model is taking advantage of a Siamese network. The column *SL* refers to Supervised Learning. In this case, we highlight the fact that the policy has been pre-trained in a supervised way with a set of actions derived from the ground truth. The column *OU* highlights the works in which the model is updated at test time (based on the target detection collected during tracking, hence by a form of semi-supervised learning). *CF* stands for Correlation Filter and refers to the fact that CFs are employed for tracking purposes. Works employing CF are online updated. Finally, the column *Reward* describes the adopted reward function.

Learning a policy with a large state space is challenging. Most of the approaches have highlighted slow convergence and/or stability issues. Such problems have in general been addressed by introducing supervised pre-training strategies, data augmentation techniques, ad-hoc design of the reward function, or increased size of the training data. In the following, we analyze the above-mentioned issues and techniques.

A. STATE SPACE

All models take in input images or heatmaps. The number of inputs can vary depending on the adopted architecture. Images are generally resized to adapt to the input size of the adopted (generally pre-trained) convolutional network.

When a two-branches network is used (such as a Siamese network), two images are provided in input: a template of the target and a search area. To account for large appearance changes and tracking failures, re-detection or validation procedures are included in the model. In actor-critic models, such procedures are implemented by using the critic to score candidate target bounding boxes (as in [101], [102]).

Methods selecting CFs [123], [124] use two different approaches. Reference [124] evaluates a heatmap at a time and can maintain an unlimited number of CFs. The agent decides if the heatmap is reliable or no. On the contrary, [123] takes in input a set of heatmaps of fixed size. This limits the number of CFs to be used to perform tracking but allows the agent to make a decision by jointly considering all available heatmaps.

B. ACTION SPACE AND HISTORY

As shown in Table 2, most of the approaches adopt discrete actions (between 4 and roughly 11). This choice limits the complexity of the model. Works focusing on continuous actions to modify the target bounding box have adopted models trained by DDPG [78] (as in [101], [104]) or its variant TD3 (in [102]) to train a deterministic policy. Works such as [105], [106] use instead the REINFORCE algorithm [76] while [103], [107] adopt A3C [71]. The only method using DQL [22] is [114] where the network is used to also predict the tracking hyper-parameters. An elegant approach is the one proposed in [111] where a probability distribution over the action is trained by NAF [75].

As pointed out in [90], when discrete actions are used to modify iteratively the target bounding box, the agent can be trapped in a local optimum and it starts to select sequentially actions that cancel each other (such as move up and down). Often, more than an initial bounding box/candidate in works such as [91], [92], [114], [115], [126]. Methods in [23], [90], [94], [95], [97], [104], [106], [118], [119], [121] prefer to provide in input to the network the history of the selected actions. This implementation choice has the effect of stabilizing the learning procedure limiting the problem of entering into an action selection cycle.

Methods that do not provide in input the action history, tend to provide in input the past target locations (as in [125]) or to include in the model a recurrent neural network (as in [93], [96], [103], [105], [107], [108]).

It is worth noting that all methods using a discrete set of actions to modify the target bounding box limit the shifts/scaling to a number of pixels computed in proportion to the bounding box size. Hence, the accuracy reachable by these methods is implicitly bounded. The only exception is the BBCorrection algorithm [90] where the box is moved of one pixel along with the vertical/horizontal directions and hence accuracy can be reached at a pixel-level at the cost of an increased number of iterations.

C. DESIGN OF THE REWARD FUNCTION

Often, agents are rewarded with the same metric used to assess the tracking performance, namely by IoU of the tracking results and the ground truth. In works such as [111], [124], where tracking hyper-parameters and model appearance selection are to be decided, the annotated bounding boxes are only indirectly linked with the action meaning.

The last column in Table 2 refers to the reward function adopted in each paper. The reward is a function $f(\cdot)$ of the IoU of the currently estimated bounding box and the ground-truth. In its most general form, the function is defined as follows:

$$f(s_t, a_t, g_t) = \begin{cases} \alpha, & \text{if } IoU(s_t, g_t) > \tau \\ -\alpha, & \text{else} \end{cases} \tag{10}$$

where τ is a threshold (often in the range 0.65 - 0.9) and α is a constant value, often 1. In some works, such as [90], [101], [114], [115], [121], the ΔIoU is used, meaning that the agent is rewarded based on the improvement of the bounding box with respect to that at the previous step (this is indeed used in papers where the bounding box is iteratively estimated). In general, the use of IoU versus ΔIoU should depend on the value function that has to be learned. If a V-function is learned, then we are interested in rewarding the agent based on the final state (and hence IoU should be used). If instead a Q-function is learned, then the agent should be rewarded for taking action a while in state s, and hence the agent should be positively rewarded if the new state is more convenient than the former one (and hence ΔIoU should be used). IoU is a good measure to compare the extent of the bounding box. For the coordinates of the center of the bounding box, or more generally, the coordinates of the points defining the target or the camera parameters required to locate the target, a better measure could be based on the Euclidean distance. This is done in [91], [107]. Absolute difference D is used in [105]. In [119], [120], a function of the improvement of the tracking score is used, while in [106], since the agent has to make decisions about the visibility of the target, the reward depends on the visibility of the target derived from the ground-truth information.

Overall, the reward is in general assuming a value in a discrete set. However, the use of continuous value reward has been investigated in [92], [93], [95], [96], [102], [103], [105], [107], [108], [111], [117], [118], [123], [126].

D. PRE-TRAINING OF THE POLICY

There are a number of works [23], [92]–[95], [101], [102], [104], [106], [111], [124] that overcome convergence issues in DRL by adopting a supervised pre-training of the network, namely by providing the agent with the actions derived from the ground truth.

In some works [115], [117]–[120], [125], despite a supervised pre-training is not directly employed, still, the model takes advantage of trackers trained in a supervised way. In other cases [101], [102], [104], an expert is used to guide the learning process. Somewhat different is the case of [103] where imitation learning is implemented and an external pre-trained tracker is used such that the RL agent learns to perform better than the expert (which may also fail).

Convergence issues might be due to an ineffective policy initialization. It is of interest to note that, in [23], a form of heuristically guided Q-learning is adopted to initialize the policy. In particular, the Q-network is pre-trained in a supervised way by considering not only the reward for the optimal action derived by the ground truth but also assuming that, starting from the next state, the agent will only select optimal actions. This permits the definition of a target policy that, differently from the original DQL approach in [22], is truly optimal.

E. TRAINING STRATEGY

Most of the approaches, especially those adopting DQL, employ a memory buffer to store the agent's past experience. The memory buffer size in experience replay needs to be carefully tuned. With a small memory buffer, there is a high risk of over-fitting on recent data. With a large replay buffer, it is less likely to sample correlated elements; on the other hand, a large buffer can slow training and can hurt the learning process [143].

During learning, mini-batches are sampled from the memory buffer. As pointed out in [101], imbalanced batches can hurt the learning process. In [101], annotations are used to provide more guidance during the learning process.

F. TRACKER INITIALIZATION AND UPDATING

In most of the analyzed approaches, part of the network is devoted to extracting appearance features of the target, while another part is used to implement the policy. In many approaches, fine-tuning is used to adapt the model to the target appearance by using bounding boxes from the first frame. At test time, updates are also required when using CFs.

As shown in Table 2, most of the approaches do not require any updating at test time or special initialization, apart from the bounding box of the first frame. The methods taking advantage of some form of updating at test time are [96], [112], [123], [124] to retrain the CFs, and [90]–[92], [94], [95], [101], [102], [104], [115], [119] to retrain part of the network/policy.

We highlight here an interesting feature of the latter methods. While [90], [91], [115] perform the policy updating and/or fine-tuning on the first frame by using the same DRL algorithm used at training time, the methods in [94], [95], [101], [102], [104] claim to use a supervised learning strategy to update the policy network.

G. ON THE ADOPTED DRL ALGORITHM

In recent years, the evolution of DRL algorithms has been very fast, especially for policy gradient algorithms. Most of the DRL-based tracking algorithms are based on DQL, and only [91] compares DRL approaches (Dueling DQN [144] vs DQN [22]), but no comparison among DQL, AC, and D-PG has been presented yet.

Whenever drifting occurs, re-detection or validation procedures need to be devised. In this sense, actor-critic models are preferable. Indeed, the critic is used to guide the training process but, during tracking, it can also be used to validate the agent's decisions. Often, when the action-value estimated by the critic is below a threshold or negative, a re-detection procedure where multiple target candidates are scored by the critic is used (as in [101], [102]).

In conclusion, from a theoretical point of view, AC methods might be preferable. However, as shown in Table 1, regardless of the actions/goals of the tracking algorithm (reported in the column Decisions of Table 2), the best performing DRL-based tracking algorithms are MT-Exp [124] end DP-Siam [114], both based on DQL. We note that Hierarchical Tracker [96] and DRL-IS [115], based on D-PG and AC, respectively, achieve results comparable to the state-of-the-art.

VIII. CHALLENGES, LIMITATIONS AND FUTURE DIRECTIONS

In this section, we discuss some characteristics of the surveyed papers that may be improved in future work, and that should be considered when designing novel tracking approaches based on DRL.

A. STATE AND ACTION SPACES

In most of the surveyed works aiming at finding the target location, the search area is in general enlarged to facilitate the target detection. In such works, it remains unclear how agents learn to deal with occlusions, and indeed re-detection strategies are often designed to track the target through the occlusion, for instance through particle filtering. Under this point of view, the approach in [105] is interesting in that it processes the whole frame. Hence the agent may have more chances to reacquire the target when the occlusion ends. Instead, the method in [91] is interesting in that it takes in input a search area and a cropped target image. The latter is not a template from the previous frame. Instead, search area and target images are obtained from the same frame. The goal is that of modifying the bounding box used to crop the target image by also considering the contextual information in the search area image. Somehow, this task may take advantage of the target class (i.e. person, car,...), especially if pre-trained networks (such as VGG) are adopted.

Considering that the tracker may need some target appearance information, and has to know where to search the target and which is the current state, we find it surprising that a three-branches network receiving such inputs to train the policy has not been proposed yet. This can be an interesting future direction to investigate.

The most annoying issue in DRL-based tracking approaches aiming at locating the target through a sequence of discrete actions is the action cycling, which happens when actions canceling each other are iteratively selected. A possible research direction might be the design of discrete actions that do not cancel each other. This would limit the cycling issue but not certainly solve it. By following an idea similar to that in [91], it might be possible to devise several policies to select the actions. The policies might be designed to be used in cascade ([91]) or to select multiple actions at once, one from each policy. In both cases, if each policy can model a subset of canceling actions (such as move up, down), then the cycling issue can be limited. Moreover, a lower number of iterations may be required to locate the target.

The existence of the cycling issue suggests that the agent may benefit from knowing what attempts it has already made to correctly learn what decision to make. And indeed, several approaches include the action history in the network input. It is possible to speculate that knowing the progress associated with the past attempts may help the agent to select optimal new actions. However, at the test time, the reward is unavailable and more investigations to provide such kind of feedback to the agent are suggested. While the above argument is easy to understand when the agent decides how to iteratively modify the bounding box, it becomes cumbersome to understanding what form should have the action history in works where the agent deals with a selection problem (such as selecting the appearance model to use) or any other decision that changes the tracker state (as the choice to update or not the appearance model).

B. REWARD FUNCTION

The main limitation of functions like that in Eq. 10 is that there is no difference in the reward of actions yielding to high IoU. In other words, the agent has settled for bounding boxes whose IoU with the ground-truth is higher than the threshold τ even if it could get more accurate ones. A continuous value reward could address this issue. However, especially at the beginning of the training, the agent has to be strongly penalized for selecting not promising actions. A reward function similar to that in [123] and that can be easily adapted to use ΔIoU may suffice:

$$f(s_t, a_t, g_t) = \begin{cases} IoU(s_t, g_t) + \alpha, & \text{if } IoU(s_t, g_t) > \tau \\ -\alpha, & \text{else.} \end{cases}$$

In theory, it might be possible to adopt a behavior shaping strategy [21] in which, while the agent learns to take more and more convenient actions, the reward function changes and the agent attempts to solve problems of increasing difficulty. However, in [68], the use of behavioral shaping for tabular Q-functions is discouraged due to the risk of getting the agent trapped in a local optimum from which it cannot escape by varying the reward function. Certainly, this is another issue that should be investigated in DRL.

The approaches considered in this paper define the reward functions by using ground-truth information. There might be other correlated information to be used to define the reward function in methods where actions represent changes to the target bounding box. High-level annotations, such as verbal descriptions of the target [145], or IRL might be exploited to define new reward functions; ideally, it might be possible to learn off-line and independently a function that estimates the IoU in a supervised way to be used within the DRL framework. All these strategies are worthy of future research.

C. INITIALIZATION AND TRAINING OF THE MODEL

In practice, most of the surveyed approaches complain that convergence is difficult to reach without supervised learning. We actually suspect that the problem is mainly ascribable to the policy network parameter initialization. To understand this problem, we focus on DQL approaches whilst the problem arises also in the other DRL frameworks. In DQL, the state is given in input to the Q-network. During learning, it is generally adopted an ϵ-greedy policy, namely with a probability ϵ the action is chosen uniformly at random, otherwise, the action is the one providing the highest Q-value. In practice, in the latter case, the agent trusts in its former experience. At the beginning of the training procedure, however, no experience has been accumulated and the training may proceed in a direction that does not bring to the optimal solution since some non-optimal action-state pairs would seem preferable to others due to the random initialization of the policy. Even if ϵ is initially set to 1 and annealed over time, this may not be enough to cancel the initial bias of the policy network. This problem has also been pointed out in [68] for tabular Q-learning where it is suggested to initialize the Q-function uniformly to a value that is higher than the Q-value achieved if the reward would be constant whatever the action selected by the agent is. Such kind of initialization guarantees that the policy is not initially biased towards some subset of action-state pairs. To the best of our knowledge, this initialization issue has been overlooked in DRL-based visual tracking approaches. It is so far unclear what is the effect of initializing a policy network to a constant value considering that such functions are highly non-linear, and we believe this is another research direction to investigate in the future.

A problem in RL is that of obtaining batches of uncorrelated samples. In our understanding, this problem has been always approached by sampling mini-batches from the memory buffer. However, as pointed out in [101], imbalanced batches can hurt the learning process. In [101], annotations are used to provide more guidance during the learning process. We believe that other solutions are possible. For instance, as done in classification, the learning might improve by sampling mini-batches where the number of samples with positive and negative rewards is balanced. Boosting strategies may also be devised to push the agent towards the most difficult states. Such an idea is similar to the Prioritized Experience replay [146] where samples from the experience replay buffer are drawn based on the TD error.

D. DRL ALGORITHMS

In other benchmarks, such as the Atari games, different DRL approaches have been compared and new algorithms proposed. For instance, in [147] it has been shown that the integration of Double DQL, Dueling DQN, N-Step learning, Prioritized Experience Replay, Distributional RL, Noisy Network yields to an improved algorithm named Rainbow. The Rainbow algorithm outperforms the other DQL algorithms also in terms of required training time. The adoption of Rainbow in visual tracking is an interesting topic of future investigations.

On the other hand, it has been shown that Q-learning methods implement policy gradient updates in entropy-regularized RL, which makes the differences between DQL and D-PG methods thinner [148]. In [149], [150], it is reported that AC has certain convergence properties compared to QL but it suffers from a high variance of the policy estimators. However, AC approaches are prone to instability, due to the interaction between actor and critic during learning [151] such that regularization terms are required.

IX. CONCLUSION

This paper presents the current state-of-the-art in DRL-based visual tracking. First, an overview of the visual tracking problem and the main deep tracking algorithms is presented. Then, possible ways to model visual tracking within the RL framework have been presented. RL and its evolution into DRL are summarized.

This paper categorizes DRL-based tracking approaches based on the main pursued goal. Some methods focus on the problem of estimating the target bounding box on the image plane. Other approaches focus on the tracking strategy, namely the prediction of the best tracking hyper-parameters, the search area where to locate the target, or the tracker transition from a state to another (such as re-initialize the tracker or update it). Finally, a third group includes methods aiming at modeling the target appearance selection/update process. Within each group, some approaches require model parameters adaption at test time, while others use pre-trained models. This paper further categorizes DRL-based tracking approaches based on the adopted DRL framework, that is DQL, D-PG, and AC.

DRL has been applied to the tracking problem in different ways, demonstrating its versatility. The state is generally represented by an image: a heatmap or an image cropped around the last predicted target location. The definition of the actions largely changes depending on the application. When DRL is used to determine the target bounding box through discrete actions, actions represent shifts or scaling factors to adjust the bounding box in a vertical/horizontal direction. At each frame, the policy is interrogated several times until a terminal action is selected. In this case, the next frame is processed. Continuous actions are used to regress the bounding box parameters, and the policy is interrogated once per frame.

Analysis of the state-of-the-art shows that most of the DRL-based approaches achieve state-of-the-art performance. Apparently, there is not a neural architecture to be preferred over the others. Training/pre-training strategies widely differ from paper-to-paper. Pre-training the models in a supervised way on actions derived from the ground truth seems to accelerate the learning.

Overall, the application of DRL to visual tracking is exciting because it allows the system to learn by a trial-and-error strategy, and it is interesting to see how much a system can learn from long interactions with the environment. However, there are still issues to study and solve. For example, the problems of initializing the policy effectively, of designing a reward function, and a proper set of actions are still open. Furthermore, while continuous actions might be the appropriate choice in visual tracking, still there are limited training strategies that can enable the learning of proper policies. All these issues open up to novel and exciting research directions.

REFERENCES

[1] Y. Bengio, A. Courville, and P. Vincent, "Representation learning: A review and new perspectives," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 35, no. 8, pp. 1798–1828, Aug. 2013.

[2] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, May 2015.

[3] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 779–788.

[4] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "SSD: Single shot MultiBox detector," in *Proc. Eur. Conf. Comput. Vis.* Cham, Switzerland: Springer, 2016, pp. 21–37.

[5] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask R-CNN," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, Oct. 2017, pp. 2961–2969.

[6] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*, vol. 521, no. 7553. Cambridge, MA, USA: MIT Press, 2016, p. 800.

[7] D. Jakubovitz, R. Giryes, and M. R. Rodrigues, "Generalization error in deep learning," in *Compressed Sensing and its Applications*. Cham, Switzerland: Springer, 2019, pp. 153–193.

[8] E. López-Rubio, "Computational functionalism for the deep learning era," *Minds Mach.*, vol. 28, no. 4, pp. 667–688, Dec. 2018.

[9] J. C. Caicedo and S. Lazebnik, "Active object localization with deep reinforcement learning," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, Dec. 2015, pp. 2488–2496.

[10] D. Held, S. Thrun, and S. Savarese, "Learning to track at 100 FPS with deep regression networks," in *Proc. Eur. Conf. Comput. Vis.* Cham, Switzerland: Springer, 2016, pp. 749–765.

[11] A. Toshev and C. Szegedy, "DeepPose: Human pose estimation via deep neural networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2014, pp. 1653–1660.

[12] P. Li, D. Wang, L. Wang, and H. Lu, "Deep visual tracking: Review and experimental comparison," *Pattern Recognit.*, vol. 76, pp. 323–338, Apr. 2018.

[13] M. Fiaz, A. Mahmood, S. Javed, and S. K. Jung, "Handcrafted and deep trackers: Recent visual object tracking approaches and trends," *ACM Comput. Surv.*, vol. 52, no. 2, pp. 43:1–43:44, 2019.

[14] A. Yilmaz, O. Javed, and M. Shah, "Object tracking: A survey," *ACM Comput. Surv.*, vol. 38, no. 4, p. 13, 2006.

[15] X. Li, W. Hu, C. Shen, Z. Zhang, A. Dick, and A. Van Den Hengel, "A survey of appearance models in visual object tracking," *ACM Trans. Intell. Syst. Technol.*, vol. 4, no. 4, p. 58, Sep. 2013.

[16] H. Nam and B. Han, "Learning multi-domain convolutional neural networks for visual tracking," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 4293–4302.

[17] Z. Zhang and H. Peng, "Deeper and wider Siamese networks for real-time visual tracking," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2019, pp. 4591–4600.

[18] R. Tao, E. Gavves, and A. W. M. Smeulders, "Siamese instance search for tracking," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 1420–1429.

[19] M. Cen and C. Jung, "Fully convolutional Siamese fusion networks for object tracking," in *Proc. IEEE Int. Conf. Image Process. (ICIP)*, Oct. 2018, pp. 3718–3722, doi: 10.1109/ICIP.2018.8451102.

[20] K. Chen and W. Tao, "Once for all: A two-flow convolutional neural network for visual tracking," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 28, no. 12, pp. 3377–3386, Dec. 2017.

[21] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. Cambridge, MA, USA: MIT Press, 2018.

[22] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," 2013, *arXiv:1312.5602*. [Online]. Available: http://arxiv.org/abs/1312.5602

[23] J. Supancic, III, and D. Ramanan, "Tracking as online decision-making: Learning a policy from streaming videos with reinforcement learning," in *Proc. Int. Conf. Comput. Vis.*, 2017, pp. 322–331.

[24] W. Wang, Y. Huang, and L. Wang, "Language-driven temporal activity localization: A semantic matching reinforcement learning model," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2019, pp. 334–343.

[25] W. Wu, D. He, X. Tan, S. Chen, and S. Wen, "Multi-agent reinforcement learning based frame sampling for effective untrimmed video recognition," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, Oct. 2019, pp. 6222–6231.

[26] J. Han, L. Yang, D. Zhang, X. Chang, and X. Liang, "Reinforcement cutting-agent learning for video object segmentation," in *Proc. Comput. Vis. Pattern Recognit.*, 2018, pp. 9080–9089.

[27] R. E. Kalman, "A new approach to linear filtering and prediction problems," *J. Basic Eng.*, vol. 82, no. 1, pp. 35–45, Mar. 1960.

[28] L. Ljung, "Asymptotic behavior of the extended Kalman filter as a parameter estimator for linear systems," *IEEE Trans. Autom. Control*, vol. AC-24, no. 1, pp. 36–50, Feb. 1979.

[29] E. A. Wan and R. Van Der Merwe, "The unscented Kalman filter for nonlinear estimation," in *Proc. IEEE Adapt. Syst. Signal Process., Commun., Control Symp.*, Oct. 2000, pp. 153–158.

[30] N. J. Gordon, D. J. Salmond, and A. F. M. Smith, "Novel approach to nonlinear/non-Gaussian Bayesian state estimation," *IEE Proc. F, Radar Signal Process.*, vol. 140, no. 2, pp. 107–113, Apr. 1993.

[31] A. Blake, R. Curwen, and A. Zisserman, "A framework for spatiotemporal control in the tracking of visual contours," *Int. J. Comput. Vis.*, vol. 11, no. 2, pp. 127–145, Oct. 1993.

[32] P. Li, T. Zhang, and B. Ma, "Unscented Kalman filter for visual curve tracking," *Image Vis. Comput.*, vol. 22, no. 2, pp. 157–164, Feb. 2004.

[33] J. Lou, H. Yang, W. M. Hu, and T. Tan, "Visual vehicle tracking using an improved EKF," in *Proc. Asian Conf. Comput. Vis.*, 2002, pp. 296–301.

[34] M. Isard and A. Blake, "Condensation-conditional density propagation for visual tracking," *Int. J. Comput. Vis.*, vol. 29, no. 1, pp. 5–28, Nov. 1998.

[35] B. Babenko, M.-H. Yang, and S. Belongie, "Robust object tracking with online multiple instance learning," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 33, no. 8, pp. 1619–1632, Aug. 2010.

[36] Z. Kalal, K. Mikolajczyk, and J. Matas, "Tracking-learning-detection," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 34, no. 7, pp. 1409–1422, Jul. 2011.

[37] D. S. Bolme, B. A. Draper, and J. R. Beveridge, "Average of synthetic exact filters," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2009, pp. 2105–2112.

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

[39] M. Danelljan, G. Hager, F. S. Khan, and M. Felsberg, "Learning spatially regularized correlation filters for visual tracking," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, Dec. 2015, pp. 4310–4318.

[40] Y. LeCun, B. E. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. E. Hubbard, and L. D. Jackel, "Handwritten digit recognition with a back-propagation network," in *Proc. Adv. Neural Inf. Process. Syst.*, 1990, pp. 396–404.

[41] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2015, pp. 3431–3440.

[42] D. H. Ballard, "Modular learning in neural networks," in *Proc. AAAI*, 1987, pp. 279–284.

[43] S. Chopra, R. Hadsell, and Y. LeCun, "Learning a similarity metric discriminatively, with application to face verification," in *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2005, pp. 539–546.

[44] J. J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities," *Proc. Nat. Acad. Sci. USA*, vol. 79, no. 8, pp. 2554–2558, 1982.

[45] Z. C. Lipton, J. Berkowitz, and C. Elkan, "A critical review of recurrent neural networks for sequence learning," 2015, *arXiv:1506.00019*. [Online]. Available: http://arxiv.org/abs/1506.00019

[46] W. Choi and S. Savarese, "A unified framework for multi-target tracking and collective activity recognition," in *Eur. Conf. Comput. Vis.* Berlin, Germany: Springer, 2012, pp. 215–230.

[47] J. Schmidhuber, J. Zhao, and M. Wiering, "Simple principles of metalearning," *Tech. Rep. IDSIA*, vol. 69, pp. 1–23, Jun. 1996.

[48] S. Hochreiter, A. S. Younger, and P. R. Conwell, "Learning to learn using gradient descent," in *Proc. Int. Conf. Artif. Neural Netw.* Berlin, Germany: Springer, 2001, pp. 87–94.

[49] C. Finn, P. Abbeel, and S. Levine, "Model-agnostic meta-learning for fast adaptation of deep networks," 2017, *arXiv:1703.03400*. [Online]. Available: http://arxiv.org/abs/1703.03400

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

[51] H. Li, Y. Li, and F. Porikli, "DeepTrack: Learning discriminative feature representations online for robust visual tracking," *IEEE Trans. Image Process.*, vol. 25, no. 4, pp. 1834–1848, Apr. 2015.

[52] I. Jung, J. Son, M. Baek, and B. Han, "Real-time MDNet," in *Proc. Eur. Conf. Comput. Vis. (ECCV)*, 2018, pp. 83–98.

[53] M. Danelljan, G. Hager, F. S. Khan, and M. Felsberg, "Convolutional features for correlation filter based visual tracking," in *Proc. IEEE Int. Conf. Comput. Vis. Workshop (ICCVW)*, Dec. 2015, pp. 58–66.

[54] C. Ma, J.-B. Huang, X. Yang, and M.-H. Yang, "Hierarchical convolutional features for visual tracking," in *Proc. Int. Conf. Comput. Vis.*, 2015, pp. 3074–3082.

[55] M. Danelljan, G. Bhat, F. S. Khan, and M. Felsberg, "ATOM: Accurate tracking by overlap maximization," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2019, pp. 4660–4669.

[56] Y. Song, C. Ma, X. Wu, L. Gong, L. Bao, W. Zuo, C. Shen, R. W. H. Lau, and M.-H. Yang, "VITAL: VIsual tracking via adversarial learning," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 8990–8999.

[57] L. Bertinetto, J. F. Henriques, J. Valmadre, P. Torr, and A. Vedaldi, "Learning feed-forward one-shot learners," in *Proc. Adv. Neural Inf. Process. Syst.*, 2016, pp. 523–531.

[58] J. Choi, J. Kwon, and K. M. Lee, "Deep meta learning for real-time target-aware visual tracking," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, Oct. 2019, pp. 911–920.

[59] E. Park and A. C. Berg, "Meta-tracker: Fast and robust online adaptation for visual object trackers," in *Proc. Eur. Conf. Comput. Vis. (ECCV)*, 2018, pp. 569–585.

[60] G. Wang, C. Luo, X. Sun, Z. Xiong, and W. Zeng, "Tracking by instance detection: A meta-learning approach," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2020, pp. 6288–6297.

[61] G. Bhat, M. Danelljan, L. Van Gool, and R. Timofte, "Learning discriminative model prediction for tracking," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, Oct. 2019, pp. 6182–6191.

[62] M. Danelljan, L. Van Gool, and R. Timofte, "Probabilistic regression for visual tracking," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2020, pp. 7183–7192.

[63] B. Li, J. Yan, W. Wu, Z. Zhu, and X. Hu, "High performance visual tracking with Siamese region proposal network," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 8971–8980.

[64] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards real-time object detection with region proposal networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2015, pp. 91–99.

[65] B. Li, W. Wu, Q. Wang, F. Zhang, J. Xing, and J. Yan, "SiamRPN++: Evolution of Siamese visual tracking with very deep networks," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2019, pp. 4282–4291.

[66] T. Yang and A. B. Chan, "Recurrent filter learning for visual tracking," in *Proc. IEEE Int. Conf. Comput. Vis. Workshops (ICCVW)*, Oct. 2017, pp. 2010–2019.

[67] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: A survey," *J. Artif. Intell. Res.*, vol. 4, no. 1, pp. 237–285, Jan. 1996.

[68] L. Matignon, G. J. Laurent, and N. L. Fort-Piat, "Reward function and initial values: Better choices for accelerated goal-directed reinforcement learning," in *Proc. Int. Conf. Artif. Neural Netw.* Berlin, Germany: Springer, 2006, pp. 840–849.

[69] C. J. C. H. Watkins and P. Dayan, "Q-learning," *Mach. Learn.*, vol. 8, nos. 3–4, pp. 279–292, 1992.

[70] M. Riedmiller, "Neural fitted Q iteration–first experiences with a data efficient neural reinforcement learning method," in *Proc. Eur. Conf. Mach. Learn.* Berlin, Germany: Springer, 2005, pp. 317–328.

[71] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *Proc. Int. Conf. Mach. Learn.*, 2016, pp. 1928–1937.

[72] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, and S. Petersen, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, p. 529, 2015.

[73] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double Q-learning," in *Proc. Conf. Artif. Intell.*, 2016, pp. 1–7.

[74] Z. Wang, T. Schaul, M. Hessel, H. van Hasselt, M. Lanctot, and N. de Freitas, "Dueling network architectures for deep reinforcement learning," 2015, *arXiv:1511.06581*. [Online]. Available: http://arxiv.org/abs/1511.06581

[75] S. Gu, T. Lillicrap, I. Sutskever, and S. Levine, "Continuous deep Q-learning with model-based acceleration," in *Proc. Int. Conf. Mach. Learn.*, 2016, pp. 2829–2838.

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

[77] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, "Deterministic policy gradient algorithms," in *Proc. Int. Conf. Mach. Learn.*, 2014, pp. 387–395.

[78] 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," 2015, *arXiv:1509.02971*. [Online]. Available: http://arxiv.org/abs/1509.02971

[79] S. Fujimoto, H. van Hoof, and D. Meger, "Addressing function approximation error in actor-critic methods," 2018, *arXiv:1802.09477*. [Online]. Available: http://arxiv.org/abs/1802.09477

[80] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *Proc. Int. Conf. Mach. Learn.*, 2015, pp. 1889–1897.

[81] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017, *arXiv:1707.06347*. [Online]. Available: http://arxiv.org/abs/1707.06347

[82] F. Liu and J. Su, "Reinforcement learning-based feature learning for object tracking," in *Proc. 17th Int. Conf. Pattern Recognit. (ICPR)*, vol. 2, 2004, pp. 748–751.

[83] Q. Guo, R. Han, W. Feng, Z. Chen, and L. Wan, "Selective spatial regularization by reinforcement learned decision making for object tracking," *IEEE Trans. Image Process.*, vol. 29, pp. 2999–3013, 2020.

[84] A. D. Bagdanov, A. del Bimbo, W. Nunziati, and F. Pernici, "A reinforcement learning approach to active camera foveation," in *Proc. 4th ACM Int. Workshop Video Surveill. Sensor Netw. (VSSN)*, 2006, pp. 179–186.

[85] A. Cohen and V. Pavlovic, "Reinforcement learning for robust and efficient real-world tracking," in *Proc. 20th Int. Conf. Pattern Recognit.*, Aug. 2010, pp. 2989–2992.

[86] K. Meshgi, M. S. Mirzaei, and S. Oba, "Long and short memory balancing in visual co-tracking using Q-learning," 2019, *arXiv:1902.05211*. [Online]. Available: http://arxiv.org/abs/1902.05211

[87] S. Khim, S. Hong, Y. Kim, and P. K. Rhee, "Adaptive visual tracking using the prioritized Q-learning algorithm: MDP-based parameter learning approach," *Image Vis. Comput.*, vol. 32, no. 12, pp. 1090–1101, Dec. 2014.

[88] Y. Xiang, A. Alahi, and S. Savarese, "Learning to track: Online multi-object tracking by decision making," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, Dec. 2015, pp. 4705–4713.

[89] C. Wu, H. Sun, H. Wang, K. Fu, G. Xu, W. Zhang, and X. Sun, "Online multi-object tracking via combining discriminative correlation filters with making decision," *IEEE Access*, vol. 6, pp. 43499–43512, 2018.

[90] Y. Jiang, H. Shin, and H. Ko, "Precise regression for bounding box correction for improved tracking based on deep reinforcement learning," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, Apr. 2018, pp. 1643–1647.

[91] Y. Jiang, D. K. Han, and H. Ko, "Relay dueling network for visual tracking with broad field-of-view," *IET Comput. Vis.*, vol. 13, no. 7, pp. 615–622, Oct. 2019.

[92] Z. Teng, B. Zhang, and J. Fan, "Three-step action search networks with deep Q-learning for real-time object tracking," *Pattern Recognit.*, vol. 101, May 2020, Art. no. 107188.

[93] M. Guo, J. Lu, and J. Zhou, "Dual-agent deep reinforcement learning for deformable face tracking," in *Proc. Eur. Conf. Comput. Vis. (ECCV)*, 2018, pp. 768–783.

[94] S. Yun, J. Choi, Y. Yoo, K. Yun, and J. Y. Choi, "Action-decision networks for visual tracking with deep reinforcement learning," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 2711–2720.

[95] D. Huang, L. Kong, J. Zhu, and L. Zheng, "Improved action-decision network for visual tracking with meta-learning," *IEEE Access*, vol. 7, pp. 117206–117218, 2019.

[96] B. Zhong, B. Bai, J. Li, Y. Zhang, and Y. Fu, "Hierarchical tracking by reinforcement learning-based searching and coarse-to-fine verifying," *IEEE Trans. Image Process.*, vol. 28, no. 5, pp. 2331–2341, May 2018.

[97] M.-X. Jiang, C. Deng, Z.-G. Pan, L.-F. Wang, and X. Sun, "Multiobject tracking in videos based on LSTM and deep reinforcement learning," *Complexity*, vol. 2018, pp. 1–12, Nov. 2018.

[98] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2014, *arXiv:1409.1556*. [Online]. Available: http://arxiv.org/abs/1409.1556

[99] A. Newell, K. Yang, and J. Deng, "Stacked hourglass networks for human pose estimation," in *Proc. Eur. Conf. Comput. Vis.* Cham, Switzerland: Springer, 2016, pp. 483–499.

[100] S. Yun, J. Choi, Y. Yoo, K. Yun, and J. Y. Choi, "Action-driven visual object tracking with deep reinforcement learning," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 29, no. 6, pp. 2239–2252, Jun. 2018.

[101] B. Chen, D. Wang, P. Li, S. Wang, and H. Lu, "Real-time 'actor-critic' tracking," in *Proc. ECCV*, 2018, pp. 318–334.

[102] S. Zheng and H. Wang, "Real-time visual object tracking based on reinforcement learning with twin delayed deep deterministic algorithm," in *Proc. Int. Conf. Intell. Sci. Big Data Eng.* Cham, Switzerland: Springer, 2019, pp. 165–177.

[103] M. Dunnhofer, N. Martinel, G. L. Foresti, and C. Micheloni, "Visual tracking by means of deep reinforcement learning and an expert demonstrator," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. Workshop (ICCVW)*, Oct. 2019, pp. 2290–2299.

[104] Q. Wang, L. Zhuang, N. Wang, W. Zhou, and H. Li, "Learning motion-aware policies for robust visual tracking," in *Proc. IEEE Int. Conf. Multimedia Expo (ICME)*, Jul. 2019, pp. 1786–1791.

[105] D. Zhang, H. Maei, X. Wang, and Y.-F. Wang, "Deep reinforcement learning for visual object tracking in videos," 2017, *arXiv:1701.08936*. [Online]. Available: http://arxiv.org/abs/1701.08936

[106] X. Liu, Q. Xu, T. Chau, Y. Mu, L. Zhu, and S. Yan, "Revisiting jump-diffusion process for visual tracking: A reinforcement learning approach," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 29, no. 8, pp. 2431–2441, Aug. 2019.

[107] W. Luo, P. Sun, F. Zhong, W. Liu, T. Zhang, and Y. Wang, "End-to-end active object tracking and its real-world deployment via reinforcement learning," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 42, no. 6, pp. 1317–1332, Jun. 2020.

[108] F. Zhong, P. Sun, W. Luo, T. Yan, and Y. Wang, "AD-VAT: An asymmetric dueling mechanism for learning visual active tracking," in *Proc. Int. Conf. Learn. Represent.*, 2018, pp. 1–16.

[109] W. Luo, P. Sun, F. Zhong, W. Liu, T. Zhang, and Y. Wang, "End-to-end active object tracking via reinforcement learning," in *Proc. Int. Conf. Mach. Learn.*, 2018, pp. 1–10.

[110] F. Zhong, P. Sun, W. Luo, T. Yan, and Y. Wang, "AD-VAT+: An asymmetric dueling mechanism for learning and understanding visual active tracking," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 43, no. 5, pp. 1467–1482, May 2021.

[111] X. Dong, J. Shen, W. Wang, Y. Liu, L. Shao, and F. Porikli, "Hyperparameter optimization for tracking with continuous deep Q-learning," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 518–527.

[112] X. Dong, J. Shen, W. Wang, L. Shao, H. Ling, and F. Porikli, "Dynamical hyperparameter optimization via deep reinforcement learning in tracking," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 43, no. 5, pp. 1515–1529, May 2021.

[113] H. K. Galoogahi, A. Fagg, and S. Lucey, "Learning background-aware correlation filters for visual tracking," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, Oct. 2017, pp. 1135–1143.

[114] M. H. Abdelpakey and M. S. Shehata, "DP-Siam: Dynamic policy Siamese network for robust object tracking," *IEEE Trans. Image Process.*, vol. 29, pp. 1479–1492, 2019.

[115] L. Ren, X. Yuan, J. Lu, M. Yang, and J. Zhou, "Deep reinforcement learning with iterative shift for visual tracking," in *Proc. Eur. Conf. Comput. Vis.*, 2018, pp. 684–700.

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

[117] Z. Zhong, Z. Yang, W. Feng, W. Wu, Y. Hu, and C.-L. Liu, "Decision controller for object tracking with deep reinforcement learning," *IEEE Access*, vol. 7, pp. 28069–28079, 2019.

[118] K. Song, W. Zhang, W. Lu, Z.-J. Zha, X. Ji, and Y. Li, "Visual object tracking via guessing and matching," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 30, no. 11, pp. 4182–4191, Nov. 2020.

[119] W. Song, S. Li, T. Chang, A. Hao, Q. Zhao, and H. Qin, ''Cross-view contextual relation transferred network for unsupervised vehicle tracking in drone videos,'' in *Proc. IEEE Winter Conf. Appl. Comput. Vis. (WACV)*, Mar. 2020, pp. 1707–1716.

[120] X. Wang, C. Li, B. Luo, and J. Tang, ''SINT++: Robust visual tracking via adversarial positive instance generation,'' in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 4864–4873.

[121] C. Huang, S. Lucey, and D. Ramanan, ''Learning policies for adaptive tracking with deep feature cascades,'' in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, Oct. 2017, pp. 105–114.

[122] Y. Liu, X. Dong, X. Lu, F. S. Khan, J. Shen, and S. Hoi, ''Teacher-students knowledge distillation for Siamese trackers,'' 2019, *arXiv:1907.10586*. [Online]. Available: http://arxiv.org/abs/1907.10586

[123] Y. Xie, J. Xiao, K. Huang, J. Thiyagalingam, and Y. Zhao, ''Correlation filter selection for visual tracking using reinforcement learning,'' 2018, *arXiv:1811.03196*. [Online]. Available: http://arxiv.org/abs/1811.03196

[124] W. Huang, Y. Wu, and Y. Jia, ''Tracker-level decision by deep reinforcement learning for robust visual tracking,'' in *Proc. Int. Conf. Image Graph.* Cham, Switzerland: Springer, 2019, pp. 442–453.

[125] J. Choi, J. Kwon, and K. M. Lee, ''Real-time visual tracking by deep reinforced decision making,'' *Comput. Vis. Image Understand.*, vol. 171, pp. 10–19, Jun. 2018.

[126] M. Sun, J. Xiao, E. G. Lim, B. Zhang, and Y. Zhao, ''Fast template matching and update for video object tracking and segmentation,'' in *Proc. CVPR*, 2020, pp. 10791–10799.

[127] D. Bolya, C. Zhou, F. Xiao, and Y. J. Lee, ''YOLACT: Real-time instance segmentation,'' in *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, Oct. 2019, pp. 9157–9166.

[128] T. Xu, Z.-H. Feng, X.-J. Wu, and J. Kittler, ''Joint group feature selection and discriminative filter learning for robust visual object tracking,'' in *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, Oct. 2019, pp. 7950–7960.

[129] H. Fan and H. Ling, ''Siamese cascaded region proposal networks for real-time visual tracking,'' in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2019, pp. 7952–7961.

[130] S. Tian, X. Liu, M. Liu, S. Li, and B. Yin, ''Siamese tracking network with informative enhanced loss,'' *IEEE Trans. Multimedia*, vol. 23, pp. 120–132, 2021.

[131] M. Danelljan, G. Bhat, F. S. Khan, and M. Felsberg, ''ECO: Efficient convolution operators for tracking,'' in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 6638–6646.

[132] Y. Wu, J. Lim, and M.-H. Yang, ''Online object tracking: A benchmark,'' in *Proc. Comput. Vis. Pattern Recognit.*, 2013, pp. 2411–2418.

[133] M. Kristan, J. Matas, A. Leonardis, T. Vojir, R. Pflugfelder, G. Fernandez, G. Nebehay, F. Porikli, and L. Čehovin, ''A novel performance evaluation methodology for single-target trackers,'' *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 38, no. 11, pp. 2137–2155, Nov. 2016.

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

[135] H. Fan, H. Ling, L. Lin, F. Yang, P. Chu, G. Deng, S. Yu, H. Bai, Y. Xu, and C. Liao, ''LaSOT: A high-quality benchmark for large-scale single object tracking,'' in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2019, pp. 5374–5383.

[136] M. Muller, A. Bibi, S. Giancola, S. Alsubaihi, and B. Ghanem, ''TrackingNet: A large-scale dataset and benchmark for object tracking in the wild,'' in *Proc. Eur. Conf. Comput. Vis. (ECCV)*, 2018, pp. 300–317.

[137] P. Liang, E. Blasch, and H. Ling, ''Encoding color information for visual tracking: Algorithms and benchmark,'' *IEEE Trans. Image Process.*, vol. 24, no. 12, pp. 5630–5644, Dec. 2015.

[138] L. Huang, X. Zhao, and K. Huang, ''GOT-10k: A large high-diversity benchmark for generic object tracking in the wild,'' *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 43, no. 5, pp. 1562–1577, May 2021.

[139] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, and A. C. Berg, ''ImageNet large scale visual recognition challenge,'' *Int. J. Comput. Vis.*, vol. 115, no. 3, pp. 211–252, Dec. 2015.

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

[141] L. Čehovin, A. Leonardis, and M. Kristan, ''Visual object tracking performance measures revisited,'' *IEEE Trans. Image Process.*, vol. 25, no. 3, pp. 1261–1274, Mar. 2016.

[142] M. Jiang, T. Hai, Z. Pan, H. Wang, Y. Jia, and C. Deng, ''Multi-agent deep reinforcement learning for multi-object tracker,'' *IEEE Access*, vol. 7, pp. 32400–32407, 2019.

[143] S. Zhang and R. S. Sutton, ''A deeper look at experience replay,'' 2017, *arXiv:1712.01275*. [Online]. Available: http://arxiv.org/abs/1712.01275

[144] Z. Wang, T. Schaul, M. Hessel, H. Hasselt, M. Lanctot, and N. Freitas, ''Dueling network architectures for deep reinforcement learning,'' in *Proc. Int. Conf. Mach. Learn.*, 2016, pp. 1995–2003.

[145] Q. Feng, V. Ablavsky, Q. Bai, G. Li, and S. Sclaroff, ''Real-time visual object tracking with natural language description,'' in *Proc. IEEE Winter Conf. Appl. Comput. Vis. (WACV)*, Mar. 2020, pp. 700–709.

[146] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, ''Prioritized experience replay,'' 2015, *arXiv:1511.05952*. [Online]. Available: http://arxiv.org/abs/1511.05952

[147] M. Hessel, J. Modayil, H. Van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M. Azar, and D. Silver, ''Rainbow: Combining improvements in deep reinforcement learning,'' in *Proc. 32nd Conf. Artif. Intell. (AAAI)*, 2018, pp. 3215–3222.

[148] J. Schulman, X. Chen, and P. Abbeel, ''Equivalence between policy gradients and soft Q-learning,'' 2017, *arXiv:1704.06440*. [Online]. Available: http://arxiv.org/abs/1704.06440

[149] V. R. Konda and J. N. Tsitsiklis, ''Actor-critic algorithms,'' in *Proc. Adv. Neural Inf. Process. Syst.*, 2000, pp. 1008–1014.

[150] S. Bhatnagar, M. Ghavamzadeh, M. Lee, and R. S. Sutton, ''Incremental natural actor-critic algorithms,'' in *Proc. Adv. Neural Inf. Process. Syst.*, 2008, pp. 105–112.

[151] S. Parisi, V. Tangkaratt, J. Peters, and M. E. Khan, ''TD-regularized actor-critic methods,'' *Mach. Learn.*, vol. 108, nos. 8–9, pp. 1467–1501, Sep. 2019.

GIORGIO CRUCIATA is currently pursuing the Ph.D. degree in innovation technology with the University of Palermo, Italy. He belongs to the CVIP Research Group. His main research interest includes deep reinforcement learning applied to computer vision problems.

LILIANA LO PRESTI is currently an Assistant Professor with the University of Palermo. She teaches programming and algorithms and data structures in engineering courses. She belongs to the CVIP Research Group. Her main research interests include machine learning techniques applied to computer vision problems, especially visual tracking and action recognition.

MARCO LA CASCIA is currently a Full Professor with the University of Palermo, where he is the Director of the CVIP Research Group. He teaches programming and web programming to students in computer engineering. His main research interests include computer vision and multimedia processing, especially tracking, action recognition, and content-based image/video retrieval.