UNIVERSITÀ DEGLI STUDI DI PALERMO

Dottorato in Ingegneria dell'Innovazione Tecnologica

Dipartimento di Ingegneria

Deep learning techniques for visual object tracking

PhD Candidate
GIORGIO CRUCIATA

Tutor
Prof. LILIANA LO PRESTI

Chair of the Doctoral School
Prof. SALVATORE GAGLIO

CICLO XXXV

ANNO DI CONSEGUIMENTO 2023

Desidero ringraziare tutte le persone che mi hanno sostenuto e incoraggiato lungo il percorso del mio dottorato. Senza il loro supporto, la realizzazione di questo traguardo sarebbe stata molto più difficile. In primo luogo, vorrei ringraziare la Professoressa Liliana Lo Presti per la sua guida e il suo sostegno costante. Grazie per aver creduto nelle mie capacità e per avermi spinto a superare i miei limiti. La sua fiducia in me è stata un motore di ispirazione per affrontare le sfide di qualsiasi tipo in particolare sfide accademiche, di Erasmus e di tirocinio. Un ringraziamento speciale va alla mia famiglia tutta, per essere stati al mio fianco in questo percorso. Le vostre parole di incoraggiamento, il vostro sostegno incondizionato e la vostra presenza costante sono stati fondamentali per superare le difficoltà e per ricordarmi che non sono mai solo. Voglio esprimere la mia gratitudine alla mia ragazza Silvia per aver creduto in me quando io stesso avevo dei dubbi, per essermi stata vicina durante i momenti di stress e per avermi sostenuto in ogni decisione presa lungo questo percorso. La tua presenza amorevole ha reso il cammino del mio dottorato un'esperienza ancora più preziosa. Desidero anche ringraziare tutte le persone che ho avuto il privilegio di incontrare durante il mio dottorato. Sia i miei colleghi che i professori e il personale amministrativo hanno contribuito a rendere questa esperienza arricchente. Grazie per le vostre idee, il vostro supporto e la vostra collaborazione. Le interazioni con voi sono state fondamentali per la mia crescita accademica e personale. A tutti voi, i miei più sinceri ringraziamenti. Senza il vostro sostegno e incoraggiamento, la realizzazione di questo dottorato sarebbe stata impossibile. Sono grato di aver avuto l'opportunità di condividere questo percorso con persone così speciali e generose.

Abstract

Visual object tracking plays a crucial role in various vision systems, including biometric analysis, medical imaging, smart traffic systems, and video surveillance. Despite notable advancements in visual object tracking over the past few decades, many tracking algorithms still face challenges due to factors like illumination changes, deformation, and scale variations.

This thesis is divided into three parts. The first part introduces the visual object tracking problem and discusses the traditional approaches that have been used to study it. We then propose a novel method called Tracking by Iterative Multi-Refinements, which addresses the issue of locating the target by redefining the search for the ideal bounding box. This method utilizes an iterative process to forecast a sequence of bounding box adjustments, enabling the tracking algorithm to handle multiple non-conflicting transformations simultaneously. As a result, it achieves faster tracking and can handle a higher number of composite transformations.

In the second part of this thesis we explore the application of reinforcement learning (RL) to visual tracking. Presenting a general RL framework applicable to problems that require a sequence of decisions. We discuss various families of popular RL approaches, including value-based methods, policy gradient approaches, and Actor-Critic Methods. Furthermore, we delve into the application of RL to visual tracking, where an RL agent predicts the target's location, selects hyperparameters, correlation filters, or target appearance. A comprehensive comparison of these approaches is provided, along with a taxonomy of state-of-the-art methods.

The third part presents a novel method that addresses the need for online tuning of offline-trained tracking models. Typically, offline-trained models, whether through supervised learning or reinforcement learning, require additional tuning during online tracking to achieve optimal performance. The duration of this tuning process depends on the number of layers that need

training for the new target. However, our thesis proposes a pioneering approach that expedites the training of convolutional neural networks (CNNs) while preserving their high performance levels.

In summary, this thesis extensively explores the area of visual object tracking and its related domains, covering traditional approaches, novel methodologies like Tracking by Iterative Multi-Refinements, the application of reinforcement learning, and a pioneering method for accelerating CNN training. By addressing the challenges faced by existing tracking algorithms, this research aims to advance the field of visual object tracking and contributes to the development of more robust and efficient tracking systems.

Contents

Chapter 1

Introduction

Visual object tracking represents one of the most important fields of computer vision since is used in many applications, such as video surveillance, autonomous driving, human-machine interaction and activity recognition. The success of these applications is related to the goodness of the visual object tracking algorithms. Furthermore, when talking about visual tracking, approaches depend on the camera setting: single (static, PTZ or mobile) camera or camera network (with overlapping or non-overlapping fields of view). This thesis focuses on visual object tracking (VOT) with a single mobile camera.

1.1 Overview

The formulation of the VOT problem is quite simple: given the initial location and size of the target in a frame, the visual tracker aims to estimate the target's position in the subsequent video frames.

The main components of the visual tracking process are: *target initialization, appearance model, motion prediction, and model update, motion model. Target initialization* happens on the first

frame of the video and aims at defining the initial target's location in terms of bounding box, PTZ camera parameters, or target's contours, depending on the camera setting and application at hand. *Appearance model* is the representation of the target taken from the initialization. It is designed to be discriminative as much as possible to distinguish the target from the other elements in the scene (the background), and to characterize the visual appearance of the target. Motion prediction is an important aspect of visual object tracking, as it involves forecasting the future position of an object based on its previous motion. *Model update* aims to update the target appearance representation during tracking by using the last status detected on the last frames. Recently, state of the art trackers allow skipping this part by speeding up the entire tracking process. The *motion model* is an important component of motion prediction in visual object tracking. It describes how the position of the object changes over time and is used to forecast its future position. The motion model parameters are estimated using machine learning techniques such as regression or deep learning, or optimization techniques such as the Kalman filter or particle filter.

To predict the status of the target, the tracker relies on the appearance model and during the years the methods used to extract the target's feature representation are changed a lot. Broadly speaking, we can divide the extracted visual features into two categories: handcrafted features and deep features.

Before the breakout of Deep Neural Networks(DNNs), features were extracted from the images depicting the target by handcrafted methods such as Histogram of Oriented Gradients (HOG)[1], Scale-Invariant Feature Transform (SIFT)[2], color histograms, bag-of-features[3].

Compared to the handcrafted methods, features extracted by DNNs are more invariant to the appearance changes, and the success of such models seems to be ascribable to their capability of extracting higher-level features through their multiple layer structure and their non-linearity. The typical neural network used for the Visual Tracking are the Convolutional Neural Networks (CNNs).

Regardless of the methods of extracting features, three main strategies are used for visual object tracking: *tracking-by-detection, tracking-by-regression, and tracking-by-detection and regression.*

Tracking-by-detection approaches [4, 5] are classification methods, which aim to discriminate between target and background. A large number of candidate target bounding boxes are drawn around the last known target location. The one yielding to the highest classification score is selected. As a result, performance is closely linked to the sampling strategy.

Tracking-by-regression approaches [6, 7] use regression to locate the bounding box in subsequent frames by minimizing an objective function such as least-squared error.

Tracking-by-detection and regression methods [6, 8, 9, 10] are hybrid approaches whose goal is detecting the most similar region to the target, and then refining the region through regression.

Tracking-by-segmentation approaches [11, 12, 13, 14] aim at tracking objects while producing a mask of the target object for each frame of the processed video.

Despite CNNs are more discriminative and robust to represent the appearance, they require an huge amount of data that in some cases are difficult to obtain, but in the last few years many benchmarks have been released in computer vision especially in visual object tracking field allowing to fill this lack. Nonetheless, some

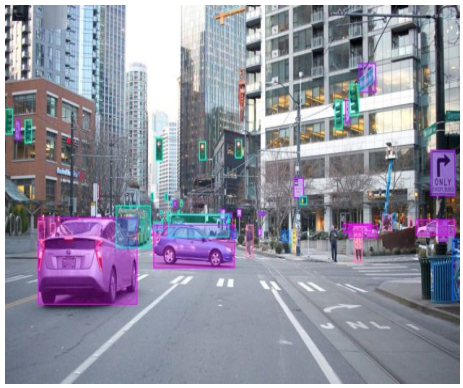
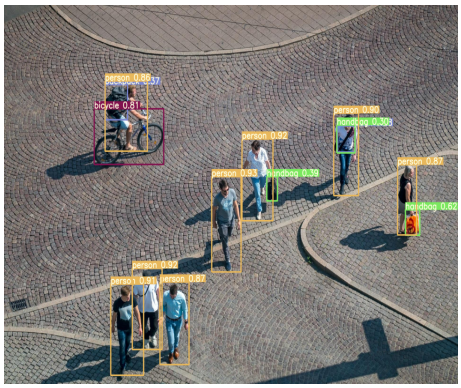
well noted problems of the visual tracking still remain unsolved such as illumination change, scale variation, occlusion, etc. So far, the state of the art (SOTA) tracker have not completely solved these issues, but since the performance of the visual object tracking algorithms are increasing, the applications on real world are growing too.

1.2 Applications

There are many applications of the visual object tracking in real world. The typical applications are video surveillance and autonomous driving.

Video Surveillance intervention in tracking a particular object. Video surveillance has been grouped into three types, namely manual, semi-autonomous, and self-autonomous. Manual video surveillance involves the analysis of the video content by a human. Such systems are currently widely used. Semi-autonomous video surveillance involves some form of automatic video processing, but with significant human intervention. Typical examples are systems that perform simple motion detection. Only in the presence of significant motion, the video is recorded and sent for analysis to the human expert. In a fully autonomous system, the only input is the video sequence taken at the scene where surveillance is performed. In such a system, there is no human intervention, and the system does both the low-level tasks, such as motion detection and tracking, and also high-level decision-making tasks, such as abnormal event detection and gesture recognition.[15]

Autonomous Driving represents one of the ambitious applications of visual tracking especially in the last decade since it is becoming more attractive for companies and academia. Autonomous

A  B 

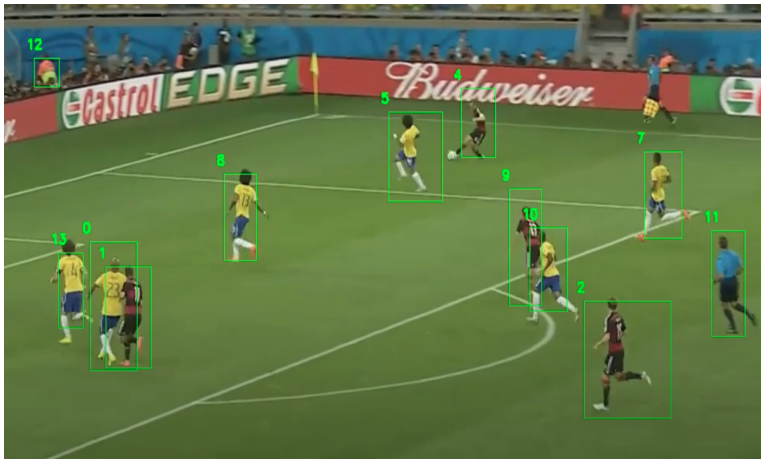
C 

Figure 1.1: Visual tracking has numerous applications across various fields. For instance, Image A depicts an example of visual tracking used in autonomous driving, whereas Image B showcases its utilization in video surveillance. In Image C, we see visual tracking applied to sports analysis, demonstrating the versatility of this technique.

driving means that a car is able to sense the environment and perform actions as a consequence of what it perceived without the help of human. Sensing the environment means that the car has to avoid eventual risks by using several sensors. The perception system usually consists of multi-sensors including LIDAR, GPS receiver, cameras and radars. All sensors give information about the position of the obstacles; cameras and visual tracking algorithms allow to calculate the trajectories of the obstacles by predicting their future movements. This information is important to help the autonomous car to plan paths and avoid obstacles [16].

Traffic monitoring is a popular application of visual object tracking (VOT) that involves tracking vehicles or pedestrians in real-world traffic scenarios helping to improve road safety [17] or optimize traffic flow [18]. Different approaches that have been proposed for traffic monitoring using VOT. These could include traditional feature-based trackers, deep learning-based trackers, or hybrid methods that combine both.

Motion analysis of athletes in sports has gained considerable attention from researchers. Traditionally, analysis has relied on data collected from live observation or post-event video analysis [19], which necessitates manual recording and analysis. Thus, automating this manual process with visual tracking is highly sought after. In Figure 1.1, visual tracking is used to track football players, who move rapidly with unpredictable changes in direction, making manual collection even more challenging. Visual tracking can predict player trajectories and provide their positions at each time instant, enabling applications such as strategy planning, team management, and individual performance evaluation. These benefits extend to both players and coaches.

The multitude of applications and the persistent challenges

present in this field have ignited my enthusiasm to undertake this thesis. By delving into the complexities and possibilities of visual object tracking, I aim to contribute to the existing knowledge and advancements in this dynamic area of research.

1.3 Motivation

Visual Object Tracking still represents a challenge for the computer vision community due to well known problems such as illumination changes, fast (non-linear) motion, scale variation, occlusions, target rotation and pose changes. The data acquisition process itself is imperfect and the target appearance is affected by the environment where the target moves. In particular, an unreliable appearance model affects the performance and it becomes harder for the visual tracker making the correct prediction on the position of the target. We have already claimed that appearance features are nowadays extracted through CNNs. Therefore, how CNNs learn and what they learn could represent a breakthrough point to propose novel visual tracker capable of facing the current challenges.

Essentially, we can distinguish different types of learning of CNNs: supervised, unsupervised and reinforcement learning.

Supervised learning (SL) algorithms exploit a dataset in which a label or a scalar is associated to each example. Traditional goals of SL are the solution of regression and classification problem. In classification problem, the trained model predicts the class among a finite number of known ones to which the input sample belongs. In regression problem, the goal of the model is to predict numerical values.

Unsupervised learning (UL) algorithms exploit a dataset to

highlight features of the data structure. Usually UL algorithms aim to learn the entire probability distribution that generated the data as in density estimation, or implicitly, for tasks-like synthesis or denoising. Other unsupervised learning algorithms are used to cluster data of similar examples.

The basic idea in reinforcement learning (RL) is letting the system, called Agent, to partially exploit its experience or explore new possibilities during training. While in SL exemplars and corresponding expected answers are given, in RL the Agent is rewarded a posteriori for the effective behaviors, and penalized for the ineffective ones. Reinforcement Learning can solve problems by using a variety of machine learning methods and techniques, from decision trees to SVMs, to neural networks. The latter is what the “deep" part of deep reinforcement learning (DRL) [20] refers to. DRL has been applied in several computer vision problems such as visual tracking [21], activity localization [22], object detection [23], video recognition [24] and segmentation [25].

Despite the DRL for tracking is applied in different ways, from target location prediction to hyper-parameters setting, there are still limitations and open problems to solve. In facts, analysis of DRL-based tracking algorithms shows that, in general, DRL-based visual tracking approaches are often below the state-of-the-art.

The use of DRL for tracking could be useful to learn behaviors that allows the agent to maximize reward generating samples he needs to make more confident decisions during tracking (exploration). In this way, the agent is allowed to self-correct its behavior. From another point of view, it would be interesting to understand if an agent trained using the DRL is able to generalize a behavior in visual tracking better than an agent trained using

SL.

This question drove me to investigate the DRL methods applied to visual object tracking and at the same time do a comparison between the learning paradigms DSL (deep SL) and DRL applied to visual object tracking.

In visual tracking there are three types of algorithms, those that update the appearance model during tracking by updating the parameters of the network in an online manner, those that make only one update on the first available target appearance and those that use the first template of the target available without any updates.

Considering the amount of data available for visual tracking and the time required to train the CNNs, we investigated the possibility to speed up the learning of CNNs using a sort of approximation learning.

1.4 Research contributions

The goals of this thesis are manifold: comparing SL and RL learning paradigms for visual object tracking by highlighting pros and cons of each approach, proposing a novel method for visual object tracking, and proposing a new method to speed up the training of CNNs for visual tracking algorithms and other neural networks.

Throughout the experimental process, it was observed that significant improvements could be achieved in a visual tracker by employing iterative refinements of the candidate bounding box for the target. By iteratively refining the bounding box within the same frame until the optimal configuration was attained, tracking performance could be enhanced. As a result, a novel method was developed to select multiple refinements of the bounding box.

This approach not only accelerated the tracking system but also improved overall tracking performance. The details of this method were published in [26].

This thesis also delves into the investigation of different learning paradigms that enable deep model learning and generalization in visual tracking. The primary objective was to determine the applicability of Deep Reinforcement Learning (DRL) in the context of visual object tracking. A comprehensive study was conducted to explore the use of DRL in visual tracking, and the findings were published in [27].

In pursuit of the objective to update the deep model online during tracking while enhancing time performance, thorough investigations were conducted to expedite the training of Convolutional Neural Networks (CNNs). Drawing inspiration from previous works in this domain, a novel method was proposed. This approach involved analyzing the gradient of the loss with respect to the model parameters to determine the opportune moments and locations for truncating the CNN. Additionally, it leveraged the feature maps extracted by the last layer before the truncation. Although this work has been submitted for publication, it is yet to be officially published.

Each chapter of this thesis details some scientific contribution, which can be summarized as following:

- Comparison of SL and RL learning paradigms for visual object tracking by analysing the characteristics of each method and investigating pros and cons.

- A new method for visual object tracking based on multiple transformations of the bounding box to speed up the tracking process. As a consequence, a higher number of composite

transformations can be handled by the model without any increases of its complexity;

- A new framework is proposed to speed up the training of CNNs by looking at the gradient magnitude of the weights of the CNN. The framework generalize to several fields.

1.5 Structure of the thesis

After introducing the visual tracking problem and providing a general overview on the challenges to solve and on the most common applications in the real world, this thesis is organized as it follows. Chapter 2 provides a deeper background on the visual object tracking problem by presenting traditional methods and state of art approaches. The chapter also illustrates how deep learning is used in visual tracking. It also details how the tracking problem is approached when using SL methods and DRL.

In Chapter 3, the visual tracking problem is formulated as iterative refinements on the same frame to find the optimal bounding box. The presented method adopts SL to train the model.

In Chapter 4, we provide an introduction to RL and its application in visual tracking. We discuss the different DRL paradigms used in this area and present an overview of all the existing DRL visual tracking methods.

In Chapter 5, a framework is formulated to speed up the training of CNNs. We compare different metrics to analyse where and when truncating the CNNs.

In Chapter 6, we will delve into the findings from the previous chapters.

Finally, Chapter 7 discusses the thesis's contributions and potential directions for future work.

Chapter 2

Visual Object Tracking

Visual tracking is an active research area in computer vision. Formally, the problem can be formulated as the one of analyzing a video, namely a sequence of N images $v = \{I_1, I_2, \ldots, I_N\}$, and estimating the target's location $l_i = (x_i, y_i)$ with $i \in [1, N]$ indexing the image frames. The problem can also be extended to the 3D when a world coordinate system is known or can be estimated. However, this thesis focuses on methods devised for 2D object tracking with a single camera.

The performance of existing techniques is still challenged by various factors such as occlusions and abrupt changes in the visual appearance of the target. To better understand how to deal with visual tracking, in this chapter are summarized the most popular approaches starting from the traditional ones up to the state of the art methods.

2.1 Traditional methods

Taking inspiration from control systems, for years tracking has been modeled by linear dynamical systems and approaches such as Kalman filter [28] have been adopted [29]. Since targets gener-

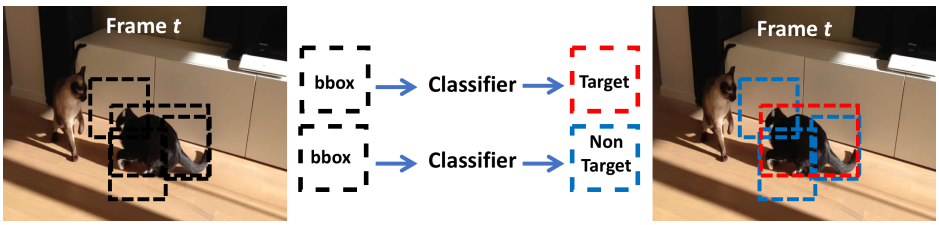


Figure 2.1: In tracking-by-detection, feature representations of image patches sampled around the last known target location are extracted, then the classifier is used to discriminate between target/non-target.

ally do not move linearly, advanced techniques such as Extended/ Unscented Kalman Filter [30, 31] and Monte Carlo based Bayesian filters like particle filter [32] have been adopted as well [33, 30, 34]. The above approaches modeled the target motion dynamics, and appearance models were used as correction tools, in the Kalman filter, or to estimate the observational likelihood in the particle filter framework. Hence, an unreliable appearance model affects the performance of such methods; this has led to many studies about people re-identification, whose goal is to re-identify persons from just one or few shots. Such methods aim at modeling reliably the target appearance by devising novel hand-crafted feature representations or defining/learning similarity functions to be used to establish matches among pedestrian images [35].

2.1.1 Tracking-by-detection

Some methods are based on the tracking-by-detection paradigm [36, 37]. These trackers are trained on test sequences and the detector is specialized in recognizing the target in the scene, turning the visual tracking problem in a binary classification of target/background. These trackers are based on strong feature representations, a classifier, an online procedure to re-train the classifier

at test time, a strategy to assemble an appropriate training set of positive and negative samples collected at test time. Fig. 2.1 shows the typical application of tracking-by-detection algorithms. First, 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. The patch with the most confident positive response of the classifier may be chosen as the new target location. The training strategy used to retrain the classifier with the new target appearance brings uncertainty that generally results in drifting of the tracker.

2.1.2 Tracking-by-correlation

Another approach to visual tracking is template matching [38]. Given a template of the target, tracking is performed by correlating the template with the next image representation to predict the target location using the peaks in the correlation results [39].

In some cases, the matching is established by the sum of squared differences [38] of the pixel intensities/ feature representations. Despite simple, in practice, this approach fails due to the inability of the tracker to adapt to the target appearance changes and may not be able to discriminate between target and background. Interesting results can be achieved by overturning the problem and learning the template that allows localizing the target in the training images. Training may be performed in a discriminative way to estimate a template able to respond to the target rather than to the background [40]. This idea is at the basis of the correlation filters used in recent years in visual tracking. Training of the correlation filter (the template) can be done in the Fourier space where correlation turns into an element-wise product. Supervision to the Average of Synthetic Exact Filters (ASEF) [40] is

provided in terms of heatmap whose peak indicates the location of the target in the original image and can be represented in terms of a Gaussian distribution. Correlation filters (CFs) estimated for each image in the training set are then averaged.

ASEF-like CFs require many training samples and might not be ideal for tracking in real-time. Minimum Output Sum of Squared Error (MOSSE) [41] was proposed to learn ASEF-like filters with limited samples to be specifically used in visual tracking. The main difference between ASEF [40] and MOSSE [41] filters is that training the latter entails a least-square minimization problem for which a closed-form solution can be found and computed in an efficient way.

CFs assume there is a periodicity in the images that can produce unwanted boundary effects, and the detection scores are accurate only near the center of the region. To alleviate this problem, spatially regularized discriminative CF (SRDCF) [42] penalizes CF coefficients based on the a priori known spatial extent of the filter. Overall, as shown in Fig. 2.2, to train CFs, the search region is element-wise multiplied with a 2D weight function; its FFT is element-wise multiplied with the CF and IFFT allows computing an heatmap whose peak indicates the target location. An image sample set is built around the target location and used to retrain the CF.

2.2 Recent Trends in Deep Visual Tracking

Since CNNs have taken their place in image processing, their use has expanded to include visual tracking.

With the term deep learning, we refer to a class of methods (generally, artificial neural network (ANN)) that processes data

by a cascade of non-linear functions (referred as layers in ANN) to progressively extract higher-level features. Training deep models aims at estimating the parameters of these non-linear functions such that an error function is minimized. Since each non-linear function transforms its input into more general representations to feed subsequent non-linear functions, training results in learning a feature representation of the input. It is believed that deeper models (i.e, networks with high number of layers) are able to extract better features than shallow models [43].

In computer vision, where inputs are mostly images, CNN [44] are in general adopted. In such networks (see Fig. 2.3(a)), the basic operation is the image convolution, and training learns the convolutional kernel weights.

CNNs have demonstrated to be well suited for pattern recognition, image segmentation and object detection. Several architectures have been proposed, among which VGG [45], AlexNet [46], GoogleNet [47]. In some cases, fully convolutional neural net-

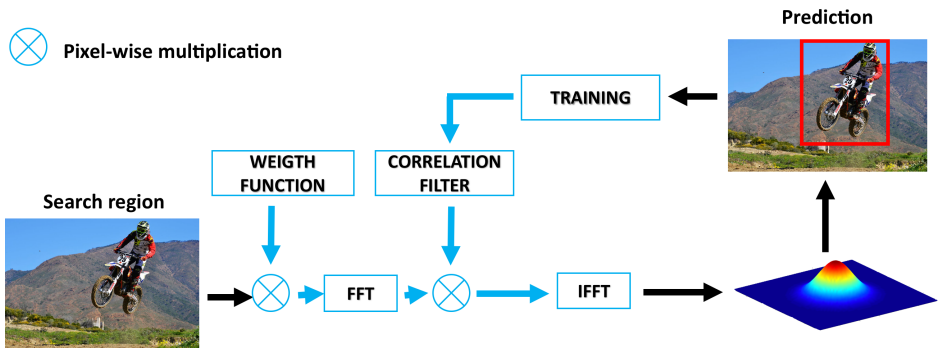


Figure 2.2: Training of a CF. The search region is element-wise multiplied with a 2D weight function, then FFT is computed and element-wise multiplied with the CF. The IFFT allows computing an heatmap whose peak indicates the target location. Based on such target location, an image sample set is assembled to retrain the CF.

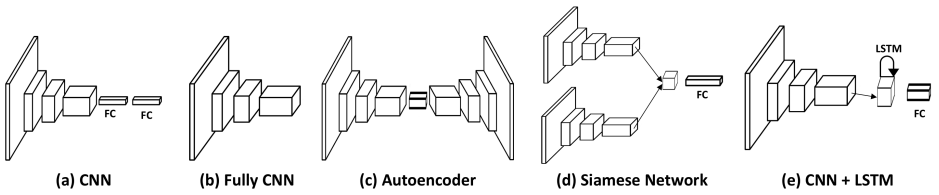


Figure 2.3: The most commonly adopted architectures for tracking. (a) CNN model where convolutional layers are followed by fully connected (FC) ones. (b) Fully convolutional model can output feature maps. (c) Autoencoder can encode and decode its input. (d) In the Siamese network, the two branches share the same architecture and parameters. A layer is used to merge the branch outputs. Additional FC layers can be also added. (e) LSTM works like a memory of its inputs (generally from a CNN). Additional FC layers can be added to the model.

work [48], where no fully-connected layer is adopted, have been used (see Fig. 2.3(b)). In autoencoders, the first part of the network, encoder, has the goal of extracting a compact feature representation of the input. The second part of the architecture, decoder, is used to reconstruct the input from the compact feature representation. Since the training of the model is unsupervised, sometimes autoencoders have been used to pre-train deep models [49]. Another famous architecture is the Siamese network (see Fig. 2.3(d)). Firstly proposed in [50] for face verification, the Siamese network can learn similarity functions to compare two inputs. It is composed of two branches sharing the same architecture and parameters. On top of the two branches, a layer is used to merge the outputs from the two branches and further processing layers can be added to estimate the output of the network. In visual tracking, such architecture has been extensively used by providing in input to the network a target template and a search area image.

Recurrent Neural Networks (RNN) [51, 52] are usually adopted

to process sequences of data among which temporal relationships exist. A widely used RNN is the Long Short Term Memory (LSTM) [53], which uses its internal state to implement a double memory, respectively long and short term memories. In video processing [54], LSTM uses information from previous frames to represent the information in the future ones. Generally, the input to a RNN is the output of a CNN (see Fig. 2.3(e)).

In recent years, generative adversarial network (GAN) [55] have been used to generate realistic-looking images from random noise. The model consists of two networks playing the roles of discriminator and generator. The former network aims at discriminating between real and fake images; the latter takes in input noise and generates fake images to fool the discriminator without using any supervision at training time. GANs are used to augment data in the image space and have recently been used also in the tracking domain [56].

Chapter 3

Visual Object Tracking: Supervised approach

After the introduction of CNNs, it was not immediately apparent the best way to use CNN for visual tracking. CNNs have been initially designed for image classification and trained on large dataset to be used, at test time, on single images. Time required for training the model is in general very high. On the other hand, visual tracking requires models to process online a sequence of images by taking into account the dynamics of the target. Ideally, the model should be adapted at test time to the target appearance changes. The above requirements make the use of CNNs apparently unsuitable.

However, during the years, CNNs have proved to be more effective than traditional methods, and more sophisticated deep approaches have been developed by following two directions: tracking by test-time model adaptation and tracking by pre-trained models. In the latter approaches, the model parameters are not updated at test time.

This chapter, first reviews the most common supervised approaches falling in these two categories, then presents a novel su-

pervised method to track objects by test-time model adaptation.

3.1 Tracking by test-time model adaptation

First attempts to adapt deep models to the target appearance changes were re-training the model from time-to-time on positive/negative patches cropped from each frame around the target location.

Multi-domain tracking (MDNet) [57] focuses on extracting target-specific features from shared, general feature representations. The model is based on a set of CNNs, each one trained off-line on a specific video, regarded as a domain. All CNNs share the same structure and parameters but differ for the last fully-connected (FC) layers, which account for the target-specific features. At test time, only the shared part is retained while FC layers are retrained to recognize between target/ non-target. The network takes in input candidate target images (cropped images around the previous target position) and returns its confidence on each image. MDNet may not discriminate among similar objects and classifies several patches to locate the target. Nonetheless, the MDNet strategy of only retraining the last FC layers has been frequently adopted in deep tracking.

In real-time MDNet (RT-MDNet) [58], the tracking process is speed-up by including a RoiAlign layer [1] to improve target lo-

[1]RoiAlign layer [59] was originally introduced for object detection to improve RoIPool. In [60], a CNN produces a feature map for the input image. The RoI pooling layer (RoIPool) [60] uses max-pooling to compute a small fixed-size feature map from the features within a region of interest (RoI, it represents an object proposal) and speedups the feature extraction process for all object proposals. Since the size of the pooling operator depends on the size of the RoI and is computed by quantization, misalignments between RoI and extracted features are introduced. The RoIAlign layer [59] removes the harsh quantization of RoIPool by using bilinear interpolation.

calization during tracking. Fully convolutional layers are used for learning shared feature representations; an adaptive RoIAlign layer computing a denser feature map by dilated convolutions is used for extracting features of each region of interest (RoI); FC layers are adapted at test time based on the selected RoI. To improve the discriminative capabilities of the shared representations, the network is trained to keep targets of different videos apart from each other.

MDNet [57] and RT-MDNet [58] re-train only part of the network, but tracking performance might degrade considering that the samples used to re-train the network are uncertain and, in the long-term, this may lead to tracking drift. The work in [61] proposes to regard a CNN as an ensemble where each kernel is considered a base learner and updated using a different loss criterion to avoid that the learned features are highly correlated with each other. At test time, fine-tuning of the CNN built upon a pre-trained network is then formulated as a sequential ensemble learning problem.

Other works have replaced the fine-tuned FC layers with correlation filters. In [62], discriminative CFs are trained on convolutional features from a pre-trained VGG model. It is shown that features from the first layer provide superior tracking performance compared to the ones from deeper layers. In [63], CFs are adaptively trained on the outputs of each CNN layer to use semantics and fine-grained details for handling large appearance variations. Linear interpolation is used to accommodate the varying feature map size along the network depth. Target location is estimated by a coarse-to-fine searching approach.

Recently, in ATOM [64], three modules are used. The target estimation module is trained offline to predict the intersection over

union (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 within the tracking-by-detection framework 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. Such robust features are generated by letting the generator to produce a weight mask to be applied to the extracted features. Each mask represents a specific appearance variation. While the model learning progresses, the generated mask focuses on more and more discriminative target features. At test time, the model is incrementally updated frame-by-frame.

Recently, the work in [65] (DiMP) highlights limitations of the Siamese learning framework such as the inability to consider background information when discriminating between target and background, the fact that the learned similarity measure may not be reliable for objects not included in the training set, the difficulty to update the Siamese network parameters at test time. DiMP proposes a two branch network; one of the branch is used to predict a model of the target appearance in terms of weights of a convolutional layer. This convolutional layer is used to perform target classification on the feature map extracted from the test frame by the second branch. Since the method learn to produce target appearance models, it can be considered a meta-learning approach.

The method has been improved in [66] by integrating a re-

gression formulation into both branches of DiMP. The model is trained by minimizing the KL divergence.

3.2 Tracking by ad-hoc pre-trained models

Instead of adapting the models online, another approach is to use pre-trained model trained (Siamese network) [67, 68, 69, 70] to find the target location processing the search region using the target features as appearance information.

The available implementations differ in the output of the network. Indeed, at test time, the network can predict the target's bounding box by ranking proposed candidate boxes [67], regressing it directly from images [70], estimating its centre position and scale [69, 68].

In SINT [67], each branch of the Siamese network takes in input an image and several bounding box proposals. A RoiPool layer [60] is used to speed-up the feature extraction for each proposal. The network incorporates optical flow to improve tracking. At test time, a radius sampling strategy is used to sample bounding boxes at different scales around the last predicted target location. The bounding box that best matches the input target image is selected and refined through a regressor trained on the first video frame.

Branches of the Siamese network in SiamFC [68] are fully convolutional and the last layer of the model computes a cross-correlation map whose highest value should indicate the target location. Large displacements are penalized by multiplying the score map by a cosine window and the score map is up-sampled to the original frame size by bicubic interpolation to improve target localization accuracy. The target is searched at different scales.

Instead of processing multiple images, GoTurn [70] uses the last FC layer to regress the target bounding box location and size, and can easily handle scale and aspect ratio changes. To account for the invariance to translation typical of CNN, the training set is augmented to show the target at different locations within the search area.

YCNN [69] 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 is that the target appears in the search image. Since the last layers are fully connected, the size of the search region cannot be dynamically adapted.

In Siamese region proposal network (Siamese-RPN) [71], 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 [72] feature map. At test time, meta-learning is used by reinterpreting the output of the template branch as parameters to predict the detection kernels. Only the detection branch is retained, thus yielding to a speed up of the tracking process. One-shot detection is done in the other frames by using the detection kernel estimated on the first one.

In [73], 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. Firstly, padding in deep networks tends to destroy the translation invariance property. Secondly, RPN [72] requires asymmetrical features. To account for such issues, a novel sampling strategy is proposed to ensure spatial invariance. Furthermore, multi-branch features are extracted from various layers of the network to infer the target location. Finally, depth-wise correlation is included in the model

to reduce computational cost. The resulting model is referred to as *Siam-RPN++*

Instead of using a Siamese net, the work in [74] 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 estimated filter to the feature map of the next frame to estimate the target response map.

3.3 Tracking by iterative refinements

Currently, the state-of-the-art performance in visual tracking is achieved by utilizing deep learning methods [75, 76].

As already mentioned, one such algorithm is MDNet, which was proposed in[57]. MDNet is a tracking-by-detection and regression algorithm that involves categorizing a set of bounding boxes, sampled every frame around the last known target location, into either target or background. The bounding box with the highest classification confidence score is then adjusted through regression.

The CNN used in MDNet is trained in a multi-domain fashion. Despite not being the most recent tracker, MDNet still achieves state-of-the-art performance on the well-known OTB benchmark [77]. However, MDNet has two primary limitations: first, it requires sampling and classification of several bounding boxes at each frame to select the optimal one; second, it uses a regression model to refine the chosen bounding box.

Recent studies have attempted to enhance the performance of MDNet by redefining the search for the ideal bounding box through either an iterative process, where a discrete sequence of bounding box adjustments is forecasted to locate the target, as

demonstrated in [78, 79, 80, 81, 82], or by regressing the bounding box coordinates at each frame, as seen in [83].

In [23, 78, 84], the bounding box is refined iteratively at each frame through shift (moving right/left/up/down on the image plane) and scale (reducing/enlarging the bounding box size) adjustments. The identity transformation is also incorporated to account for cases in which the bounding box must be accepted as it is. To implement this approach, the deep learning model receives the image patch that corresponds to the presently estimated target bounding box as input and produces, as output, the likelihood associated with each possible bounding box refinement. The refinement with the highest probability value is applied to the bounding box, and the process is repeated until either a maximum number of iterations or the null transformation is selected. The process is summarized in Figure 3.1A.

In this thesis we focus on improving these types of approaches after observing that the iterative process described above has several disadvantages. Firstly, only one transformation can be applied per iteration, which results in a high computational cost to determine the optimal bounding box. Secondly, the strategy introduces ambiguity during the model parameter learning process.

Specifically, supervised training of the model is carried out by providing the target patch and the bounding box refinement type that should be applied to enhance the tracking result. However, the effect of applying a transformation can be measured by estimating the intersection-over-union (IoU) value of the resulting bounding box and the ground-truth, as shown in Figure 3.2.

Often, more than one transformation can result in an improved target localization, but this is typically ignored by the supervised training procedure in [78, 84, 79], which uses the 1-hot encoding

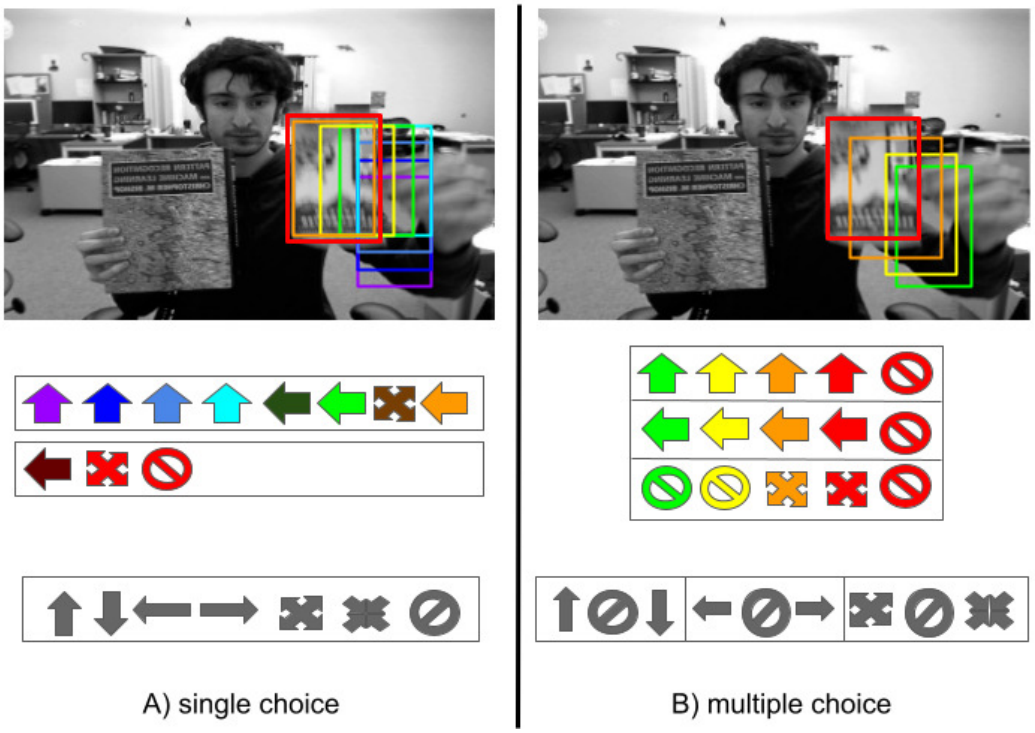


Figure 3.1: (**A**) In iterative single refinement, starting from an initial bounding box, a sequence of transformations to the bounding box is predicted and applied to locate the target. (**B**) With multiple refinements, a sequence of multiple non-conflicting transformations is predicted and applied.

schema to select the refinement and prioritize bounding box refinement over another by considering the first transformation with the highest IoU. One of the key contributions of this thesis is to enable the selection of multiple transformations during training, using multiple 1-hot encoding to indicate the transformations that might be applied to enhance the bounding box. Additionally, it should be noted that bounding box transformations may cancel each other out, resulting in conflicting transformations, such as shifting the bounding box to the left and right. As mentioned in [84], such refinements canceling each other out can cause cycling behaviors.

This thesis proposes a new approach to address the issues present in the traditional method of selecting the bounding box refinement during target tracking. In particular,

- It introduces the concept of conflicting refinements and train our model to handle them.
- It formulates the problem in such a way that multiple non-conflicting transformations can be applied simultaneously, leading to faster tracking with the ability to handle a higher number of composite transformations.
- It eliminates ambiguity during the training procedure by avoiding giving priority to some refinements over others.

3.4 Multi-Refinements of the Bounding Box

At time t, the candidate target bounding box is defined as a 4-dimensional vector $b_t = [x_t, y_t, w_t, h_t]$, where (x_t, y_t) is the center coordinates and w_t, h_t represent the width and height of the

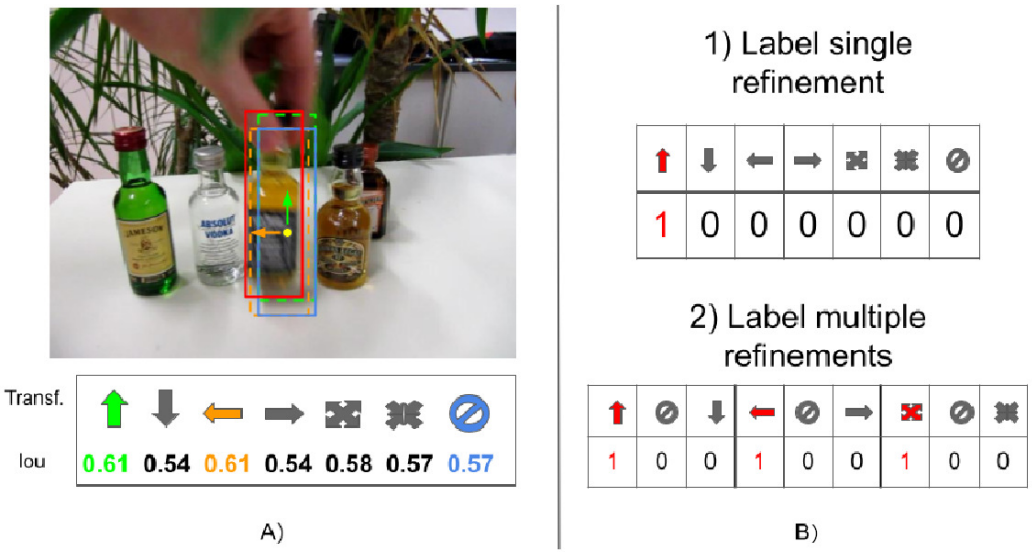


Figure 3.2: Image (**A**) shows the IoU values calculated after each of the possible refinements. Image (**B**) shows how 1-hot-encoding is used to annotate the data in single and multiple bounding box refinements.

bounding box, respectively. Given an image frame F_t, the image patch p_t is obtained by cropping F_t based on the bounding box b_t through the selection patch function $f_b(\cdot)$. The function can also include pre-processing steps to adapt the resulting image patch to the network input:

$$p_t = f_b(b_t, F_t). \tag{3.1}$$

Let V be the number of basic linear transformations that can be used to refine the bounding box b. These transformations define the space Φ of allowed discrete bounding box transformations. Let φ be a subset of k transformations $\varphi = \{\phi_1, \phi_2, \ldots, \phi_k\} \subseteq \Phi$. A transformation $\theta(\cdot) \in \Phi$ is *conflicting* with the transformations in φ if

$$b = \gamma_i(\theta(b)) \tag{3.2}$$

where γ_i indicates any sequence of transformations in φ. In other words, bounding box changes operated through the refinement θ are canceled out by some of the refinements in φ.

If instead, for any sequences γ_i

$$b \neq \gamma_i(\theta(b)), \tag{3.3}$$

then $\theta(.)$ is *non-conflicting* with the transformations in φ.

Keeping in mind the above definitions, we propose to split the bounding box refinements into N groups such that each transformation in one group is not in conflict with the transformations in all the other ones. We include within each of these groups a null transformation, namely, the identity transformation. In the following, the obtained groups are named *non-conflicting* groups.

As an example, let us consider the following discrete bounding box transformations: $\Phi = \{$*Left(*Δ*)*, *Right(*Δ*)*, *Up(*Δ*)*, *Down(*Δ*)*$\}$, where the name of the transformation indicates the shifting direction of direction into which shifting the bounding box, while Δ represents the number of pixels by which the bounding box is shifted. A possible partition into two non-conflicting groups, G_h and G_v, is $G_h = \{Left(\Delta), Right(\Delta)\}$ and $G_v = \{Up(\Delta), Down(\Delta)\}$.

3.4.1 Output Layer for Non-Conflicting Refinements

To deal with multiple non-conflicting refinements of the bounding box, we need to properly design the output layer of the deep model used within the tracking strategy. Figure 3.3 shows, on the top, the typical deep model used for iterative bounding box refinements [23, 78, 84]. At each iteration, the model takes as input an image patch and provides the probabilities of each of the V transformations $\phi_i \in \Phi$. Only the transformation with the highest probability is applied.

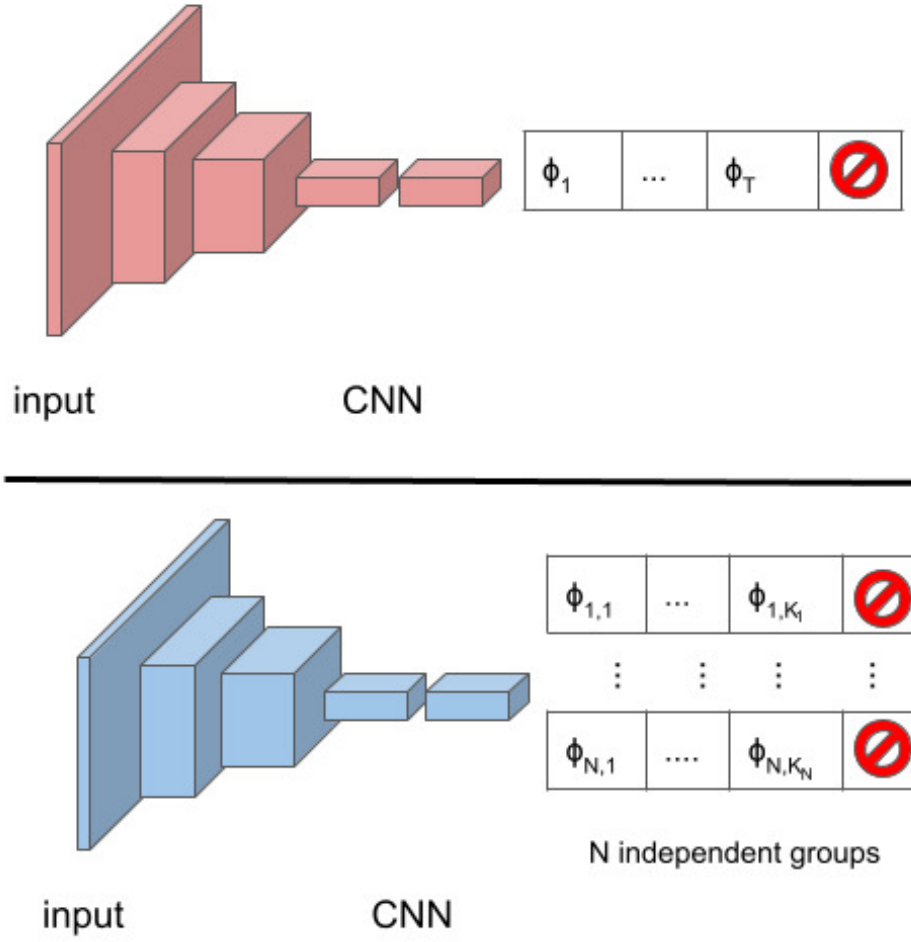


Figure 3.3: On the top, the model estimates the probability distribution over all the transformations. On the bottom, the model estimates N probability distributions, one for each group of transformations.

On the bottom, Figure 3.3 shows how the model needs be adapted to deal with N non-conflicting transformation groups. The network will provide N probability distributions over the k_i transformations belonging to each group, with i varying in $[1, N]$. The distributions are computed independently from each other, and N different non-conflicting transformations are applied, one from each group.

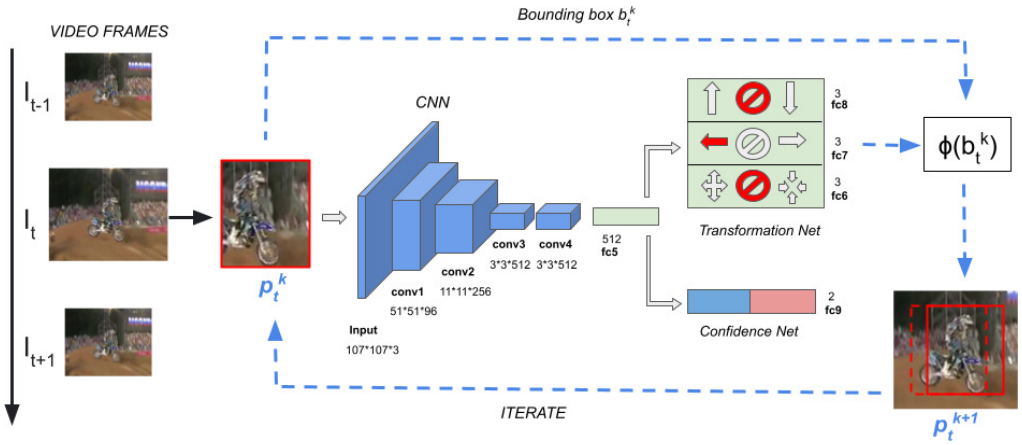


Figure 3.4: Architecture of our model. On the left, the arrow indicates the flow of frames. At time t, starting from the bounding box estimated at time $t-1$, the method iterates a sequence of multiple transformations highlighted in red on the right side. The transformations are applied to the bounding box, which is used to get a new image patch to feed the model.

3.5 Tracking by Iterative Multi-Refinements

To demonstrate our idea, we implemented our own tracker. Our tracking architecture is template-free, and requires online fine-tuning of the parameters to adapt the model to the target appearance changes over time. As shown in Figure 3.4, our tracker takes advantage of one network with multiple output branches sharing the convolutional layers and the first dense layer. The first N output branches, namely, the subnet Transformation-Net, estimate N probability distributions over the refinements within the N non-conflicting groups of transformations. In the figure, $N = 3$. The last output branch network, namely, the subnet Confidence-Net, provides a confidence score of the classification of the image patch into background/target.

At time t, starting from the target bounding box estimated at time $t-1$, our method uses the networks to estimate a se-

quence of linear transformations of the bounding box to locate the target in the image frame. The process ends when, for each non-conflicting transformation group, the null transformation is selected or a maximum number of iterations is reached.

The final confidence score is used to assess if the tracking failed, and in this case, a re-detection procedure based on particle filter is used. The entire tracking procedure is presented in Algorithm 1. In the following, we present details about the various steps performed by our tracker.

3.5.1 Designing Non-Conflicting Refinements

In our implementation, each basic transformation depends on parameters calculated based on the size of the current bounding box, similarly to what has been done in [78]. In particular, at the k-th iteration, we consider the values $\Delta w_t^k = \sigma \cdot w_t^{k-1}$ and $\Delta h_t^k = \sigma \cdot h_t^{k-1}$, where σ is a constant value equals to 0.03. These parameters represent the number of pixels by which modifying the center coordinates or the width/height of the bounding box. Based on our experiment, considering the structure of the adopted CNN (a VGG-M), σ equals to 0.03 is the minimum value for the network to notice the effect of the transformations to the bounding box.

We also consider $N = 3$ non-conflicting groups of refinements, each including two basic transformations. The first group, $G_h = \{Left(\Delta w_t^k), Right(\Delta w_t^k)\}$, shifts the bounding box to the left or to the right by adding/subtracting the value Δw_t^k to the coordinate x_t^{k-1}.

The second group, $G_v = \{Up(\Delta h_t^k), Down(\Delta h_t^k)\}$, shifts the bounding box up or down by adding/subtracting the value Δh_t^k to the coordinate y_t^{k-1}.

Finally, $G_s = \{Enlarge(\Delta w_t^k, \Delta h_t^k), Reduce(\Delta w_t^k, \Delta h_t^k)\}$, rescales the bounding box to decrease or increase its size by adding or subtracting the value Δw_t^k and Δh_t^k to the bounding box width and height, respectively. Each of the above groups is augmented to also include the null transformation. These groups were created starting from the transformations used in ADNet [78], separating them into non-conflicting groups and excluding double shifts. The introduction of double shifts, namely, shifting transformations with a doubled value of σ, did not produce significant improvements in our experiments.

3.5.2 Deep Model

The structure of our network is described in Figure 3.4. Similarly to former approaches [57, 78], the first four convolutional layers (conv1–conv4), with weights $(w_1 - w_4)$, are taken from the VGG-M network [85]. The input layer is adapted to our input dimension and subsequent features maps; we used the same input size as in [57]. The fully-connected layer fc5, with weights w_5, is shared by both the Confidence- and Transformation-Net. Layers fc6–fc8, with weights $(w_6 - w_8)$, provide the probability distributions for each of the transformation groups we defined and apply softmax activation functions. Layer fc9, with weights w_9, belongs to the Confidence-Net; with 2 units, it applies a softmax activation function.

3.5.3 Supervised Training of the Model

Offline training: An offline training procedure is used to learn the parameters $(w_1 - w_5)$. This procedure is based on multi-domain learning [57] where layers (conv1-fc5) dealing with domain-

Algorithm 1 Online tracking algorithm.

Input: Pre-trained CNN (w_1,...,w_5)
Initial target b_0^*
Sequence of V frames $\{F_0; F_1; ...; F_t; ..., F_V\}$
Output: Estimated sequence of target bounding-boxes $\{b_t^*\}_{t=1}^{V}$

Randomly initialize parameters w_6-w_9
Draw training samples X_0 around b_0^*, M_S.push(X_0), M_L.push(X_0)
Update parameters w_4-w_9
for $t = 1, \ldots, V$ **do**
    $b_t^0 \leftarrow b_{t-1}^*$
    for $k = 1, ..., max\_iter$ **do**
        $p_t^k = f_b(b_t^{k-1}, F_t)$
        Select N refinements φ_i based on Transformation-Net(p_t^k)
        Apply φ_i with $i \in [1, N]$ to the bounding-box to estimate b_t^k
    end for
    Evaluate confidence score q^* = Confidence-Net($f_b(b_t^k, F_t)$))
    if $q^* > 0.5$ **then**
        $b_t^* \leftarrow b_t^k$
        Draw sample X_t around b_t^*
        Update M_S and M_L by adding X_t and limiting their size
    end if
    if $q^* <= 0.5$ **then** (failure!) apply Re-detection to find b_t^*
        Evaluate confidence score q^* =Confidence-Net($f_b(b_t^*, F_t)$)
    end if
    if $q^* <= 0.5$ **then** Update $w_4 - w_9$ by using M_S
    else if $t\ mod\ 10 = 0$ **then** Update $w_4 - w_9$ by using M_L
    end if
end for

independent information (such as motion blur, illuminations changes, and scale variations) and domain-specific layers (fc6–fc9) are treated differently. In particular, while the former are shared among all videos, the latter are initialized and trained for each video. The number of domains is equal to the number of videos contained in the training dataset. At each training iteration we

use $X = [X_c, X_b]$, where X_c indicates the data used to train the Confidence-Net, and X_b indicates the data used to train the Transformation-Net. The entire model is trained by alternating the training of the Transformation-Net (while freezing the Confidence-Net) and the training of the Confidence-Net (while freezing the Transformation-Net).

Online training: At test time, parameters $(w_6 - w_9)$ are randomly initialized for each video sequence to be adapted online to the target appearance changes, and parameters $(w_1 - w_3)$ are fixed and not trained online to speed up the online training, and to limit overfitting of the network. Online parameter adaptation is done every s frames ($s = 10$) and, whenever a tracking failure has been detected, a re-detection step is performed. A failure is detected whenever the model predicts a confidence score lower than 0.5. Inspired by the works in [78, 57], we update the parameters by using a long-term memory M_L every s frames. This memory stores random samples from the last $mem_L = 1000$ frames. In case of tracking failures, to speed up the model adaptation to the current target appearance, we update the parameters by using a short-term memory M_S. This memory stores random samples from the last $mem_S = 20$ frames.

3.5.4 Sample Generation

Considering N groups of k transformations (including the identity one), there are overall k^N composite transformations. In our implementation, $k^N = 3^3 = 27$. To train the Transformation-Net, we used balanced mini-batches of 81 samples where the 27 composite transformations were equally represented. A grid sampling approach over the 4-dimensional space with a discrete uniformly distributed random step has been used.

To generate balanced mini-batches of 64 samples for training the Confidence-Net, we referred mainly to the technique used by [57]. We used the sampling methods reported in the public code, which slightly differs from the one described in the paper. The sampling is based on normal distributions whose mean and variance depend on the bounding box estimated at the previous iteration. If the sample comes from the first frame, it is considered positive if it yields to $IoU > 0.7$. Otherwise, it is considered positive if the predicted confidence value is >0.5. Furthermore, we used hard negative sampling, meaning that we randomly selected a large number of negative samples from the short memory M_S and select 32 samples with the highest positive classification score. This procedure improves the discriminative abilities of the Confidence-Net.

3.5.5 Implementation Details

Whenever a failure occurs, similarly to the works in [57, 78], we sample 256 candidate target bounding boxes with the same schema adopted to generate mini-batches for training the Confidence-Net. The confidence score of these bounding boxes is evaluated, and the candidate with the highest score is selected as the predicted target bounding box.

As loss function, we adopted the categorical cross-entropy. The learning rate is fixed to 0.001 and the network is trained with SGD with a momentum of 0.9 and weight decay parameter of 0.0005. During offline training of the model, we used 287 domains, that is, 287 videos from the ALOV300 dataset [86]. In particular, for a number of 3 iterations, we sampled 5 frames from each video. For each frame we trained the model for 5 iterations. Training has been done alternating the domains.

At the first frame, layers (conv4-fc9) are trained for 30 iterations on samples generated based on the known target. The Transformation-Net is initialized by sampling 6 times all the possible composite transformations, for a total of 162 samples for each iteration. Online adaptation of the parameters was done for 10 iterations. During tracking, at each frame, 15 negative samples and all transformed bounding boxes with a confidence score > 0.5 were stored in the long and short memories.

Chapter 4

Visual Object Tracking: Reinforcement learning approach

In the previous chapter, we delved into supervised approaches for visual tracking and introduced a novel method. Building upon that foundation, this chapter focuses on the application of Reinforcement Learning (RL) to problems involving sequential decision-making. We begin by presenting the general RL framework, which serves as a versatile tool for tackling such problems.

Subsequently, we explore several prominent families of RL approaches, namely value-based methods, policy gradient approaches, and Actor-Critic Methods. These methods have garnered considerable attention and have proven effective in various domains.

Moving forward, the chapter specifically examines the diverse applications of RL in visual tracking. Within the tracking context, RL agents play a vital role in predicting the target's location within a frame and selecting relevant factors such as hyperparameters, correlation filters, or target appearance.

To provide a comprehensive overview of the field, this chapter not only compares the aforementioned approaches but also proposes a taxonomy of the state-of-the-art methods.

4.1 Reinforcement Learning Framework

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:

$$G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{4.1}$$

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 state $s \in S$ respectively. The *value function* is defined as :

$$\begin{aligned} v_\pi(s) = \mathbb{E}_\pi[G_t|S_t = s] &= \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s \right] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1}|S_t = s], \forall s \in S \end{aligned} \tag{4.2}$$

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 state s, it starts following the policy π thereafter:

$$
\begin{aligned}
q_\pi(a,s) &= \mathbb{E}_\pi\left[G_t|S_t=s,A_t=a\right] \\
&= \mathbb{E}_\pi\left[\sum_{k=0}^{\infty}\gamma^k R_{t+k+1}|S_t=s,A_t=a\right], \\
&\forall s\in S, a\in A(s)
\end{aligned}
\tag{4.3}
$$

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:

$$
q_*(s,a) = \mathbb{E}\left[R_{t+1}+\gamma v_*(S_{t+1})|S_t=s,A_t=a\right]. \tag{4.4}
$$

The optimal value function in 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 [87, 88]. 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. Three main families of model-free approaches can be identified: value-based methods, policy gradient methods and actor-critic approaches.

4.1.1 Rewarding the agent

In RL, the reward signal 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 reward function is refined with the agent's learning progresses [89, 87].

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. Methods of this type are often called imitation learning, learning from demonstration and apprenticeship learning.

A different problem is that of learning a reward function. This problem takes the name of *inverse reinforcement learning (IRL)*, and learns an unknown reward function under which the expert's behavior is optimal. Eventually, by 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 expert behavior is optimal and the problem becomes that of choosing a proper reward function formulation.

4.2 Value-based approaches

Methods in such category aim at learning iteratively the policy by estimating the Q-function. In general, given the current estimate of the Q-function and a state s, the action a is chosen based on a policy The action a might be selected by an ϵ - greedy policy: randomly from the set of possible actions with probability ϵ or $a = argmax_a Q(s, a)$ 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. Once a has been taken, the Q-function is updated.

If the sequence of decisions to improve the Q-function is generated by the learned policy then we have an *on-policy learning*

method; If a different policy is used instead, we have an *off-policy learning* method.

Learning of 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 only 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 [90].

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

$$Q(s,a) = Q(s,a) + \alpha \left[r + \gamma \max_{a'} Q(s',a') - Q(s,a) \right]. \quad (4.5)$$

Thus, the method is off-policy. 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. During learning, with a given probability (say 0.5), only one of the two Q-functions is updated.

4.2.1 Deep Q-Learning

The former techniques can be generalized by replacing the tabular functions with functions parameterized by a weight vector $w \in R^d$ such as those computed by deep models. 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

$$L(w) = \sum_{s,a} \mu(s,a)[q_\pi(s,a) - \hat{q}(s,a,w)]^2 \tag{4.6}$$

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) that yields to the following weight updating rule:

$$w_{t+1} = w_t + \alpha[q_\pi(s_t,a_t) - \hat{q}(s_t,a_t,w_t)]\nabla\hat{q}(s_t,a_t,w_t). \tag{4.7}$$

The target action-value $q_\pi(s_t,a_t)$ can be substituted with the discounted expected return G_t.

The advantage of the former model is that it accommodates for continuous value states, while actions are, in general, still discrete values. In Deep Q-Learning (DQL), training of the network might be done, in theory, by minimizing iteratively the error function on the approximated target value $y_t = EX[r + \gamma maxa'hatq(s',a',w_t)]$ of the Q-function. This leads to the following parameter updating:.

$$\begin{aligned} w_{t+1} = w_t + \alpha[r_{t+1} + \gamma \max_{a'} \hat{q}(s_{t+1},a',w_t) + \\ -\hat{q}(s_t,a_t,w_t)]\nabla\hat{q}(s_t,a_t,w_t). \end{aligned} \tag{4.8}$$

In practice, the above training strategy fails because the sequential states are strongly correlated and the target value is always changing during training. This leads to divergence of the Q-function. Problems in approximating the value function by neural network were pointed out in [91], where it is noted that changes of the parameters during training affect globally the whole policy and make the learning unsuccessful. To account for such issue, 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 batch on a set of transition experiences collected in the form of (s, a, r, s'). However, the work in [91] proposes to accumulate all past experience, which has a computational cost proportional to the dataset size.

The work in [20] was the first proposing to apply DRL by training the network end-to-end directly from visual data. Differently than [91], it proposes to sample a minibatch of tuple (s, a, r, s') from the replay memory. Randomization of the samples breaks correlation among them and reduces the variance of the updates. As a consequence, learning is smoother and oscillations of the parameters, that may led to divergence of the Q-function, are limited. 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 [92].

As for the changing target value y_t from iteration to iteration, a target network different than the optimized one can be used [20] and updated by the learned network parameters every C steps [93].

Later on, in [94], *Double DQL* implements the concept of double Q-learning to avoid that action-values get overestimated. The only change required to transform the DQN algorithm in double DQL is in the computation of the target, which yields to $y_t \simeq \mathbb{E}[r + \gamma \hat{q}(s', \arg\max_{a'} \hat{q}(s', a', w_t), w^-)]$ where w^- are the parameters of the target network, possibly updated every C steps.

To accelerate the training of the DQN, *Dueling DQL (DDQL)* [95] proposes to decouple the estimation of the value of the state $v(s)$ and of the advantage $A(s, a)$ of taking an action a in state s. In

practice, the Q-value can be decomposed in the sum $Q(s,a) = v(s) + A(s,a)$. This approach does not modify the learning algorithm itself but, instead, it modifies the architecture of the network that will be composed of two branches.

Finally, in [96], a continuous variant of the Q-learning algorithm, named *Normalized Advantage Function (NAF)*, is proposed. The approach also decomposes the Q-function into value and advantage functions. However, being the actions continuous value, the advantage function is parameterized as a quadratic function of nonlinear features of the state.

4.3 Policy Gradient Methods

Another family of model-free approaches 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. Since the probability must sum 1, in general a softmax function is used to produce valid probability values.

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:

$$\theta_{t+1} = \theta_t + \alpha \nabla J(\theta_t). \tag{4.9}$$

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[1], 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

[1]This sequence is often called episode or trial.

decisions. In such a case, with $\mu(s) = \sum_a \mu(s,a)$, the policy gradient theorem [87] establishes that:

$$\begin{aligned} \nabla J(\theta) &\propto \sum_s \mu(s) \sum_a \left[q_\pi(s,a) + b(s) \right] \nabla \pi(a \| s, \theta) \\ &= \mathbb{E}_\pi \left[\sum_a \left[q_\pi(s_t,a) - b(s_t) \right] \nabla \pi(a \| s_t, \theta) \right] \end{aligned} \tag{4.10}$$

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. 4.10 is used to develop the most widely adopted algorithm for policy learning: the *REINFORCE* algorithm [97, 87]. In REINFORCE, by taking advantage of Eq. 4.3 and remembering that the baseline does not depend on the policy π, the gradient of the performance measure $J(\theta)$ is computed as:

$$\nabla J(\theta) = \mathbb{E}_\pi \left[\sum_a \pi(a \| s_t, \theta) \left[q_\pi([s_t,a) - b(s_t) \right] \frac{\nabla \pi(a \| s_t, \theta)}{\pi(a \| s_t, \theta)} \right] \tag{4.11}$$

$$= \mathbb{E}_\pi \left[[G_t - b(s_t)] \nabla \ln \pi(a \| s_t, \theta) \right] \tag{4.12}$$

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

$$\theta_{t+1} = \theta_t + \alpha [G_t - b(s_t)] \nabla \ln \pi(a \| s_t, \theta_t). \tag{4.13}$$

A possible choice for the baseline is that of using $b(s) = \hat{v}(s,w)$, a functional learnable approximation of the value function. To implement such modification, we would modify the architecture of the policy network by adding one more output value representing the value of the state (hence policy and value networks might share the rest of the architecture and parameters).

The above methods handle finite number of discrete actions. When actions vary in a continuous space and are infinite number,

the methods can be adapted to determine the statistics of a probability distribution over the actions, such as mean and variance.

4.4 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 variants, the Q-function $\hat{q}(s, a, w)$). In contrast to the REINFORCE [97] with baseline algorithm, in the actor-critic method the estimated value function is used to assess the effect of the taken action, and parameters of the policy network are updated by the following rule:

$$\begin{aligned} y_t &= R_{t+1} + \gamma \hat{v}(s_{t+1}, w) \\ \theta_{t+1} &= \theta_t + \alpha[y_t - \hat{v}(s_t, w)] \nabla \ln \pi(a \| s_t, \theta_t) \end{aligned} \tag{4.14}$$

A variant of AC methods is the *Deterministic Policy Gradient algorithm (DPG)* [98], 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 [98]. During off-policy learning, the deterministic target policy is trained by using a stochastic behaviour policy that ensures exploration. The critic is a differentiable Q-function $\hat{q}(s, a, w)$ and the actor is improved also in the direction of the approximate Q-function gradient with respect to the policy parameters $\nabla_{\pi(\cdot)} \hat{q}(s, a, w)$. Hence, parameters θ of the policy and

w of the Q-function are updated as follow:

$$\begin{aligned} a_{t+1} &= \pi(s_{t+1}, \theta_t) \\ y_t &= R_{t+1} + \gamma \hat{q}(s_{t+1}, a_{t+1}, w_t) \\ w_{t+1} &= w_t + \alpha[y_t - \hat{q}(s_t, a_t, w_t)] \nabla_w \hat{q}(s_t, a_t, w_t) \\ \theta_{t+1} &= \theta_t + \alpha[y_t - \hat{q}(s_t, a_t, w_t)] \nabla_\theta \pi(s_t, \theta_t) \nabla_{\pi(.)} \hat{q}(s_t, a_t, w_t) \end{aligned} \tag{4.15}$$

where the chain rule is used to differentiate the Q-function with respect to the policy network parameters.

4.4.1 DRL-based Actor-Critic Methods

In the deep versions of the AC methods, both actor and critic are typically represented by CNN. The *Asynchronous Advantage Actor-Critic (A3C)* algorithm [92] maintains a policy $\pi(a_t, s_t, \theta)$ and a value function $v(s_t, w)$. The method performs parallel training where multiple agents, in parallel environments, independently follow and update the policy. This helps breaking correlations among samples. Furthermore, entropy of the policy is added to the objective function as regularization term. A2C is a variant of the A3C algorithm that does not take advantage of the asynchronous updating.

Deep Deterministic Policy Gradient (DDPG) [99] adapts DPG and the main ideas in DQL, including the use of experience replay memory and target network, to the learning of a policy for the continuous action domain. To improve exploration during learning, noise is added to the deterministic policy. DDPG is sensitive to the hyper-parameters choice and can led to overestimated Q-values. *Twin Delayed DDPG (TD3)* [100] improves over DDPG by including double Q-learning, and delaying policy updates with respect to the Q-function update (the Q-function is updated more frequently than the policy).

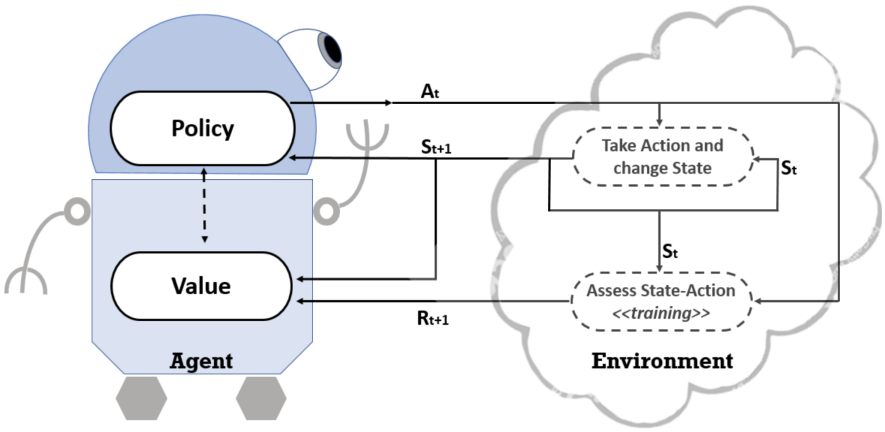


Figure 4.1: In RL, an agent interacts with the environment. Information about the environment at time t is encoded in the state S_t. Based on the learned policy, the agent takes the action A_t, which will have an impact on the environment and will determine the new state S_{t+1}. During training, the agent receives a reward R_{t+1} depending on A_t and S_t that modifies the agent's value function and the policy.

In *Trust Region Policy Optimization (TRPO)* [101], 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. Such constraint helps to avoid the maximization problem to move towards excessively large policy update. In contrast, in *Proximal Policy Optimization (PPO)* [102], 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.

4.5 Modeling Visual Tracking by DRL

The goal of reinforcement learning (RL) is to obtain a policy to maximise the expected rewards by taking sequential actions interacting with the environment from interactions.

In a typical RL setting, shown in Fig. 4.1, there is an agent that interacts with the environment at discrete time steps, $t = 0, 1, 2, ...$ At each time t, the agent decides on the action to perform, A_t based the state S_t, that encoding the agent's perceived environment (whatever information is available to the agent about the environment). R_{t+1} measuring the goodness of the taken decision in state S_t. The state transition T brings a new observation state S_{t+1} depending by the state S_t taken the action A_t.

Fig. 4.2, shows all possible ways in which RL can be applied to visual tracking. The tracker can be a RL-agent predicting location of the target in a frame or the RL can be used to solve other tasks, selecting some hyper-parameters, select one correlation filter on a set, or in some case selecting the target appereance.

Normally, is spread to use as the state an image patch called "search region", extracted based on the hypothetical target bounding box, i.e. the estimated target location at the previous frame, and taking as action set the possible changes of the bounding box coordinates through horizontal/vertical shifts, and scale adjustment. These action can be discrete or continuous, on the first case 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, one the second case at each the 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.

As we claimed before, the RL application in the visual tracking

can be different, it's also possible to use RL agent to take account the target appereance changes or decide the tracking strategy, in the first case the RL action can be select the the target template to feed in input to the model or the correlation filter to use from a pool of pre-existing filters, or of a specific tracker from an ensemble, in the second case the agent can decide to track or re-detect the target or to enlarge the search area to improve the 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 on.

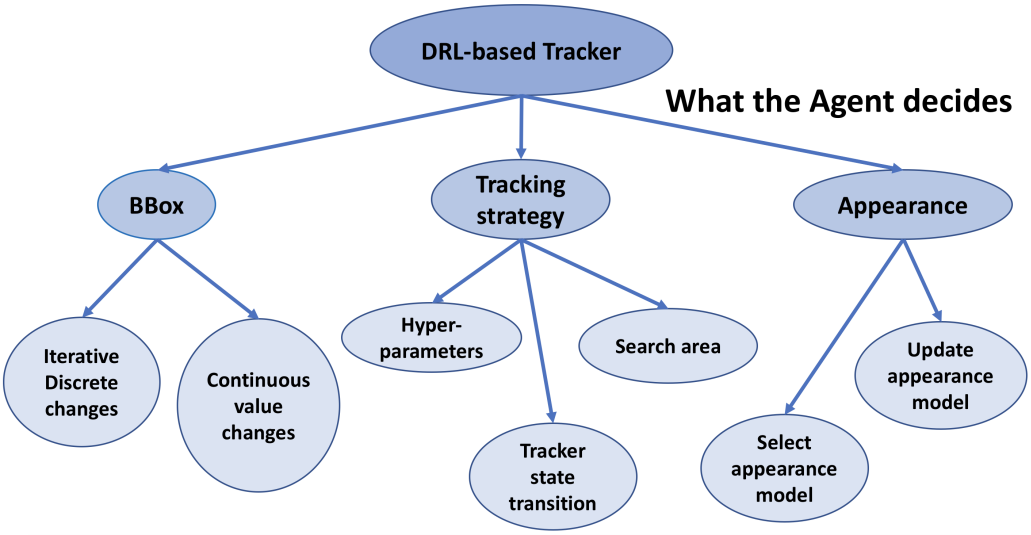


Figure 4.2: The agent can take decisions about the target bounding box, the tracking strategy to adopt, or the target appearance model. The bounding box can be refined by iterative discrete changes or by continuous values changes. Agents can also select the tracking hyper-parameters, the search region 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.

Regardless the decision of the agent, the features are extracted using CNNs, generally the output of the CNN represents the policy of the agent. The reward normally can be defined in two ways, the intersection-over-union of the estimated bounding box and the true bounding box or the distance between the coordinates of the estimated bounding box and the true bounding box, hence, the bounding box annotation is always necessary to compute the reward.

In the following section we summarize some DRL based visual tracking method categorising them in three sub section depending of the decision of the agent.

4.6 DRL-based Visual Tracking

First attempts to use RL for visual tracking have focused on feature/appearance model selection [103, 104], PTZ camera parameter estimation [105], tracking strategy selection [106, 107] or tracking hyper-parameters estimation [108]. They have mostly adopted the Q-learning algorithm and have greatly suffered from the difficulty of properly representing the environment state.

An attempt to use IRL is proposed in [109], 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 [110], where the agent also decides whether updating or no the discriminative Correlation Filter (CF) used to detect the target. Hierarchical discriminative CF (HCF) [63] 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. 4.2. First, we discuss works that estimate the target bounding box through discrete or continuous-value actions; Then we consider methods that affect the tracking strategy by acting on the agent's state transition during tracking, tracking hyper-parameters or search area extension; Finally, we describe methods operating choices on the selection or updating of the target appearance model. When summarizing such works, we also refer to the categorization in Figure 4.3 where the same works are analysed 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.

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 where the instantaneously optimal action to select is derived based on the ground-truth available with the training data.

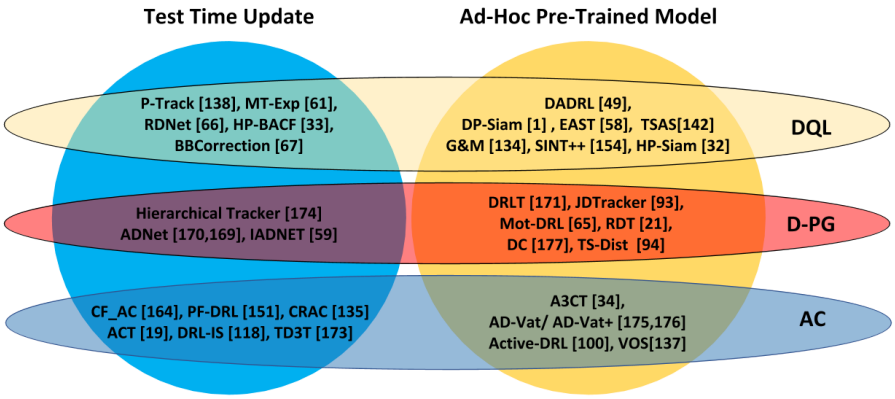


Figure 4.3: Categorization of DRL-based visual tracking approaches. Methods can take advantage of some form of test time adaptation or can be ad-hoc pre-trained (blue and yellow circles respectively). The methods differ for the adopted RL framework: Deep Q-Learning (DQL), Deep policy gradient (D-PG) and Deep Actor-Critic (AC) methods.

4.6.1 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 in the bounding box coordinates.

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). Iterative actions are applied to the bounding box on the same frame until a stop action is not taken. Methods differ in the number of pixels or proportionality factors adopted to define the

agent's actions.

The methods in [84, 111, 112, 113] adopts DQL [20] to train the agent, while [78, 114, 115, 116] use the REINFORCE algorithm [97] (D-PG). The methods take all advantage from test-time updates of the model with the only exception of [113, 112, 116] that use an ad-hoc pre-trained tracking models.

In [84], the method, which we name *BBCorrection*, aims at correcting the target bounding box by exploiting features extracted by a CNN. The method adapts the work in [23] for object detection to visual tracking. Given an initial detection of the target, a pre-trained deep network (derived from the VGG-16 model [45]) 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; Hence, a large number of iterations are required to converge to the target location. During tracking, whenever the agent get stacked into a local optimum (i.e., the agent selects pairs of actions cancelling each other for 10 steps continuously), the bounding box is randomly moved.

The work in[111], *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 [95]. 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 to capture most of the target information. The movement agent is rewarded based on the improvement of the Euclidean distance before and after the selected action is performed.

The tracking success might depend on the initial bounding box. At test time, the search of the target starts from three different initial bounding boxes, and the average target bounding box is then used. If none of the three searches is successful, Kalman filter is used to locate the target. Differently than other approaches using Siamese network, where inputs are cropped from different frames, RDNet takes inputs from the current frame. The other interesting aspect is about the training strategy. The policy is represented by the last FC layer and the model is end-to-end off-line trained. Every new video, the policy is re-initialized to uniform parameters. At test time, the convolutional layer parameters are kept fixed, the policy is re-initialized and retrained by DRL on the first frame.

The method in [113], *DADRL*, takes advantage of the cooperation of two different agents for deformable face tracking: the tracking agent is implemented by a CNN and aims at generating the face bounding box by estimating at each step how to move the current bounding box (horizontal/vertical shifts and scale changes), the alignment agent estimates the face landmarks and is based on the hourglass network [117] for pose estimation. It takes the decision of continuing to adjust the face region or stopping. The two agents interact over time by exchanging features. A LSTM storing the effects on landmark detection of the past tracking agent's actions is used in the model. The landmark detection accuracy is used as reward. The network is pre-trained in a supervised way, by providing the correct actions derived from the ground-truth.

The work *TSAS* [112] 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 by considering as label the

action that allows achieving a higher IoU. The second network models a Q-function and is trained by RL to assess the action value. The search of the target is performed in three steps. First, multiple candidates surrounding the last known target location are processed and shifted based on the first network output by considering a stride D. Later on, the largest confident bounding boxes are reprocessed by taking half the stride. The process is repeated other two times and it terminates with the selection of the highest confident locations. Later on, a regressor is used to refine the estimation. The regressor is trained independently of the model. During tracking, both the model and the regressor are fine-tuned based on the annotation in the first frame.

In [78, 118], the target bounding box is estimated by repetitive actions selected by means of an action-decision network (*ADNet*). The state is represented by the cropped target image and the history of the last 10 selected actions represented by one-hot encoding. The target image is fed to the policy network (a CNN), which outputs the action probability distribution. 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, 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* [114]. At training time, the action is sampled from the estimated action distribution to ensure higher exploration. Instead of rewarding the agent only when the terminal action is selected, different rewards are given on a frame basis. To improve robustness of the tracker, multi-domain learning in [57] is also included such that, during the supervised learning stage, a shared feature representation is learned. 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 [115] proposes an *Hierarchical Tracker* composed of a ad-hoc pre-trained motion model (CNN and LSTM), and an appearance model based on HCF [63] that is updated at test time. The motion model takes in input the cropped target image representing the state s_t. Similarly to ADNet [78], the motion model provides the action probabilities. Furthermore, the network also provides a confidence score. 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 new target location. Failures are detected by analyzing the correlation values around the target location. If a failure is detected, a number of surrounding samples are fed in input to the motion model, and the one having the highest confidence score is chosen as predicted target location. In this work, only the HCF is adapted at test-time. In their ablation study, authors show that the RL-based motion model contributes to the tracking performance.

Instead, in [116], a set of single-target trackers are used (*Mot-DRL*) to iterativelly adjust the target bounding box. The state is represented by a crop image and the last 10 selected actions.

Continuous-Value Bounding Box Adjustments

The works in [119, 120, 121, 122] adopt the Actor-Critic framework to handle with continuous-value action prediction, while [123, 124] adopt the REINFORCE algorithm [97]. Among the former works, only the methods in [119, 120, 122] take advantage of test-time adaptation of the model.

In the Actor-Critic tracker (ACT) [119] and in TD3T [120], 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 takes in input the state and the selected action and provides the Q-values and a verification score representing the decision reliability. In *ACT* [119], whenever the verification score provided by the critic is below zero, the critic is used on samples around the previous target location to select the one with the highest score. 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 [99]. During training, with an annealed probability ϵ, the agent can take an action based on an expert guidance (for example, the ground-truth annotation). In *TD3T* [120], the policy is learned by means of the TD3 algorithm [100], 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 [119].

In [122], the particle filter framework is used to perform tracking (*PF-DRL*). The work proposes a motion-aware RL agent that guides the particle sampling. In this sense, the work revisits the MDNet model [57] which tracks adaptively the target by evaluating a number of samples drawn around the past target location. Such samples can be regarded as particles. The proposed motion-aware network takes in input the cropped target image and the action history, and outputs a continuous action representing pa-

rameters 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 [99]. At training time, supervised learning is adopted to consider the annotation in the first frame of each sequence and, similarly to [119], an expert guidance is required. At test time, the model is fine-tuned in a supervised way by using the annotation in the first frame. During test, adaptation of the model is done as in MD-Net [57].

Also *A3CT* in [121] proposes to train the RL agent under the guidance of an expert tracker. The network is composed of a Siamese network and a LSTM. The output of the LSTM is used to fed the actor and the critic, which estimates the V-function. Actions are continuous values used to update the target bounding box. The expert tracker (SiamFC [68]) performs tracking by generating a sequence of states (estimated target locations) and actions (changes to the target locations). The RL agent is trained based on the *imitation loss*, which accumulates the L1 norm between the RL agent and the expert actions whenever the agent performed worse than the expert. In practice, the agent learns to behave better that 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 than the ground-truth. However, a form of supervision is provided by the expert, which has been trained in a supervised way.

The above works are interesting in that the training procedure mixes both supervised and reinforcement learning. This implicitly suggests that, with a huge continuous action space, RL alone might fail. The works also highlight the importance of having

balanced sample sets where the balancing refers to the presence of both positive and negative rewards.

In [123], 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 [124] 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 sub-spaces of varying dimensions. The jump allows to move from a subspace to another. In JDTracker, the state decomposes into a discrete sub-space encoding the target visibility (target visible, occluded or invisible), and a continuous sub-space encoding the target location. The method learns two sub-policies (two CNNs), the first designed on the discrete sub-space and the other on the continuous sub-space. Each CNN takes in input a cropped target image, the current target location, the past action sequence, and 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 [97] with accumulated gra-

dients to learn the parameters of the two policy networks. At the early stage of training, supervised heuristics are used to initialize the model. Visibility is defined based on the detector confidence, and continuous actions are derived from the ground-truth.

Camera Parameters Adjustments

There are also works [125, 126] 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 [127, 125], 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 has been 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*) [126, 128], 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 adversarial learning strategy adopted, 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 is non zero otherwise in order to penalize the target agent to run too far from the tracker estimation. The target agent knows what the tracker is observing and what actions is deciding, and is trained with the further goal

of estimating the tracker reward.

4.6.2 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, which may imply to re-initialize, update or delete the tracker.

Tracking Hyper-Parameters Adjustments

In [129] (*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 process and dynamically changing such hyper-parameters at a frame level may led to great improvement of the tracking results. Since ground-truth values of hyper-parameters is not available, the problem of estimating continuous values of the hyper-parameters is modeled as a MDP. In particular, a policy network estimates the hyper-parameter values, and another network assesses the Q-function value. Learning is accelerated by using NAF [96]. The work is extended in [130] (*HP-BACF*) where the proposed network is combined with a real-time correlation filter-based tracker [131]. Training of the whole model is done in three subsequent steps: first, a supervised training of the action network (which estimates the hyper-parameters) is performed by using the default values of the hyper-parameters; then, the action network is frozen and Value and Q networks are trained. Finally, the whole model is fine-tuned to improve the tracking results.

An hybrid approach that allows to estimate both the new target location and the tracking hyper-parameters is proposed in [132],

DP-Siam. This model is composed of three networks. A Siamese network takes in input the target template and the search area image and produces an heatmap representing the per-pixel likelihood of finding the target in the search area. The Agent Network approximates a continuous-value policy; it takes in input the search area image and the heatmap and estimates the action, namely the new target position in the search area. Such 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 latter FC layer of the Q-network is augmented with the policy network output. The Q-network 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. During tracking, 20 candidates are considered based on the heatmap returned by the Siamese network. The candidate with the highest score is selected as target. The choice of considering many candidates makes unclear the contribution of the Agent Network during the tracking process.

Tracker State Transition

Methods in [21, 133] 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 [20], the latter is based on the AC-framework. Both the methods take advantage of model updates at test-time.

Since actions depends on the full history, in [21], *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) [134] 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. One of the positive aspects of the approach is that the training can be done with limited annotations. On the other hand, it is reported that a large number of videos (of unknown length) need to be used to make the model converge.

In the method [133], DRL with iterative shift (*DRL-IS*), three network are used: the prediction network, the actor and the critic. These networks share all convolutional layers. Appearance features of the target are explicitly maintained and online updated. The prediction network adjusts the target location in an iterative way (shifting and scaling of the target bounding box). The actor network takes in input the cropped target image, the target location in the previous frame and the target appearance features. The actor makes decisions on the tracking status, whether or not to update the target representation and the prediction network, or even restart tracking. The target appearance representation is updated by the output of one of the convolutional layers. The critic takes in input the action distribution estimated by the actor, the values estimated by the prediction network and the target representation at the current frame and at the previous one. It returns a value assessing the state-action pair. If the actor select the action restart, a set of patches is sampled and assessed by the critic; the one yielding to the highest Q-value is chosen as new target location. As for the training strategy, 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.

The model is updated at test-time by using DRL.

Finally, we also include in this category the work in [135], which proposes a decision controller (*DC*) implemented by a Siamese network to choose between two trackers' results. The controller, trained by REINFORCE [97], 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 [68]) or to detect it (by SSD [136]), namely to re-initialize the target tracker whenever drifting occurs.

Search Area Adjustments

While works in [137, 138] aim at adjusting the search area through discrete shift/scale actions, the work in [139] proposes a data augmentation techniques where the goal is determing the area to occlude to generate hard samples. The methods in [137, 139] adopt the DQL framework [20], while CRAC uses the actor-critic one coupled with GAN [55]. This latter method takes advantage of model updating at test-time.

The work in [137], *G&M*, proposes to estimate the optimal search area to fed into a Siamese network-based tracker. Two modules are available: the matching and the guessing modules. The first one is a Siamese network that, given in input a template image and a search area, provides the heatmap of the new target location by cross-correlation. The guessing module has the goal of providing a rough estimation of the target location by using the current state image, the coordinates of the center of the target in the past five frames, and the action history. 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. In combination with other pre-trained trackers (such as SiamFC [68]), the strategy of estimating a rough target location to better center the search area has proved to improve the tracking results.

An interesting work (*CRAC*) designed for unsupervised vehicle tracking in drone videos is proposed in [138]. The actor takes in input observations from an image and the action history. It determines the window size of the search area (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 new generated images. The actor-critic model is trained by A3C [92] on ground-view dataset. The knowledge is then transferred to adapt for the drone-view videos by using two GANs [55]. One GAN (T-GAN) generates the drone views by preserving the local discriminative features; tracking is performed by a pre-trained tracker (such as MDNet [57]). Another GAN generates attention maps. The latter GAN is re-trained at test time. The reward for the agent is computed based on the tracking score provided by T-GAN.

Differently than the above approaches, *SINT++* in [139] couples reinforcement and adversarial learning to augment the training set with hard positive samples and improve the robustness of the SINT algorithm [67]. 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 to add occlusions on the target objects by using image patches extracted from the background. The latter module is trained by DQL [20] and the agent's actions entail the movements of the patch over the target. Despite interesting, it is unclear whether the resulting occluded images are enough realis-

tic, but experimental results show the added occlusions improve tracking results.

4.6.3 Deciding Appearance Model Selection/Update

The agent can also take 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 the appearance model or no, or can decide which tracking results to consider in case of a pool of expert trackers is available. All these methods use discrete actions. Whenever the appearance model is represented in terms of CF [41], then the methods take advantage of the appearance model update at test time.

The EArly Stopping Tracker (*EAST*) [140], aims at speeding up deep Siamese trackers in an adaptive way by choosing the optimal layer outputs to use for tracking. It is based on the intuition that complex scenario may require deeper features, while simpler scenario may need the first layer output. The agent decides if using the representation of the current layer or moving to the next one. The state represents the average score map obtained by cross-correlating the feature maps at each layer preceding the selected one and the action history. Agent's discrete actions include various scaling operations of the target bounding box whose center is found based on the cross-correlation map. Several candidate bounding boxes are considered around the estimated region from previous frame. The action set include also terminal/non-terminal actions. If a non-terminal action is selected, the process is re-iterated by considering the next layer along the forward-pass. The maximal number of iterations depends on the network depth, which makes the iteration number a-priori limited by the archi-

tecture structure. The method is based on DQL [20]. Somehow related is the work in [141] (*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 framework is composed of several modules. The first one 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. This is not the first work using RL to define the network structure, but is the only one we have found that does it to build a tracker. Learning is based on policy gradient methods.

The works in [142, 143] adopt CF as target appearance models, the work in [144] is concerned with the choice of the better heatmap to use in tracking, and [145] represents the target appearance in terms of template images.

The method in [143] (*MT-Exp*) models the multi-tracker tracking problem as a decision-making task where 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-tained 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 [142] (*CF-AC*), it is noted that older CFs may yield to 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 [143], where the state is the response map of a single CF and multiple evaluation are required. In CF-AC, the

critic is used to assess the policy decision. The policy network is trained by PPO [102]. The pool of CFs include the initial (never updated) CF, various updated CFs, and an accumulated (always updated) CF.

In *RDT* [144], 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 maintaining an updated an effective appearance model represented in terms of an ensemble of templates. Of course, the performance of such strategy strongly depends on the adopted Siamese network.

The method in [145] (*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 [146] or Mask R-CNN [59], the AC-based agent has to decide whether to update or no 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. The agent takes in input an image where only the pixels within the object bounding box are left while the others are blackened, and an image where only the pixels within the segmentation are unchanged while the others are blackened. Deep features for the two images are extracted and concatenated. The reward is defined based on the segmentation results. The work demonstrates empirically that RL-based model greatly outperforms the SL-based one.

Chapter 5

Accelerating learning of deep models

The tracking models trained offline, whether through a supervised approach or reinforcement learning, often necessitate online tuning to achieve optimal performance. Tracking tasks often involve dynamic and changing environments where the appearance and behavior of the tracked object may vary over time. Offline trained models might not be able to adapt effectively to these changes without online tuning. By fine-tuning the model online, it can better adapt to the specific characteristics of the current environment, leading to improved tracking performance. The duration of this tuning process is contingent on the number of layers requiring training for the new target audience. However, we propose a pioneering approach in this thesis that expedites the training of CNNs while simultaneously preserving their high performance levels. However, it is important to note that our new approach has not been specifically tested on visual tracking datasets. Instead, we have evaluated its performance on various other types of datasets to showcase its ability to generalize beyond the training data. By conducting experiments on diverse datasets, we aim

to demonstrate the versatility and robustness of our approach in different problem domains. Although direct evaluation on visual tracking datasets is pending, the results obtained from these alternative datasets provide valuable insights into the potential effectiveness of our approach across various applications.

5.1 Introduction

The use of deep learning models is increasing over time, since they perform well in various areas, such as computer vision, natural language processing, and speech recognition. In computer vision, the most used deep learning models are the Convolutional Neural Networks (CNNs), which consist of very deep models with many different layers processing the input by mapping it to the expected output [147].

The training time of CNNs is sometimes very long and could last hours, days or even weeks depending on the task, the size of the dataset and the available hardware. Nowadays, the trend is to increase the depth and size of architectures [148]; This usually leads to better performance despite a heavy computational cost.

Although modern deep CNNs are composed of a variety of layer types, convolutional layers are the building blocks of CNNs, and contribute greatly to the overall computational load of the network. In recent times, the deep learning community has focused on how to improve the efficiency of CNNs while saving time, computational costs, and energy without compromising the accuracy of the results [148].

We can divide the methods that improve the efficiency of CNN into two categories[148], depending on the stage in which they seek to achieve better efficiency:

- methods for inference efficiency: this category includes methods that compress the network during the training phase to use a more compact model during inference [149, 150, 151, 152, 153];
- methods for training efficiency: fall into this category methods that improve the efficiency of model training only, for example by freezing some layers during training but continuing to use the entire model during the inference phase [154, 155, 156].

At first glance, one might think that methods for inference efficiency are the most relevant because a model is trained once and used to infer multiple times. However, there are applications where new data arrives sequentially and the model has to be adapted and retrained using this new data. An example is visual tracking where the model is often used to represent the target appearance. Since appearance changes over time, it is required that the model adapt to such changes. In that case, slow retraining of the network may affect the real-time capabilities of the tracker. Also, recommendation systems or large language models (LLM) must be regularly retrained, and the training can last several weeks

Motivated by the above considerations, in this thesis we focus only on training efficiency by proposing to gradually compress the neural network during the training phase by dropping some convolutional layers and using the entire model (with all its layers) during the inference phase. We analyze the method on some of the most popular CNNs such as VGG [157] and ResNet [158] by observing their learning behaviour through gradient monitoring.

The two main steps in neural network training are forward and backward propagation; both steps have a high computational

cost, which increases according to the complexity of the network. In general, forward propagation is the process of passing data through the network from one layer to another. In each layer, the input data is processed taking into account the weight matrix of the layer itself and a suitable activation function to produce the layer output. The output of each layer becomes the input of the next layer repeating this process until the output of the final layer is produced. The back-propagation algorithm [147] is used to train a neural network by updating its weights to minimize the error between the predicted and expected output. The algorithm computes the gradient of the loss function with respect to each weight via the chain rule, starting at the output layer and propagating the gradient backwards through the network from one layer to another.

The magnitude of the gradient permits to understand how the parameters of the model vary during training: the closer it is an optimal point, the lower the value of the gradient. The variation of the gradient magnitude across the epochs provides a learning curve that can be calculated separately for each layer of the network. We have empirically found similarities among these learning curves when analyzing different CNNs and, in particular, we have found that the layers' parameters are learned sequentially from the first to the last layer. Thus, in this Thesis we propose to sequentially eliminate the convolutional layers during the training phase. Layers to drop are selected based on a metric related to the layer gradient. The adopted metric gives us an indication of which layer has stopped learning and can, therefore, be temporally eliminated from the model. However, when we drop a layer, we have to find a way to feed the next layer to continue its training. This is done by feeding the remaining model (the one

without the deleted layers) with the feature maps produced from the last dropped layer. We stress here that dropping the layers has a double consequence: the weights of the removed layers are no longer modified and, moreover, a compressed model is trained in the next epochs. During the test, the entire model is used and each layer will have as weights those obtained when the layer was selected for dropping. This method shows an huge acceleration of the training process. We can summarize the main contributions of this thesis as it follows:

- A new method to improve the efficiency of training CNNs by dropping layers based on the metric derived from gradients.
- A significant speed-up of the training process of CNNs compared to state-of-art methods, given by the possibility of processing directly feature maps extracted from dropped layers and calculating the back-propagation only for the remaining layers. We empirically show that, by using our technique, the training time of a CNN is more than halved.

5.2 Related Work

In recent years, many works have focused on how to reduce the parameter size of deep learning models. This is quite a challenge considering the energy impact of training large networks. The goal of compression techniques is to achieve a more efficient representation in a neural network, improving the generalization of the model if the model is overly parameterized. Model optimization is performed with respect to model size or training time in exchange for as little accuracy loss as possible. A popular method of reducing the complexity of neural networks is to permanently

remove neurons, filters, or layers for training. This technique is called pruning. Pruning can be performed based on different aspects of neural networks, for example it may be possible to remove parameters with low saliency scores from a pre-trained network. These techniques are often referred to as Optimal Brain Damage or Optimal Brain Surgeon, and were first introduced by LeCun et al. [159] and Hassibi et al. [160], respectively. The goal of this process is to minimize the impact of compressing the network on its performance, as measured by its validation loss. OBD approximates the saliency score by using a second-derivative of the parameters ($\frac{\partial^2 L}{\partial w_i^2}$), where L is the loss function, and w_i is the candidate parameter for removal. In [161], it is suggested that a greedy method be used to determine the minimum set of neurons needed to minimize the reconstruction loss, but this approach has a high computational cost. Other methods that prune by "saliency" consider the magnitude of the weights [162, 163]. The previously described methods for removing sparse neurons are examples of unstructured pruning methods. On the other hand, structural pruning aims at reducing the number of filters as in [164] where the L_1-norm is used to select the filters to be removed without affecting the accuracy of the classification. A similar idea is presented in [165] where the feature map channels that do not contribute to the result are removed.

In addition to pruning filters and channels, there are also methods of pruning entire layers [166, 167, 168]. Using different criteria, the selected layers from the network are removed to obtain a compact model. These methods claim that the model obtained from layer pruning require less inference time and memory usage at runtime with similar accuracy values than the model obtained from filter pruning methods. The work by Chen et al. in [166]

uses independently trained linear classifiers per layer to rank their importance. After ranking, they remove less important layers and fine-tune the remaining model. However, their method requires additional rank training. All the previously described methods fall into the category of inference efficiency methods because they produce compressed model to be used at inference time.

In other efficiency training approaches like in [169], the gradient computation through the chain rule is stopped. To speed-up training and increase accuracy in very deep networks, AFNet [169] investigates a different use of back-propagation. Only a subset of layers are trained while the others are frozen. Frozen layers weights do not need to be updated during back-propagation. In [154], a metric F, named Freezing Rate, is defined as a function of the gradient values of the set of weights of a layer. This metric is used in [154] to decide which layer should be frozen during the training. The frozen layers must be subsequent to not break the layers chain, and therefore the freezing of the layers start from the first layer and advances to the subsequent ones. During training, the layers are not removed and therefore the computational advantage of the method concerns only limiting the calculation of the gradient and the modification of the weights of the frozen layers. Our approach is inspired by [154], in the sense that we adopt the same metric F. However, unlike the method in [154], our method consists in performing a layer elimination (drop) during the training phase. Once a layer is removed, the remaining model is fed through the feature maps produced by the last deleted layer. This speeds up both forward propagation, since there is no need to recalculate the feature maps of the deleted layers, and backward propagation, since the gradient is not calculated in the dropped layers and there is no need to update their weights. Our experi-

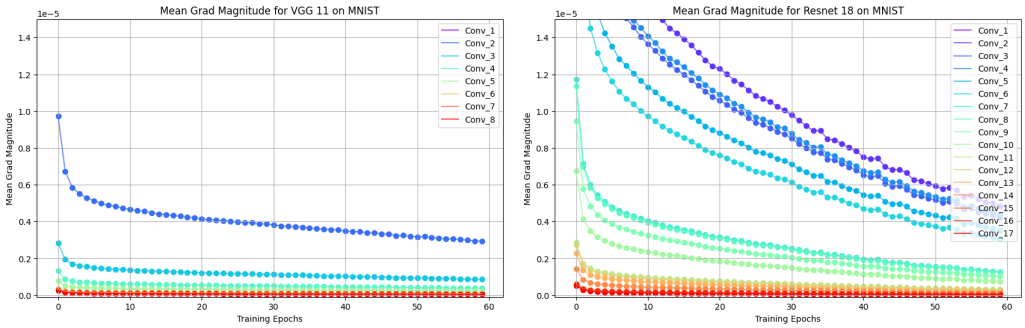


Figure 5.1: The two graphs represent the average absolute partial derivative value of each convolutional layer in the VGG-11 (left) and ResNet-18 (right) models on the MNIST dataset. The average absolute partial derivative value is indicated on the y-axis and the epoch on the x-axis. Each curve represents a different layer with colors from purple, for layers closest to the input, to red, for layers closest to the output. The figure suggests that weights in the first layers undergo much higher changes than the weights in the layers closest to the output, especially at the beginning of the training. Thus, partial derivative values can help understanding if the weights of a layer are still changing or not. However, using directly the average absolute partial derivative values could be misleading since the weights may have small magnitude but still change.

ments demonstrate how this approach increases training efficiency by gradually reducing the number of FLOPs over the epochs. At the same time our method is different from classic pruning methods, as our method does not permanently remove the layers by performing a network compression but only temporarily drops them during the training phase. In fact, in the test phase, the model will contain all the layers of the one originally created.

5.3 Dropping Layers for Training Efficiency

Our method is described in Algorithm 2 and its main steps are:

- Compute the “Layer Importance metric" based on the gradi-

ent of the layer's weights;

- Apply the "Fast Learning" algorithm, the core of our method. The algorithm consists of steps to: (1) select the layers to be dropped, (2) split the network into a "tail", composed of the dropped layers, and a "head", composed of the layers to still train, (3) compute and store output feature maps from the tail, (4) train the head by using output feature maps from the tail.

5.3.1 Layer importance

Our layer dropping method is based on the observation of the loss function gradients ∇g. Generally, the gradient values can be interpreted as the rate of change of the weights. The sign of the partial derivatives represents instead the inverse direction in which weights should be changed to reach a minimum.

Given a neural network with layers $L = \{l_0, l_1, ..., l_L\}$, the average absolute partial derivative value $\overline{g_l^{(k)}}$ corresponding to the weights of the l-th layer is calculated as:

$$\overline{g_l^{(k)}} = \frac{1}{N} \sum_{i=1}^{N} \sum_{j=1}^{M} \mid g_{lij}^{(k)} \mid \tag{5.1}$$

where N is the number of weights in layer l, M is the number of iterations in epoch k, and $g_{lij}^{(k)}$ is the partial derivative of the loss function with respect to the i-th weight in layer l at the j-th iteration in epoch k. Fig. 5.1 shows two graphs representing the average absolute partial derivative value $\overline{g_l^{(k)}}$ of each convolutional layer in the VGG11 and ResNet18 models. The layers closest to the input in both networks have a greater average partial derivative value than the layers closest to the output. In practice, the

figure suggests that weights in the first layers undergo much higher changes than the weights in the layers closest to the output, especially at the beginning of the training.

Thus, partial derivative values can help understanding if the weights of a layer are still changing or not. However, using directly the average absolute partial derivative values could be misleading since the weights may have small magnitude but still change.

Hence, we decided to adopt the metric proposed in [154], and define for the l-th layer the score:

$$P_l^{(k)} = 1 - \frac{\sum_{i=1}^{N} \mid \sum_{j=1}^{M} g_{lij}^{(k)} \mid}{\sum_{i=1}^{N} \sum_{j=1}^{M} \mid g_{lij}^{(k)} \mid} \tag{5.2}$$

with $0 \leq P_l^{(k)} \leq 1$, where $P_l^{(k)}$ measures the degree of changes of the weights in layer l at the k-th epoch. $P_l^{(k)}$ will be 1 if the partial derivatives cancel each other across the M iterations. In such a case, within the epoch the layer weights do not change much and, intuitively, the layer has stopped to learn. $P_l^{(k)}$ will tend to 0 if most of the partial derivatives are in the same direction across iterations. In this case, layer weights are changing during the epoch. Thus, the layer is learning something about the problem to solve. We note here that the normalization factors make the scores comparable across the layers despite the different magnitude of the weight's partial derivatives.

Under this point of view, the score $P_l^{(k)}$ is measuring the importance of the layer during training. Layers with a score close to 0 must be trained. Layer with a score approaching 1 are not learning much and probably can be dropped to speed up the model training. Unlike [154], where this score is used to freeze the layer and stop the back-propagation computation up to the l-th convolutional layer, our algorithm uses P_l to drop the l-th convolutional

layer. The feature maps produced by the last dropped layer are used as input to the remaining model.

5.3.2 Improving training efficiency

In our approach, the removal of layers from the model to improve the training efficiency must take place in sequential order. At the k-th epoch, the layers to be dropped are selected based on the importance score $P_l^{(k)}$.

Thus, our fast learning algorithm works as it follows:

1. At the end of epoch k, the metric $P_l^{(k)}$ is calculated for each layer l. The score values are then standardized:

$$P'_l = \frac{P_l - \overline{P}}{\sigma_p} \tag{5.3}$$

   where $\overline{P}$ and σ_p represent the average score over the layers and the standard deviation respectively. We omitted the apex k for simplicity.

   Ideally, we want to drop all subsequent layers for which the parameters do not change much anymore starting from the first layer of the network. The standardized scores P'_l can have positive and negative values. Positive values indicate that the layer's weights are changing less than the average (hence the layer is likely not learning much), while negative values indicate that the layer's weights are changing more than the average (hence the layer is still learning something).

   The problem of selecting the subsequent layers to drop starting from the first layer turns into the problem of finding the sub-vector of maximum sum starting from the first element

of an array. In our case, the array represents the list of scores P'_l with $l \in L$.

Let us assume that the current layers in the model are $L = \{l_z, l_{z+1}, \ldots, l_L\}$. Candidate layers to drop are $l_z ... l_{n^*}$ with n^* computed as:

$$n^* = \min_t \{t \in [z, \ldots, L-1] : P'_{l_t} > 0 \wedge P'_{l_{t+1}} < 0\}. \quad (5.4)$$

2. As soon the candidate layers to drop $l_z ... l_{n^*}$ are found, to avoid dropping them too early, we estimate the median M_c of the scores P'_{l_t} with $l_t \in l_z ... l_{n^*}$ (namely the scores of the candidate layers to drop), and compare it with the median M_d of the scores P'_{l_t} with $l_t \in l_0 ... l_{z-1}$ (namely the scores of the layers dropped in previous iterations and estimated when the decision of dropping the layers was taken). We perform layer dropping if $M_c \geq M_d$. In this way, we limit the effects that early layer dropping may have on the network accuracy value.

   Once the layers to drop are identified, the network is split into two parts: the "tail", composed of the layers in the network up to l_{n^*}, and the "head", composed of the layers from l_{n^*+1} to the network output.

3. In epoch $k+1$, the tail is used to extract feature maps. These feature maps are stored on a memory, such as a disk, and are also used to feed the head to continue its training.

4. In epoch $k+2$, the stored feature maps are retrieved from the memory and used to train the head.

These 4 steps are within an iterative procedure repeated until the maximum number of epochs is reached or the convolutional layers are exhausted, namely the head does not have any layer.

Our approach differs from the one in [154]. In the latter approach, layers with a high score are not "physically" removed from the network but their weights are not trained. The main limitation of the approach in [154] is that, during forward propagation, the data must be processed at each iteration even by layers for which the weights are not updated. Our approach overcomes this limitation by removing layers in order starting with the first. We have experimentally demonstrated the advantages of this approach in significantly reducing the computational cost of the training process.

Furthermore, in [154] layers to exclude from the training are selected after a prefixed number of epochs based on the vector $f = [f_1, f_2, ..., f_n]$. Each value f_i indicates the number of epochs between one freezing and another at a specific learning rate. Hyper-parameters in f are empirically defined and change over the adopted datasets. In our approach, the decision to drop a layer is fully automatic. After each epoch, the method analyzes the scores $P_l^{(k)}$ to detect candidate layer to be dropped and, as already described, the decision to remove the layers or not depends also on the median of the estimated scores.

5.4 Fast-Training Algorithm

Our approach is described in Algorithm 2. It starts with a few warm-up epochs e_1 where the *model* is trained to move from the initial random weights. After this warm-up, the weights of *model* are copied to the *head* model, which is initially equal to *model*, while the *tail* model is empty. From now on, only the *head* model is trained. The *tail* model stores the dropped layers and is used to estimate the features maps needed to feed the *head* model.

Each time the *head* model is trained, the corresponding weights in the *model* are updated accordingly. In practice, *model* always contain all layers, whose weights are iteratively updated based on the weights learned by the *head* model. The $save\_features$ flag is used to indicate whether the *tail* model should be used to estimate feature maps using the dropped layers. $Data$ stores the data for training the model. Initially, $Data$ stores the training images. When layers are dropped, $Data$ stores the features maps produced by the dropped layers, i.e. the *tail* model. At each iteration, the layer importance P_l' is recomputed only for each layer of the *head* model as described in Equations 5.2 and 5.3.

Then, n^* is computed based on Eq. 5.4. Layer dropping is performed if it is found a maximum-sum sub-sequence of scores P_l' starting from the first layer of the *head* that includes at least one layer and the median value of the scores in the found sub-sequence is greater than the median score of the previously dropped layers. The scores of the dropped layer are not recomputed every time but stored during the training process and kept updated till the layer is not dropped. The index n^* is also used to further compress the *head* model. In particular, the *tail* model stores the dropped layers, namely the first n^* layers of the *head* model. These same layers are pulled out from the *head* model, resulting in a reduced model.

The described process is iterated until only the last convolutional and dense layers remain; they continue to train till the maximal number of epochs e_2 is not reached.

We emphasize that the whole model (*model*) is tested on a validation set; this proves that removing the layers does not affect the accuracy of the original model. The model is optimized using SGD method in order to maintain a fixed learning rate over the

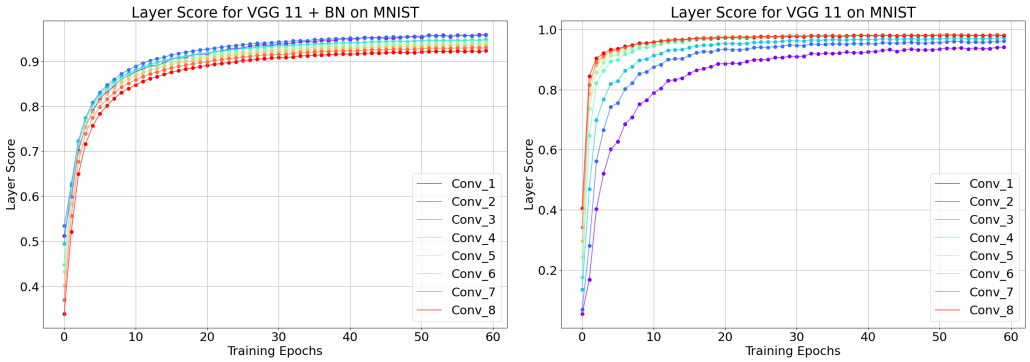


Figure 5.2: The two plots show the scores $P_l^{(k)}$ on the MNIST dataset for a VGG-11 trained with batch normalization (on the left) and without batch normalization (on the right). Adding a bath normalization reverses the order of the score curves computed for each layer across the epochs.

different iterations. This is not a limitation, and other optimizers might be used as well. Also, early stopping may be included in the algorithm to add more regularization. In our experiments, we did not use early stopping to compare different training strategies on equal terms of number of epochs.

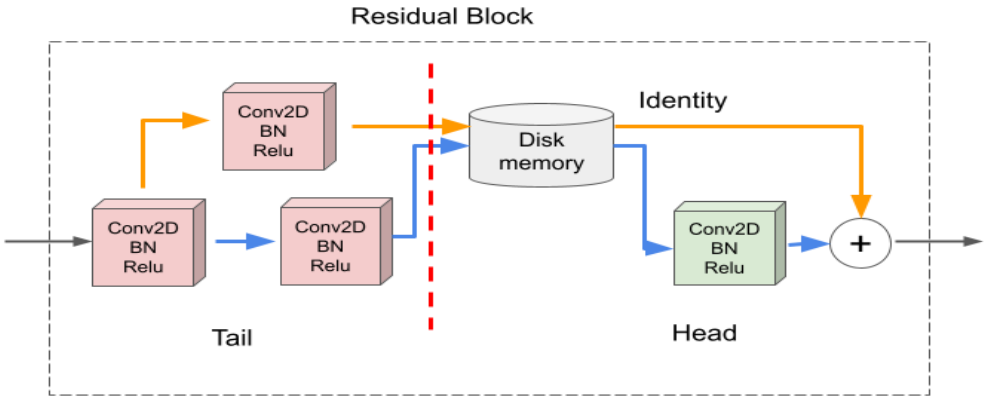


Figure 5.3: The image shows how dropping takes place in the ResNet at any residual block. The saved feature maps come from the layers in red, corresponding to the layers inside the residual block on the main track and on the skip connection. The layers on the left of the red dotted line are dropped and belong to the tail. The extracted feature maps are saved to the memory. From the next epoch, the head model (on the right of the red dotted line) will directly use the feature maps from the memory for training purposes.

Algorithm 2 Fast Learning by layer dropping

Require:
model, a model of L layers with randomly initialized parameters;
$e_1 > 0$, number of warm-up epochs;
$e_2 > 0$, number of training epochs;
L_0, number of dense layers + 1;

Ensure:
Trained *model*;

Initialize models *head* and *tail*
Initialize *Data* with the training images
$save\_features$ = False.
Train *model* for e_1 epochs on *Data*
L = *model*.layers.length()
head.layers = *model*.layers$[0 : L - 1]$
tail.layers = []
$features\_maps$ = []
Initialize P'_l with l in *head*
for $k = 0$ **to** e_2 **do**
    if $save\_features$ **then**
        $save\_features$ = False
        $features\_maps = tail(Data)$
        Train *head* on $features\_maps$
        Update *model* weights based on *head*
        store $features\_maps$ to memory
    else
        if $features\_maps != []$ **then**
            Initialize *Data* with $features\_map$
            $features\_maps$ = []
        end if
        Train *head* for 1 epoch on *Data*
        Update *model* weights based on *head*
        if $L > L_0$ **then**
            for $\forall$ conv. layer l in *head* **do**
                update P'_l (Eqs. 5.2& 5.3)
            end for
            Find n^* (Eq. 5.4) for layer dropping
            Estimate median values M_c and M_d
            if $n^* > 1 \wedge M_c \geq M_d$ **then**
                $save\_features$ = True
                tail.layers = *head*.layers$[0 : n^*]$
                head.layers.pop$(0 : n^*)$
                L = *head*.layers.length()
            end if
        end if
    end if
    validate *model* on validation set
end for

Chapter 6

Experimental Results

This Chapter discusses and summarizes:

- A concise overview of the widely adopted public benchmarks for single object tracking, providing valuable insights into their characteristics. Additionally, we summarize the recommended experimental protocols to follow when utilizing these benchmarks.
- The advantages of adopting iterative approaches for refining the target bounding-boxes are thoroughly discussed and supported by experimental results on widely used tracking benchmarks.
- The state-of-the-art results in deep reinforcement learning based tracking approaches are presented, along with modifications to existing approaches. While the superiority of DRL-based approaches over SL-based methods has not been conclusively demonstrated in this thesis, the potential benefits of such methods are explored.
- Lastly, a novel method for accelerating the training of CNNs

is presented, emphasizing its potential to improve the efficiency and effectiveness of object tracking algorithms.

6.1 Datasets and Perfomance Measurements for VOT

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.

Two main benchmarks are adopted to validate tracking algorithms: the Object Tracking Benchmark (OTB-2013) [170], and the Visual Object Tracking (VOT) dataset [171] of which several extensions/variants are provided each year, being the dataset part of an annual challange.

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 [172] 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 [171] 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 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 results on VOT-16 and VOT-18.

There are other more recent benchmarks that could have been used to assess DRL-based methods, such as LaSOT [173], TrackingNet [174], TC [175], and GOT-10k [176]. Such benchmarks would have allowed to validate the methods on long video sequences. The analysis of the reviewed papers revealed that G&M [137] is validated on TrackingNet and LaSot, A3CTD [121] is validated on LaSOT and Got-10k, DRL-IS [133] and PF-DRL [122] 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) [177], ALOV300+ [178] and VOT are often used. As reported in [67], 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 [78, 114, 115, 133, 139, 58]). Other adopted datasets for training purposes are reported in Table 6.5.

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 [170, 172, 171]. 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 as one-pass evaluation (OPE). Other methodologies (Temporal robust-

ness evaluation and Spatial Robustness Evaluation) are proposed in [170] 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 curve (AUC) of the success plot serves to rank the algorithms. In [179], 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 threshold equals to 20 pixels is used to rank the trackers.

In VOT, stochastic trackers are run 15 times on each sequence. Three measurements are computed [171]: 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 [171]. 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 to 0.5. Since the tracker is reset in case of failure, the tracking sequence is split in 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 provide attribute-based analysis (i.e. illumination changes, occlusions, etc.), only very few DRL-based papers present this analysis (namely [119, 114, 129, 130, 142]). For this reason, attribute-based comparison of the trackers cannot be done.

6.2 Visual Object Tracking: Supervised approach

The goal in this chapter is to demonstrate that iterative approaches to refine the bounding box have several drawbacks that can be overcome by allowing multiple non-conflicting refinements to the bounding box at each iteration. Therefore, we run two kinds of experiments: one to show the usefulness of our proposed approach, the other to compare our tracking strategy to state-of-the-art approaches on publicly available benchmarks. All experiments were conducted on a machine equipped with: 32 GB RAM, GPU RTX 2070 8GB RAM. Our prototype has been implemented in Python by using Tensorflow and runs at 5 fps on the GPU.

6.2.1 Single vs. Multiple Transformation Groups

We performed experiments by keeping the tracking strategy fixed and by varying the output layer of the deep model (single vs multiple transformation groups). As training of the model is important, we also test our approach by varying the model. To this purpose, we used the pre-trained parameters of the ADNet model [78], modified the last layer and compared single vs. multiple transformation groups. Finally, we tested how different training strategies of our model can affect the results. Experiments have been run on the OTB 100 benchmark [77].

All results are reported in Figure 6.1. Configurations with "NOT" directly use the initial parameters of VGG-M for the layers (conv1–4), while the fully connected layers (fc5–fc9) are initialized by random noises. Configurations with "ADNet" load the ADNet parameters.

For the above configurations, no offline training is performed and only online learning at test time is done.

Configurations with "MT" use multiple bounding box refinements, in contrast to "ST" where a single transformation is applied at each iteration. Finally, "MD" indicates that multi-domain learning is adopted to pre-train the model (offline learning), "SD" indicates a more classical training procedure where all videos are used to train all model layers.

As shown in Figure 6.1, models adopting the multiple bounding box refinements achieve higher performance than the corresponding ones with single refinements. However, the training strategy largely affects the performance of the method. Offline multi-domain learning allows achieving higher results. The method that yields to the highest performance uses ADNet parameters and the proposed refinement approach. We note that in terms of preci-

sion and success, we achieve higher results than the one published in [78] (0.88 and 0.646, respectively) the results with single refinements are almost identical, meaning that different implementation choices in the online tracking strategy may have little impact on performance.

Our offline-trained model differs from the ADNet especially because ADNet uses a reinforcement learning approach after the supervised training of the model.

6.2.2 Comparison on OTB and VOT

Table 6.1 shows the results achieved by our tracker in terms of Precision (P), Success (AUC), and frame rate (FPS). All experiments were run by using the Got10K-toolkit [180]. Compared to approaches using iterative bounding box refinements (in column Iter), such as TSAS [112] and ADNet, our model (Ours_MT_MD) achieves better/comparable results. Our ADNet-based tracker (ADNet_MT) achieves better/comparable results than those in [78, 181, 79], which also use iterative refinements.

Figure 6.2 shows some samples from three videos belonging to OTB. In the images at the first row, both our method and ADNet are unable to adapt to the actual target shape. This is because rescaling of height and width is done jointly and not separately. However, our method (red bounding-boxes) seems to center better the target. In the second row, both ADNet and our method are sensitive to large and abrupt camera motion. In the third row, where the target is among several instances of the same class (several players), ADNet drifts while our tracker is able to follow the target.

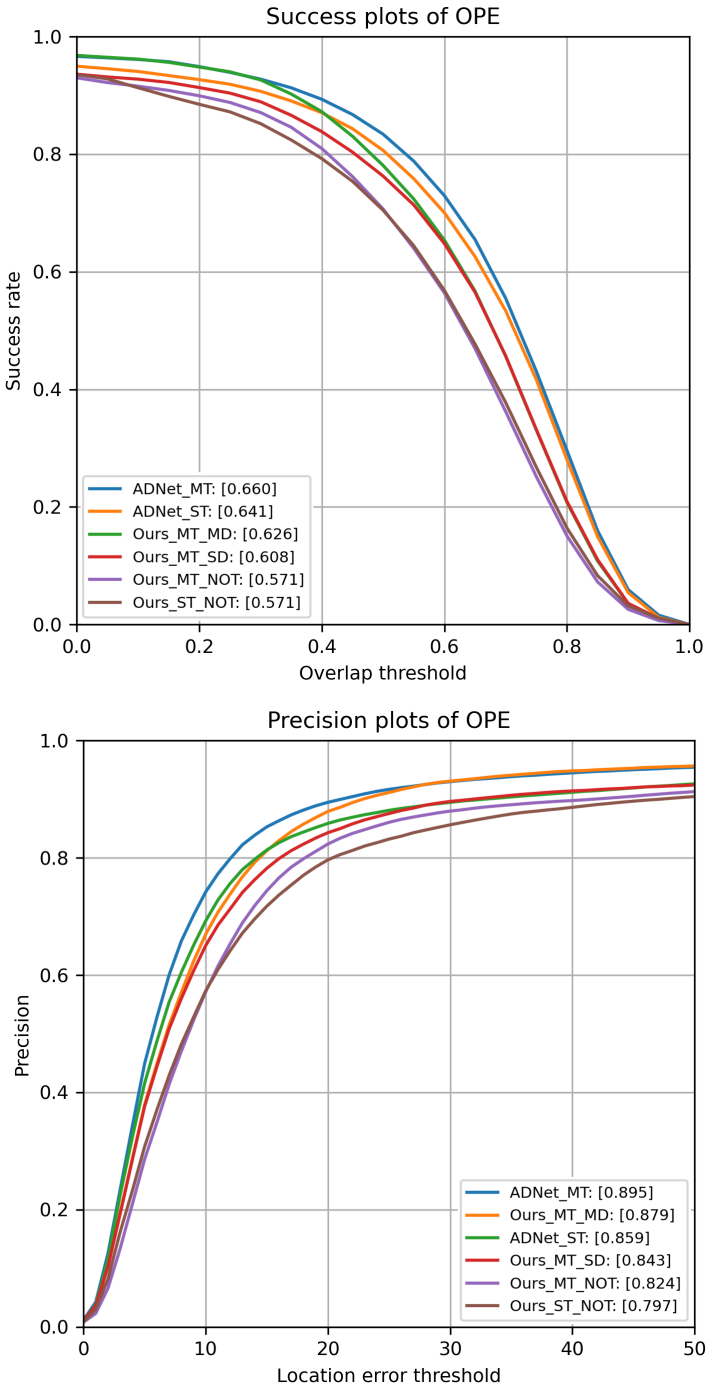


Figure 6.1: Success and Precision plots on OTB-100 (One-Pass Evaluation (OPE)). Overlap threshold and location error threshold indicate the threshold values used to compute the ROC curves. For the precision plot, the scores in the legend indicate the mean precisions when the location error threshold is 20 pixels. For the success plot, the scores indicate the area under curve (AUC).

Table 6.1: Comparison on OTB-100. Iter indicates approaches using iterative refinements. *P(20px)* indicates the mean precisions when the location error threshold is 20 pixels. *AUC of IoU*

Algorithm	**P(20px)**	**AUC of IoU**	**FPS**	**Iter**
Retina-MAML [182]	0.926	0.712	40	
VITAL [5]	0.918	0.682	2	
SiamRPN++ [183]	0.915	0.696	35	
ECO [184]	0.910	0.691	8	
MDNet [57]	0.909	0.678	1	
RDNet [79]	0.903	0.673	4	x
ADNet_MT (ours)	0.895	0.660	5	x
Hier. T. [181]	0.894	0.651	23	x
IADNET [114]	0.894	0.651	3	x
ADNet [78]	0.88	0.646	2	x
ATOM [8]	0.879	0.667	30	
Ours_MT_MD	0.879	0.626	5	x
TSAS [112]	0.861	0.651	20	x
ACT [83]	0.855	0.622	30	
TD3T [185]	0.821	0.616	23	
A3CTD [186]	0.717	0.535	50	
GOTURN [70]	0.565	0.425	125	

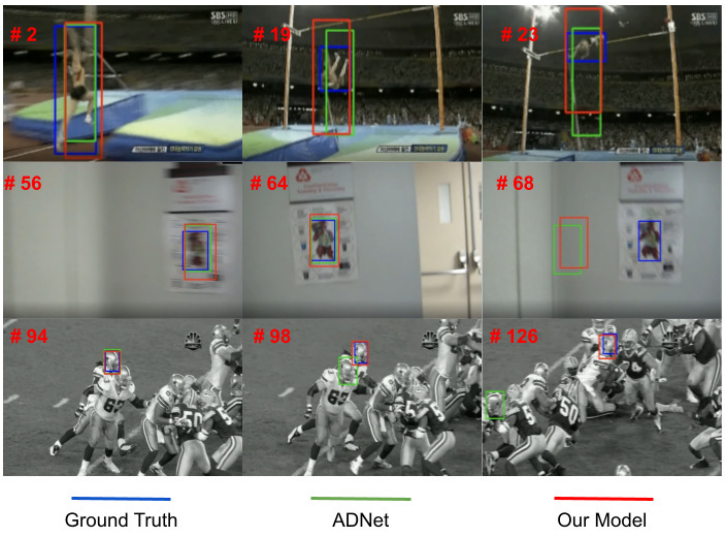


Figure 6.2: The figure shows some qualitative results of our tracker (red bounding-boxes) vs. ADNet (green bounding-boxes). Ground-truth is shown in blue.

We also compare our tracker on VOT2016 [187], VOT2018 [188], and VOT2019 [189] by adopting Expected Average Overlap (EAO), Accuracy, and Robustness as metrics, and by using the official VOT toolkit. Unfortunately, ADNet was trained on VOT data, and we could not test our modified ADNet model on this benchmarks. Results are reported in Tables 6.2–6.4, respectively.

Table 6.2: Comparison on VOT-2016.

Algorithm	**EAO**	**Accuracy**	**Robustness**
D3S [190]	0.493	0.66	0.131
UpdateNet [191]	0.481	0.61	0.21
SiamRPN++ [183]	0.464	0.64	0.20
DiMP-50 [192]	0.440	0.597	0.153
SPM [193]	0.434	0.62	0.21
ATOM [8]	0.43	0.61	0.18
ECO [184]	0.374	0.54	0.72
OUR MODEL	0.372	0.557	0.53
RDNet [79]	0.364	0.54	0.72
CCOT [194]	0.331	0.54	0.238

Table 6.3: Comparison on VOT-2018.

Algorithm	**EAO**	**Accuracy**	**Robustness**
D3S [190]	0.489	0.64	0.15
Ocean-off [195]	0.467	0.598	0.169
Retina-MAML [182]	0.452	0.604	0.159
DiMP-50 [192]	0.440	0.597	0.153
SiamRPN++ [183]	0.414	0.600	0.234
ATOM [8]	0.401	0.590	0.204
UPDT [196]	0.378	0.536	0.184
OUR MODEL	0.372	0.56	0.44
DRT [197]	0.356	0.519	0.201

Table 6.4: Comparison on VOT-2019.

Algorithm	**EAO**	**Accuracy**	**Robustness**
Retina-MAML [182]	0.313	0.57	0.366
ATOM [8]	0.292	0.603	0.411
SiamRPN++ [183]	0.292	0.58	0.446
SiamMask [198]	0.287	0.594	0.461
OUR MODEL	0.232	0.513	0.72

Among all the approaches using iterative bounding box refinements, only RDNet reports results on VOT2016. As shown in Table 6.2, our method shows improvements over RDNet in all the three metrics.

Overall, the comparison among algorithms adopting iterative bounding box refinements confirms that dealing with conflicting transformations and allowing multiple refinements at each iteration helps improve the tracking results. We also note that the performance of some algorithms such as SiamRPN++, ATOM has decreased from VOT-2016 to VOT-2018. For instance, accuracy of SiamRPN++ is 0.64 and 0.59 on VOT-2016 and VOT-2018, respectively. As for ATOM, accuracy is 0.61 and 0.6 on VOT-2016 and VOT-2018, respectively. The accuracy of the proposed algorithm is 0.557 and $0,56$ on VOT-2016 and VOT-2018, respectively. Therefore, despite accuracy slightly decreased also for our method, these results suggest that our method improves in terms of stability.

6.2.3 Discussion

The method focused on tracking strategies where the target bounding box is refined iteratively by applying a sequence of transformations. We proposed a novel formulation such that, given an image patch based on the currently estimated target bounding box, the model returns a set of N probability distributions over bounding box transformations. The method can apply multiple non-conflicting refinements at each iteration without introducing ambiguity during learning, i.e., without giving priority to some transformations over the others.

Experimental results show that the proposed iterative multi-refinement approach is superior to the single-refinement one, in-

dependently on the model/training strategies adopted. Overall, the proposed approach is competitive with respect to other state-of-the-art approaches that iteratively refine the target bounding box.

6.3 Visual Object Tracking: Reinforcement learning approach

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

In table 6.5, we group the papers based on the categorization in Fig. 4.2 and, for each method, we report the adopted DRL-methods (DQL, D-PG, AC), the adopted training dataset, and the achieved results on public benchmarks.

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 [111]	ILSVRC;ALOV	-	-	0.676	0.901	0.673	0.903	-	0.273	4	Titan Xp
BBox	DQL	DP-Siam [132]	ILSVRC	0.686	0.918	0.621	0.839	0.677	0.883	0.38	0.387	82	Titan Xp
BBox	DQL	TSAS [112]	ILSVRC	0.692	0.912	-	-	0.651	0.861	-	-	20	Titan Xp
BBox	D-PG	ADNet [78, 118]	VOT13-15; ALOV300	-	-	0.659	0.903	0.646	0.88	-	-	2.9	Titan X
BBox	D-PG	ADNet-Fast [78, 118]	VOT13-15; ALOV300	-	-	0.67	0.898	0.635	0.851	-	-	15	Titan X
BBox	D-PG	Hier. T. [115]	VOT-16	-	-	**0.681**	0.921	0.651	0.894	-	-	6.3	GTX-1060
BBox	D-PG,ML	IADNET [114]	VOT13-15	-	-	-	-	0.651	0.868	-	-	3.9	Titan X
BBox	AC	A3CTD [121]	ILSVRC, GOT-10k	-	-	-	-	0.535	0.717	-	0.1847	50	four Titan V and TITAN Xp
BBox	AC	ACT [119]	ILSVRC	0.657	0.905	-	-	0.625	0.859	0.2746	-	30	Titan X
BBox	AC	TD3T [120]	ILSVRC	0.651	0.867	-	-	0.616	0.821	-	-	23	GTX 1080
BBox	AC	PF-DRL [122]	ILSVRC	-	-	-	-	0.666	0.898	-	-	-	GTX 1080 Ti
BBox	DT,ML	FCOS-MAML [199]	TrackingNet, COCO, GOT10k,LaSOT	0.714	-	0.665	-	0.704	-	-	0.392	40	P100
BBox	DT	ATOM [64]	TrackingNet, COCO,LaSOT	-	-	-	-	-	-	-	0.401	40	Tintan X
BBox	DT	DiMP [65]	TrackingNet, COCO, GOT10k,LaSOT	-	-	-	-	0.684	-	-	**0.44**	30	GT-1080
BBox	DT	MDNet [57]	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 [71]	Youtube-BB	-	-	-	-	0.637	0.851	0.344	0.243	**160**	GTX 1060
BBox	DT	SIAM-RPN++ [73]	COCO, ImageNet, ImageNet,YouTube-BB	-	-	-	-	0.696	0.914	-	0.414	35	Titan Xp
TS	DQL	HP [129]	ILSVRC	0.629	-	0.554	0.745	0.601	0.796	0.2726	0.201	69	N.A.
TS	DQL	HP-BACF [130]	ILSVRC	-	-	-	-	0.634	0.833	-	0.113	12	GTX 1080
TS	DQL	G&M [137]	ILSVRC	0.667	0.883	0.564	0.763	0.613	0.81	-	0.2138	68	Titan X
TS	DQL,AL	SINT++ [139]	VOT13-16	-	-	0.624	0.839	0.574	0.768	-	-	-	GTX1080
TS	D-PG	DC [135]	TinyTLP; NFS	-	-	-	-		-	**0.3942**	0.311	2	Titan X
TS	D-PG	DC-RT [135]	TinyTLP; NFS	-	-	-	-	-	-	0.3516	0.252	50	Titan X
TS	DT	C-RPN [200]	LaSOT, VID, YT-BB	0.675	0.902	-	-	0.663	0.882	0.363	0.289	36	N.A.
TS	DT	SA-SiamR-IE [201]	ILSVRC, COCO, GOT10k	0.689	0.887	0.631	0.83	0.682	0.883	-	0.311	36	N.A.
TS	CF	ECO* [202]	-	0.709	0.93	-	-	0.691	0.91	0.374	0.28	12.5	N.A.
App	AC	DRL-IS [133]	ILSVRC, VOT13-15	-	-	-	-	0.671	0.909	-	-	10.2	GTX 1080 Ti
App	DQL	MT-Exp [143]	TC-128; UAV123	0.706	0.931	-	-	0.688	0.919	-	0.293	-	Titan X
App	AC	CF-AC [142]	VID	0.652	0.851	-	-	0.623	0.812	-	-	-	Titan X
App	CF	GFS-DCF [203]	-	**0.722**	**0.965**	-	-	0.693	**0.932**	-	0.397	7.8	Titan X

Table 6.5: 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. 4.6. 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 reportes 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 seconds, and GPU indicates the GPU architecture reported in the published paper. In bold are highlighted the best achieved results per column. *Results reported in [203].**All GPUs are Nvidia.

6.3.1 Comparing DRL-based Tracking Approaches

Table 6.5 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 [113, 84, 124, 126, 128, 125, 127, 204], are not comparable because experiments are run on different dataset 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 [21, 140, 144]). The table does not report VOT-15 where only 3 of the papers have reported results: DP-Siam [132], HP [129], and EAST [140] whose EAO scores are 0.39, 0.242 and 0.34 respectively.

The method DRLT [123] is not included in the table because the training procedure largely differ 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 [141] 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 [120] 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 [143] and DP-Siam [132]. However, on the VOT-18 dataset, DP-Siam achieves much higher EAO than the MT-Exp method [143], 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 [143] 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 [132] adopts a Siamese network as base tracker and is not adapted at test time. On the contrary, MT-Exp [143] 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 [20], as well as RDNet [111] 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 [78] and DRL-IS [133], both on-line adapted.

Among the DRL-based approaches aiming at predicting the target bounding box, the most performant method on the OTB-13 is TSAS [112], on OTB-50 is the Hierarchical Tracker [115], and on OTB-100 are RDNET [111] and DP-Siam [132]. On the VOT benchmarks, results are so sparse that we feel cannot draw conclusions.

The best performing method among those dealing with the tracking strategy, on OTB-100, is HP-BACF [130], which learns the tracking hyper-parameters and adopt also CFs. On the VOT benchmark, the best results are achieved by DC [135].

As for the methods dealing with the target appearance selec-

tion/updating, the best results on OTB-100 are achieved by MT-Exp [143].

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 one achieving the worst results. However, this is not confirmed by the results achieved on the VOT benchmark.

6.3.2 Comparison to the State-of-the-Art

Finally, we also compare to Deep tracking and CF-based methods. We have included in the table the widely known MDNet [57] as baseline and very recent works end-to-end trainable or adopting discriminative correlation filters (CF), adversarial learning and meta-learning. First, we stress that the comparison is included for completeness but is in general unfair. Indeed, in DRL the reward function is correlated to the tracking results thanks to a measure of the tracking accuracy, but exact actions are in general not provided during the training stage. In deep tracking, the network is trained with full knowledge of what is the right results that the network has to provide.

As the table shows, on OTB-100 the majority of DRL-based trackers performs worst than MDNet. However, this is true also for several recent baseline methods using SL. The method achieving the highest tracking results is a CF-based, namely GFS-DCF [203]. The best DRL-based trackers MT-Exp [143] and DP-Siam [132] achieve slightly inferior results with respect to GFS-DCF but comparable to VITAL [56]. As we will discuss later, MT-Exp [143] however takes advantage of a form of supervised pre-training.

Comparing the DRL-methods dealing with the bounding box prediction with the corresponding baseline methods on the OTB-100, there are DRL-methods [111, 132, 122] that are competitive with Deep trackers.

It is interesting to note that all actor-critic based trackers, which have a high complexity considering that they train two networks, and the policy gradient based approaches achieve results that are below that of most of the DT baseline methods in the table, with the exception of DRL-IS [133] which seems to be slower.

DRL	Tracker	Decisions	D/C	AH	RNN	Siam	SL	OU	CF	Reward
DQL	BBCorrection [84]	Shift/Scale BBox	D	✓	-	-	-	✓	-	$f(\Delta\text{IoU}(b_t, g))$
DQL	RDNET [111]	Shift/Scale BBox	D	-	-	✓	-	✓	-	$f(\Delta\text{IoU}(b_t, g), \Delta\text{Eu}(b_t, g))$
DQL	TSAS [112]	Shift BBox	D	-	-	-	✓	-	-	$f(\text{IoU}(b_t, g))$
DQL	DADRL [113]	Shift/Scale BBox & align	D	-	✓	-	✓	-	-	$f(\text{Eu}(\text{landmarks}_t, g)))$
DQL	DP-Siam [132]	Predict BBox, hyper-parameters	C	-	-	✓	-	-	-	$f(\text{IoU}(b_t, g))$
DQL	EAST [140]	Scale BBOx, Use next layer	D	✓	-	✓	-	-	-	$f(\Delta\text{IoU}(b_t, g))$
DQL	G&M [137]	Shift Search Area	D	✓	-	✓	-	-	-	$f(\Delta\text{IoU}(b_t, g))$
DQL	SINT++ [139]	Shift Area to occlude	D	-	-	✓	-	-	-	$f(\Delta\text{Score}(b_t, g))$
DQL	P-Track [21]	Track/Init/Update	D	✓	-	-	✓	✓	-	$f(\text{IoU}(b_t, g))$
DQL	MT-Exp [143]	Select CF	D	-	-	-	✓	✓	✓	$f(\text{IoU}(b_t, g))$
DQL	HP-Siam [129]	Hyper-parameters	C	-	-	✓	✓	-	-	$f(\text{IoU}(b_t, g))$
DQL	HP-BACF [130]	Hyper-parameters	C	-	-	-	✓	✓	✓	$f(\text{IoU}(b_t, g))$
D-PG	ADNet [78, 118]	Shift/Scale BBox	D	✓	-	-	✓	✓	-	$f(\text{IoU}(b_t, g))$
D-PG	IADNET [114]	Shift/Scale BBox	D	✓	-	-	✓	✓	-	$f(\Delta\text{IoU}(b_t, g))$
D-PG	Hier. T. [115]	Shift/Scale BBox	D	-	✓	-	-	✓	✓	$f(\text{IoU}(b_t, g))$
D-PG	DRLT [123]	Predict BBox	C	-	✓	-	-	-	-	$f(\text{D}(b_t, g))$, $f(\text{IoU}(b_t, g))$
D-PG	JDTracker [124]	Predict BBox & visibility	D&C	✓	-	-	✓	-	-	$f(\text{IoU}(b_t, g)), f(\text{visibility}, g_{vis})$
D-PG	RDT [144]	Select heatmap	D	-	-	✓	-	-	-	$f(\text{IoU}(b_t, g))$
D-PG	DC [135]	Select tracking results	D	-	-	✓	-	-	-	$f(\Delta_{Tr}\text{IoU}(b_t, g))$
AC	ACT [119]	Shift/Scale BBox	C	-	-	-	✓	✓	-	$f(\text{IoU}(b_t, g))$
AC	TD3T [120]	Shift/Scale BBox	C	-	-	-	✓	✓	-	$f(\text{IoU}(b_t, g))$
AC	A3CT [121]	Shift/Scale BBox	C	-	✓	✓	-	-	-	$f(\text{IoU}(b_t, b_t^{exp}))$
AC	CRAC [138]	Scale Search Area	D	✓	-	-	-	✓	-	$f(\Delta\text{Score}_{GAN})$
AC	PF-DRL [122]	Shift/Scale Expected BBox	C	✓	-	-	✓	✓	-	$f(\text{IoU}(b_t, g))$
AC	CF-AC [142]	Select CF	D	-	-	-	-	✓	✓	$f(\text{IoU}(b_t, g))$
AC	DRL-IS [133]	Track/Init/Update	D	-	-	-	✓	✓	-	$f(a,\text{IoU}(b_t, g), \Delta\text{IoU}(b_t, g))$
AC	Active_DRL [125]	Camera parameters	C	-	✓	-	-	-	-	$f(loc_{target})$
AC	ADVat [126]	Camera parameters	D	-	✓	-	-	-	-	$f(loc_{target})$
AC	VOS [145]	Update Template	D	-	-	✓	-	-	-	$f(\text{IoU}(Mask_t, g))$

Table 6.6: Theoretical comparison of DRL-based trackers grouped based on the adopted DRL algorithm. The column DRL specify 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

6.3.3 Limitations of DRL-based tracking

Table 6.6 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, which in other words means providing more annotations to train the models. In the following, we try to analyze the above mentioned issues and techniques, and the characteristics of the approaches that we believe are more critical.

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.

Implementation details about cropping and padding are often missing. In some cases, before cropping the image, the bounding box is rescaled such that the resulting image has twice the size of the target. This is generally done to include contextual information. It is unclear if padding is done to preserve the target aspect ratio especially when the latter differs from the aspect ratio of the image in input to the network. Rescaling the target bounding box poses also issues when dealing with objects of large size with respect to the size of the frame (as often happens in ILSRC-VID).

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. It seems that in general the target template is the image cropped with the latest target bounding box. This helps to address the problem of varying target appearance. Nonetheless, it may favor the drifting problem since the agent decisions are taken based on uncertain target templates. On the other hand, using the cropped image from the first frame is challenging considering that the target orientation and pose can change over time. To account for these issues, re-detection or validation procedure 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 [119, 120]).

Despite the search area is in general enlarged to facilitate the target detection, it remains unclear how agents learn to deal with occlusions. In this sense, the approach in [123], which processes

the whole frame, may have more chance to reacquire the target when the occlusion ends. The method in [111] is interesting in that it takes in input a search area and a cropped target image. The latter is not a template from 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.

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 interesting that a three-branches network receiving such inputs to train the policy has not been proposed yet.

Methods selecting CFs [142, 143] use two different approaches. [143] evaluates a heatmap at time and can maintain an unlimited number of CFs. The agent decides if the heatmap is reliable or no. On the contrary, [142] takes in input a set of heatmaps of fixed size. This limits the number of CFs to be used to perform tracking but allow the agent to take a decision by jointly considering all available heatmaps.

Action Space and History

As shown in table 6.6, most of the approaches adopt discrete actions (between 4 and roughly 11). This choice limits the complexity of the model, which is already difficult to train due to the large state space. Works focusing on continuous actions to modify the target bounding box have adopted models trained by DDPG [99] (as in [119, 122]) or its variant TD3 (in [120]). Hence such models train a deterministic policy. Works such as [124, 123] use instead the REINFORCE algorithm [97] while [121, 125] adopt A3C [92].

The only method using DQL [20] is [132] where the network is used to also predict the tracking hyper-parameters. An elegant approach is the one proposed in [129] where a probability distribution over the action is trained by NAF [96].

As pointed out in [84], 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). The problem can be addressed by perturbing the state after an action selection cycle is detected. The problem of getting stuck in a local optimum can be more common than it seems, and this may justify the implementation choice of considering more than an initial bounding box/candidate in works such as [111, 132, 112, 133, 145]. In contrast to such approaches, methods in [84, 140, 137, 21, 78, 114, 116, 124, 138, 122] prefer to provide in input to the network the history of the selected actions. This important implementation choice has the effect of stabilizing the learning procedure limiting the problem of entering in an action selection cycle. 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.

We have noticed that methods that do not provide in input the action history, tend to provide in input the past target locations (as in [144]) or to include in the model a recurrent neural network (as in [113, 115, 123, 121, 125, 126]). This may suggest that, to correctly learn what decision to take, the agent may benefit from knowing also what attempts it has already done in the past. It is possible to speculate that also knowing the progresses associated with the past attempts may help the agent to select optimal new actions. However, at test time the reward is unavailable and more

investigations to provide such kind of feedback to the agents are suggested. While the above argument is easy to understand when the agent decides how to modify iteratively the bounding box, it becomes cumbersome to understand 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 no the appearance model).

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 [84] where the box is moved of one pixel along the vertical/horizontal directions and hence accuracy can be reached at a pixel-level at the cost of an increased number of iterations. Since the number of iterations is generally bounded, this accuracy may not be easily reached especially for objects moving fast. Under this point of view, continuous actions should be preferable but, as already said, models to estimate continuous actions are more difficult to train.

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 [129, 143], 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 6.6 refers to the reward function

adopted by 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:

$$f(s_t, a_t, g_t) = \begin{cases} \alpha, & \text{if } IoU(s_t, g_t) > \tau \\ -\alpha, & \text{else} \end{cases} \tag{6.1}$$

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 [84, 119, 140, 132, 133], 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 [111, 125]. Absolute difference D is used in [123]. In [139, 138], a function of the improvement of the tracking score is used, while in [124], since the agent has to take 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 [121, 129, 113, 114, 125, 137, 142, 123, 120, 126, 115, 135, 145, 112]. The main limitation of functions like that in Eq. 6.1 is that there is no difference in the reward of actions yielding to high IoU. In other words, the agent has settled for bounding box 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 [142] 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 [87] 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 [205], 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 [206], 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 future research directions.

Pre-training of the Policy

There are a number of works [112, 113, 21, 143, 129, 78, 114, 124, 119, 120, 122] 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 [137, 139, 144, 135, 138, 133], despite a supervised pre-training is not directly employed, still the model takes advantage of trackers trained in a supervised way. In other cases [119, 120, 122], an expert is used to guide the learning process and, even if indirectly or only in a limited way, supervision over the actions to take is essentially provided. Somewhat different is the case of [121] 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).

In practice, most of the above 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 [20], 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 than 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 [205] 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. At 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 directions to investigate in the future.

We also point out that an interesting initialization of the Q-network has been discussed in [21] where a form of heuristically guided Q-learning is adopted. In particular, the Q-network is pretrained in a supervised way 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. In other words, they define a target policy that, differently than the original DQL approach in [20], is truly optimal (compare with the original definition of Q-function in Eq. 4.3). Such important implementation detail helps the network to estimate Q-values in the expected range and presumably accelerate the learning.

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 have a negative impact on the learning process [207]. Unfortunately, we have not found in the ablation study of the analyzed papers details about the effect of varying the memory buffer size. It is even unclear if such buffer is actually really needed. As detailed in Sec. 6.3.3, without a proper initialization of the network, the memory buffer would not contain useful information but mostly biased samples which may hinder the learning process.

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 [119], imbalanced batches can hurt the learning process. In [119], 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 idea is similar to the Prioritized Experience replay [208] where samples from the experience replay buffer are drawn based on the TD error.

Tracker Initialization and Updating

In most of the analyzed approaches, part of the network (we refer to it as observation network) is devoted to extract appearance features of the target, while another part is used to implement the policy. It is common practice in online adapted deep tracking to fine-tune the last layers of the observation network to adapt to the target by using bounding boxes from the first frame. One of such approaches is MDNet [57], updates at test time are also required when using CFs.

As shown in Table 6.6, most of the approaches do not require any updating at test time or particular initialization a part for the bounding box in the first frame. The methods taking advantage of some form of updating at test time are [143, 130, 115, 142] to retrain the CFs, and [84, 111, 119, 78, 114, 80, 122, 120, 112, 138] to retrain part of the network/policy.

We highlight here an interesting feature of the latter methods. While [111, 84, 133] 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 [119, 78, 122, 114, 120] claim to use a supervised learning strategy to update the policy network. For these methods, it becomes unclear the true contribution of DRL to the learning of the policy parameters. As for the method in [112], no details about the way the network is fine-tuned on the first frame are available.

Which DRL algorithm to choose?

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 [111] com-

pares DRL approaches (Dueling DQN [209] vs DQN [20]), but no comparison among DQL, AC and D-PG has been presented yet. In other benchmarks, such as the Atari games, different DRL approaches have been compared and new algorithms proposed. For instance, in [210] it has been shown that the integration of Double DQL [94], Dueling DQN [209], N-Step learning [87], Prioritized Experience Replay [211], Distributional RL [212], Noisy Network [213] 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 thinner the differences between DQL and D-PG methods [214]. In [215, 216], 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 [217] such that regularization terms are required.

As for the use of DRL algorithms in visual tracking, 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 [119, 120]).

In conclusion, under a theoretical point of view, AC methods might be preferable. However, as shown in Table 6.5, regardless of the actions/goals of the tracking algorithm (reported in the

column Decisions of Table 6.6), the best performing DRL-based tracking algorithms are MT-Exp [143] end DP-Siam [132], both based on DQL. We note that Hierarchical Tracker [115] and DRL-IS [133], based on D-PG and AC respectively, achieve results comparable to the state-of-the-art, but the differences in results may be due more to implementation details than to the adopted DRL algorithms. Overall, despite the progress in the field, more validation and studies are required to define the best DRL-algorithm to use for visual tracking and, currently, there is not a winning approach.

6.4 Accelerating learning of deep models

In this section, we present results of the validation of our method. We first present the selected neural architectures used to assess our method. We also provide details about how to apply our method to improve the training efficiency of these architectures. Then, we detail the datasets selected to perform the experiments and the hyper-parameters adopted on each dataset.

The results of our experiments are reported in Tables 6.7 and 6.8. In each table, "Network" indicates the neural architecture; "Dataset" specifies the dataset used for training and testing the model; "SGD", "Freezing" and "Dropping" indicate the strategy used to train the network. In particular, SGD is the baseline method, namely the model is trained in a standard way without attempting any training efficiency. Freezing represents our implementation of the method in [154] where, however, layers to freeze are selected in the same way as we do in our method. In this case, weights of selected layers are excluded from the training (frozen) but the layers are not physically removed from the trained model. The column Drop-

ping reports results of our layer dropping method. The columns indicated with "T" report the duration of the training and refers to the time in minutes required to complete the expected number of training epochs, including the warm up epochs. Columns "A" report the test accuracy values, measuring the percentage of correct predictions made by the model on the test set. Finally, columns "ΔT" report the percentage of time saved by applying a training efficiency strategy (Freezing or Dropping) with respect to the baseline (SGD). In particular, we compute this metric as:

$$\Delta T = \frac{T_{SGD} - T}{T_{SGD}} \cdot 100. \tag{6.2}$$

All experiments have been carried on a machine equipped with: GPU RTX 3090 24GB, RAM 96 GB, processor Intel(R) Xeon(R) CPU E5-2403 1.80GHz. In our implementation, feature maps produced by the dropped layers are stored on disk directly as PyTorch tensor using the “Pickle" Python package [218], that implements binary protocols for serializing and de-serializing a Python object. We experimentally noted that using Pickle is faster than writing and reading files on disk with the Numpy package [219] and PyTorch [220].

6.4.1 Neural Architectures

This method focused on improving training efficiency of CNNs. To assess our method we considered two neural network architectures widely adopted in the computer vision field. VGG (Visual Geometry Group) is a convolutional neural network introduced in [157]. The VGG architecture is characterized by its depth and the use of small convolutional filters. It consists of a sequence of convolutional layers, followed by a sequence of fully-connected

layers. There are several configurations of the VGG architecture. The smaller version is the VGG-11 with only 11 layers. The VGG-16 has 16 layers and the VGG-19 has 19 layers. As the number of layers in VGG increases, so do the training time and memory requirements.

From the experiments carried out with VGG, we found out that the order of the curves representing the layer scores P'_l across the epochs depends on the inclusion in the model of the batch normalization layer. In fact, as it can be seen in Fig. 5.2, we note that the scores order of the layers of VGG+BN is inverse respect to the scores order of the VGG without BN. This is because batch normalization speeds up learning in neural networks by normalizing the inputs to each layer, which reduces the internal covariate shift. This makes the optimization more stable and allows the network to learn more quickly and with higher accuracy. Hence, based on our experiments, to use our technique with a VGG it is recommended to include the batch normalization layer (one after each convolutional layer).

ResNet (Residual Network) is a convolutional neural network introduced in [158]. ResNet is characterized by the use of residual blocks, which help to alleviate the vanishing gradient problem and allow for the creation of much deeper neural networks. The original ResNet architecture has several configurations, including ResNet-18, a relatively small version of the ResNet architecture with only 18 layers. ResNet-50, ResNet-101, and ResNet-152 are much deeper versions of the ResNet architecture with 50, 101, and 152 layers, respectively.

We point out that in the case of ResNet, we have to save not only the features maps but also the output of the skip connections to maintain the original behaviour as shown in Fig. 5.3.

Table 6.7: Fast Training of VGG architectures. SGD refers to the standard training strategy of the entire model. Freezing refers to excluding the parameters of some layers from the training without removing the layers from the model. Dropping is our method where layers are deleted from the trained model. T is the training time in minutes. A is the test accuracy value. ΔT is the percentage of reduced training time with respect to the time of SGD.

		SGD		Freezing			Dropping (Ours)		
Network	**Dataset**	**T (min)**	**A (%)**	**T (min)**	**A (%)**	ΔT (%)	**T (min)**	**A (%)**	ΔT (%)
VGG-11	MNIST	20.83	98.64	19.58	98.25	6.00	8.74	98.25	58.04
VGG-11	CIFAR-10	23.83	92.02	23.54	91.72	1.21	8.21	91.73	65.54
VGG-11	Imagenette	61.01	75.33	59.33	74.08	2.75	18.32	74.08	69.97
VGG-16	MNIST	22.54	98.85	21.45	98.26	4,24	9.01	98.26	60.03
VGG-16	CIFAR-10	26.54	93.12	24.94	92.84	6.03	9.56	92.86	63.98
VGG-16	Imagenette	74.73	78.76	71.21	77.83	4.71	25.23	77.83	66.24
VGG-19	MNIST	23.02	98.52	22.68	96.14	1.48	9.45	96.22	58.95
VGG-19	CIFAR-10	27.02	93.10	25.78	91.71	4.59	11.53	91.74	57.33
VGG-19	Imagenette	110.35	80.32	105.34	78.13	4.54	37.76	78.16	65.78

Table 6.8: Fast Training of ResNet architectures. SGD refers to the standard training strategy of the entire model. Freezing refers to excluding the parameters of some layers from the training without removing the layers from the model. Dropping is our method where layers are deleted from the trained model. T is the training time in minutes. A is the test accuracy value. ΔT is the percentage of reduced training time with respect to the time of SGD.

		SGD		Freezing			Dropping (Ours)		
Network	**Dataset**	**T (min)**	**A (%)**	**T (min)**	**A (%)**	ΔT (%)	**T (min)**	**A (%)**	ΔT (%)
ResNet-18	MNIST	23.67	98.2	23.10	97.80	2.41	8.64	97.78	63.50
ResNet-18	CIFAR-10	27.67	92.25	25.97	91.82	6.14	11.90	91.82	56.99
ResNet-18	Imagenette	253.07	80.12	242.32	79.07	4.25	83.78	79.12	66.89
ResNet-50	MNIST	35.43	98.75	35.02	96.85	1.16	11.23	96.85	68.30
ResNet-50	CIFAR-10	38.43	94.40	35.40	92.05	7.88	13.05	92.04	66.04
ResNet-50	Imagenette	336.00	82.78	315.34	80.34	6.15	86.28	80.34	74.32
ResNet-101	MNIST	53.12	97.81	51.64	95.45	2.79	18.56	95.51	65.06
ResNet-101	CIFAR-10	56.12	93.98	52.03	91.26	7.29	19.53	91.28	65.20
ResNet-101	Imagenette	402.34	82.23	380.23	80.76	5.29	120.44	80.75	70.06
ResNet-152	MNIST	70.76	97.43	65.30	95.12	7.72	23.34	95.11	67.01
ResNet-152	CIFAR-10	74.76	93.45	68.93	91.03	7.80	25.75	91.14	65.56
ResNet-152	Imagenette	540.76	82.65	504.72	79.45	6.66	180.34	79.48	66.65

6.4.2 Dataset and Hyper-parameters

To evaluate our algorithm we use three popular classification datasets: MNIST [221], CIFAR-10 [222], and Imagenette [223].

The MNIST dataset contains 60,000 gray scale images, with 50,000 for training and 10,000 for test. It includes 10 classes, and each class is represented by 6,000 images.

The CIFAR-10 dataset contains 60,000 color images, with 50,000 for training and 10,000 for test. Overall, there are 10 classes, and, for each class, 6,000 images.

Imagenette [223] includes a subset of 10 classes from the larger Imagenet [224]. The classes are *tench, English springer, cassette player, chain saw, church, French horn, garbage truck, gas pump, golf ball, parachute*. Overall, the adopted dataset includes about 1,000 color images per class with a resolution of 160 x 160 pixels. The images are obtained from the ones in the original Imagenet dataset by performing a resizing that preserves the original aspect ratio.

We opted to use MNIST and CIFAR-10 datasets to ensure the comparability of our work with [154]. In addition, we included the more complex Imagenette dataset to evaluate the performance of our work on a more challenging dataset.

In all our experiments, we use a batch size of 256 samples. We set the learning rates to values generally used in literature, while the number of epochs and warm up are selected empirically. On the MNIST and CIFAR-10 datasets, the total number of epochs (including the model warm-up) is set to 60. On the MNIST dataset, the learning rate is fixed to 0.001. The warm-up epochs are 5. On the CIFAR-10 dataset, the learning rate is fixed to 0.1 and scaled x10 after 20 epochs. The warm-up epochs are 10. Finally, on the Imagenette, the number of epochs is set to

150 and the learning rate is fixed to 0.01 and scaled x10 after 50 epochs. The warm-up epochs are 25.

6.4.3 Results and Comparison

Table 6.7 reports the results obtained by applying our method to models of various depth (11, 16, and 19) in the family of VGG architectures. All models include batch-normalization. First we note that, across the models and datasets, the impact on accuracy values of freezing or dropping layers to increase training efficiency is negligible. The differences in the accuracy values vary, for both techniques, in the range 0.26 (for VGG-16 trained on CIFAR-10) and 2.38 (for VGG-19 trained on MNIST). These differences increase slightly with network depth and are generally higher for the Imagenette dataset. However, these small decreases in accuracy values come with a reduction in training time. As shown in the table, for VGG models, while the training time reduction with the freezing layer technique varies in the range 0.40% (for VGG-16 on the MNIST dataset) and 6.03% (for VGG-16 on the CIFAR-10 dataset), with our layer dropping technique the training time reduction varies in the range of 58.04% (for VGG-11 on the MNIST dataset) and 69.97% (for VGG-11 on the Imagenette dataset). While on average these percentages are 3.52% for the freezing layer method, they are 62.87% with our technique.

These results are not limited to VGG architectures. Indeed, Table 6.8 reports similar achievements for ResNet architectures of various depth (18, 50, 101, and 152). In particular, analyzing the results in a similar way to what was done for the VGG architectures, the differences in the accuracy values vary, for the layer freezing and dropping techniques, in the range 0.4% (for ResNet-18 on the MNIST dataset) and 3.2% (for ResNet-152 on the Im-

Table 6.9: FLOPs reduction across architectures. SGD refers to the standard training strategy of the entire model. Dropping is our method where layers are deleted from the trained model. FLOPs are measured during the forward propagation. ΔFLOPs is the percentage of reduced FLOPs with respect to SGD.

	SGD	Dropping (Ours)	
Network	**FLOPs**	**FLOPs**	**ΔFLOPs (%)**
VGG-11	31,203.60	25,847.02	17.17
VGG-16	50,311.19	33,049.44	34.31
VGG-19	66,000.00	44,079.31	33.21
ResNet-18	1,987.80	640.53	67.78
ResNet-50	4,704.59	2,193.45	53.38
ResNet-101	9,262.2	1,667.27	82.00
ResNet-152	13,823.39	2,247.46	83.74

agenette dataset). For both techniques, the impact on accuracy values generally increases with network depth. In terms of training time reduction, with the layer freezing technique, the percentages vary in the range 1.16% (for ResNet-50 on the MNIST dataset) and 7.8% (for Resnet-152 on the CIFAR-10 dataset). Our approach achieves training time reduction percentages in the range 56.99% (for ResNet-18 on the CIFAR-10 dataset) and 74.32% (for ResNet-50 on the Imagenette dataset). While, on average, layer freezing accounts for a 5.46% of training time reduction, our method results in average training time reduction of approximately 66.30%.

Overall, the training time for the VGG and ResNet architectures is more than halved with our technique despite the loss in accuracy values being comparable to that obtained by freezing the layers.

6.4.4 Training Time and Parameter Reduction

The results discussed in Sec. 6.4 refer to the impact of our technique on model accuracy and the training time reduction achieved on our machine. To further analyze and explain the performance of our technique, this section refers to the effect it has on the number of parameters and operations performed during forward propagation at training time. Indeed, our technique not only reduces the number of weights for which it is necessary to estimate partial derivatives during gradient back-propagation, but also affects the number of operations performed during forward propagation.

A potential bottleneck in our method is the feature map saving to disk whenever layer dropping occurs. In some networks, the size of the features maps produced by a layer may be greater than the size of the input images; thus, reading and writing feature maps with large dimensions can cause a slowdown of the training. However, we have observed empirically that layers dropping never takes place layer by layer but generally several subsequent layers are dropped together. Figure 6.3 shows (on the left) the test accuracy values of ResNet-18 (top row) and VGG-16 (bottom row) on the MNIST dataset over the epochs for SGD, the layer freezing method and our technique. As already discussed, our technique and the freezing layers one have a limited impact on the final model accuracy. On the right, the figure shows the training time per epoch of each technique. In particular, the time spent by our method refers to: the time to store feature maps on the disk whenever layer dropping arises, and the time to train the model. As shown in the plot, the time to store feature maps on the drive is concentrated only in few epochs since, as already said, our method does not constantly drop layers. The time to store the maps to disk depends on their size and, thus, is higher for the VGG model.

The time to train the model includes, for all methods, the time for computing the gradients, updating the layer weights, loading the training data and performing forward propagation. The plot shows how, in our technique, this time decreases over time and becomes much lower than that of the other methods.

To further investigate this result, we discuss the FLOPs (floating point operations per second) of our method versus the baseline method (SGD) and the layer freezing technique we are comparing. More specifically, we measured the MMAC (Mega Multiply-Accumulate), a FLOPs metric that counts the number of matrix multiplications and accumulations (MACs) a neural network performs in one second. The metric is expressed in millions (mega) of operations and is useful for evaluating the computational complexity of a neural network and comparing the performance of different architectures. Fig. 6.4 shows, on the left, the number of parameters per each epoch for the ResNet-101 (top) and VGG-16 (bottom). Red curves represent the number of model parameters when using the baseline and the layer freezing techniques; Green curves are the number of parameters when using our method. Since our method compresses the model at training time, there is a strong reduction in the number of parameters corresponding to the epochs when layer dropping arises. This reduction of parameters is correlated with the MMAC reduction during forward propagation, shown in the plots on the right. While the MMAC is constant for the baseline and the layer freezing techniques, in our method it keeps decreasing due to the shrinkage in the number of parameters.

Table 6.9 reports the FLOPS required for the forward propagation during training by using SGD versus our approach (Dropping). The FLOPs refers to various deep learning models, includ-

ing VGG-11, VGG-16, VGG-19, ResNet-18, ResNet-50, ResNet-101, and ResNet-152. The second column shows the FLOPs required when training the entire model, the third column shows the FLOPs required when using our approach. The final column ΔFLOPs shows the percentage difference between the FLOPs of the two approaches.

Overall, the table suggests that the Dropping approach is more efficient than the baseline method. The percentage difference between the two approaches ranges from 17.17% (for VGG-11) to 83.74% (for ResNet-152), indicating that Dropping can significantly reduce the computational burden of training deep neural networks, especially for very deep models like ResNet-101 and ResNet-152. The table also shows that the reduction of the FLOPs increase with the depth of the model. Moreover, has also indicated by the magnitude of the FLOPs, the VGG family performs more operations than the ResNet due to the presence of dense layers applied to feature maps of greater size.

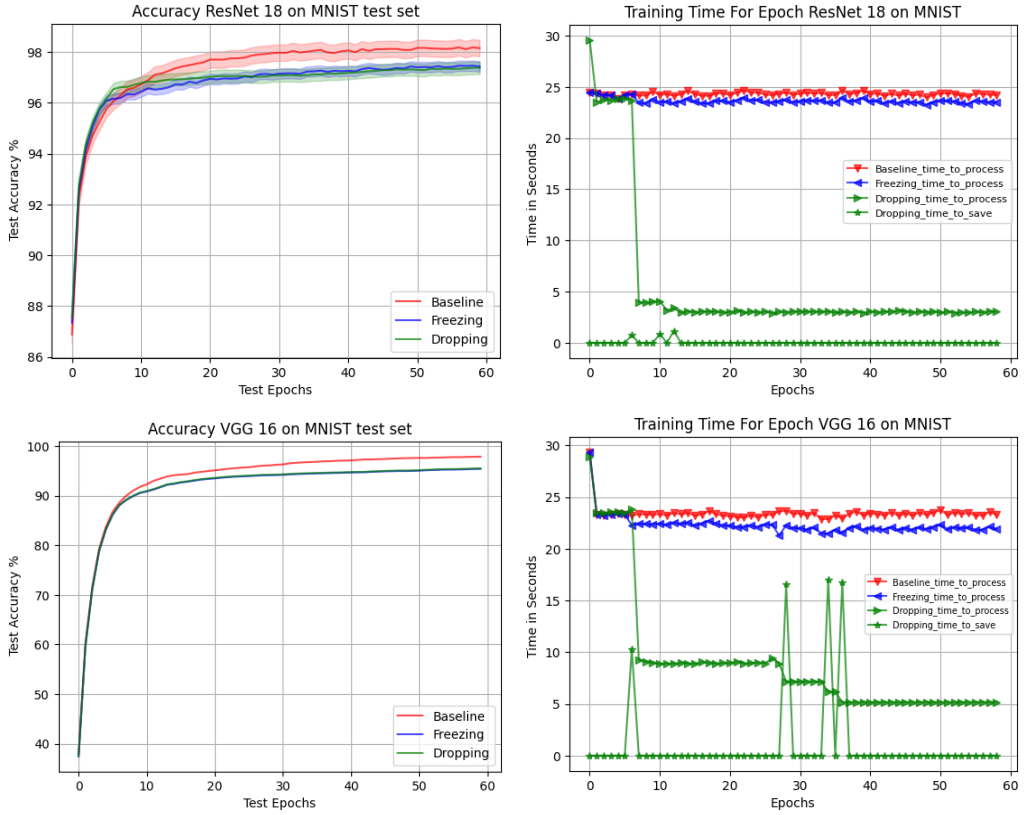


Figure 6.3: The plots on the left show the test accuracy values of a ResNet-18 (top) and VGG-16 (bottom) trained on the MNIST dataset with different strategies: SGD (red curves), layer freezing (blue curves), and layer dropping (green curves). The experiments were repeated 10 times with different starting weights and data randomization. Freezing and dropping layers achieve nearly equivalent test accuracy values for both the architectures. The values are slightly lower than those achieved when training the entire model. On the right, the plots show training time per epoch for the same models and dataset. The red and blue curves represent the training time for the SGD and layer freezing strategies respectively. The green curves refer to our layer dropping approach. Starred curves show the time required to store the feature maps to disk, while the other curves show the training time which decreases over the epochs due to the lower cost of forward propagation in our method. Note that Intermediate feature maps in VGG-16 have greater size than in ResNet and the time to store the maps to disk is higher. However, this operation is not frequent and its cost is amortized over time.

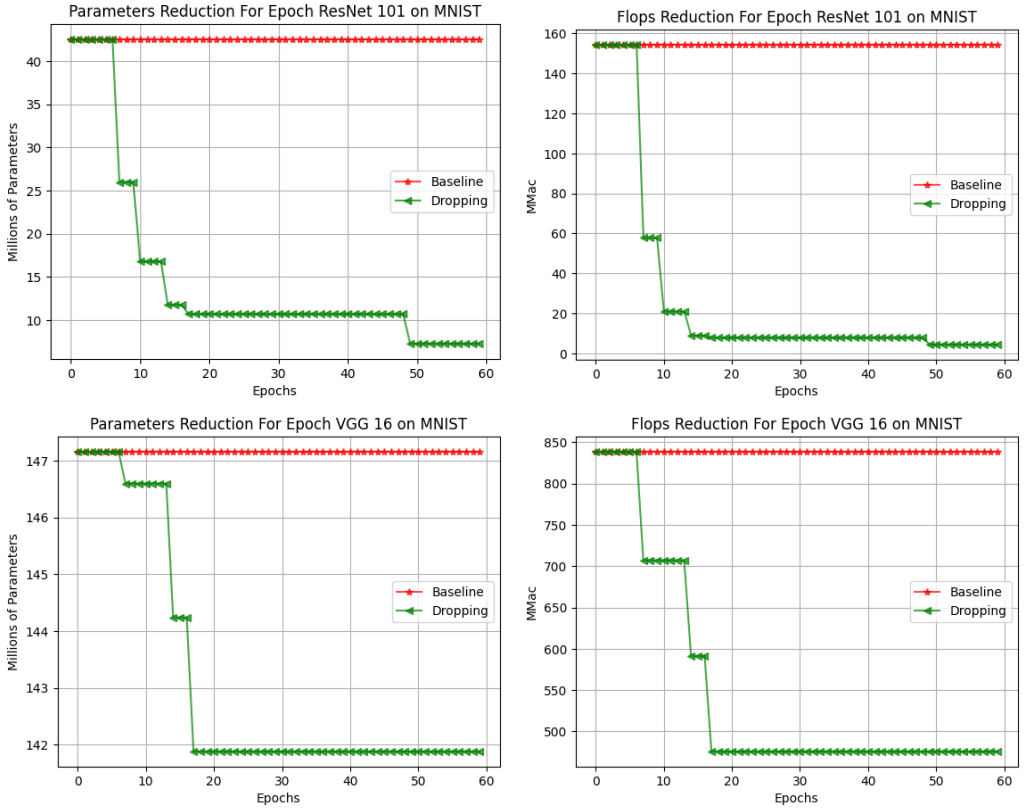


Figure 6.4: On the left, the plot shows the number of network parameters in each epoch for the ResNet-101 (top) and VGG-16 (bottom). When training the entire model or using the layer freezing technique (red curves), the number of parameters remains constant. With the proposed method (green curves), the model shrinks over the epochs and there is a reduction in the number of parameters each time a sequence of layers is dropped. This parameter reduction is correlated with the MMac reduction, shown in the plots on the right for both the models. By reducing the network parameters, the number of operations during forward propagation is decreased.

Chapter 7

Conclusions and Future Work

This chapter provides a summary of the primary contributions made by this thesis, along with potential directions for future research.

7.1 Contributions to Visual Tracking

Visual tracking is a crucial technique used in many vision-based applications, which aims to accurately and efficiently track targets within videos. However, this task is often complicated by various challenges, such as occlusion and scale variation. This thesis addresses these challenges by carrying out a study on deep reinforcement learning algorithms applied to tracking and presenting a new visual tracking algorithm that significantly improves the performance of iterative tracking methods.

- This thesis proposes a novel tracking framework based on iterative multi-refinements to address the issues present in the traditional method of selecting the bounding box refinement during target tracking. In particular, it introduces the concept of conflicting refinements and train our model to handle

them. It formulates the problem in such a way that multiple non-conflicting transformations can be applied simultaneously, leading to faster tracking with the ability to handle a higher number of composite transformations. It eliminates ambiguity during the training procedure by avoiding giving priority to some refinements over others. Experimental results show that the proposed iterative multi-refinement approach is superior to the single-refinement one, independently of the model/training strategies adopted. Overall, the proposed approach is competitive with respect to other state-of-the- art approaches that iteratively refine the target bounding box.

- This thesis presents the current state-of-the-art in DRL-based visual tracking and a possible ways to model visual tracking within the RL framework has been presented. In this thesis we categorize DRL-based tracking approaches based on the main pursued goal: estimating the target bounding box, the tracking strategy, the search area where to locate the target. Finally, a third group includes methods aiming at modeling the target appearance selection/update process. This thesis 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 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 achieves tracking results on the OTB benchmark that are below the MD-Net performance. Performance of some approaches are below the state-of-the-art, but still some few methods are competitive. Overall, the application of DRL to visual tracking seems promising because it allows the system generating samples based on the interaction of the agent with the environment and, hence, it provides a mechanism for generating diverse and informative samples during training. However, there are still issues to study and solve. For example, the problems of initializing the policy in an effective way, 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 is a lack of training strategies that can enable the learning of proper policies.

7.2 Contributions to Fast Learning of CNNs

Although connected to the fine-tuning of visual tracking models, the contributions on fast learning can be generalized to other contexts where network acceleration and fine-tuning are required. We have proposed a method to improve the training efficiency of convolutional networks by gradually compressing the model through dropping of convolutional layers during the training phase. While most of the state-of-the-art methods focus only on reducing time

and operations during back propagation, we have noticed that forward propagation contributes significantly to the overall computational cost of the training procedure.

We have empirically found that in the ResNet and VGG architectures, the layers closest to the input learn faster than those close to the output. Therefore, the proposed method uses a gradient-based score to identify the layers to be dropped. Dropping arises from the first layer of the network to the last. In contrast to previous work, which freeze layer parameters to speed up model training, our approach is to remove layers from the network only during training. To this end, our method splits the network into a “tail", consisting of the dropped layers, and a “head", consisting of the layers yet to be trained. The tail is used to prepare the data to feed the head. The head is a smaller version of the model that trains over time.

The method has been validated on three popular datasets to train models of various depth in the VGG and ResNet families. Our experiments show that, on average, our method achieves a time reduction of 62.87% on VGG architectures, and of 66.30% on ResNet models. Thus, by reducing the time of forward propagation during training, the overall training time is more than halved with a limited impact on the model accuracy that, in our experiments, never exceeds the 3.2% for very deep network (ResNet-152).

A potential limitation of our method is that it requires a warm-up training to initialize the network weights. This is often done in fast training algorithms and network compression to avoid pruning important weights. In these works, as in ours, the number of warm-up epochs is found empirically. However, in our method, it might be equivalently possible to define a threshold on the layer

score to start dropping layers when their weights no longer change much. Since our method significantly reduces the training time, it might also be possible to use a hybrid strategy that alternates among training the whole model and the head model. This strategy can help refine the first dropped layers' weights at the cost of increasing the overall training time.

Theoretically, the method can also be used on dense layers. However, in our experiments, we have observed that the layer scores of dense layers do not always have increasing values. Thus, there is a risk of dropping a dense layer whose weights can still improve.

7.3 Future Work

Visual tracking is a highly complex and challenging problem, and many state-of-the-art visual trackers can be severely impacted by numerous factors. The state-of-the-art DL-based visual trackers have been categorized into a comprehensive taxonomy based on network architecture, network exploitation, training, network objective, network output, the exploitation of correlation filter advantages, aerial-view tracking, long-term tracking, and online tracking. These methods aim to address the main problems and proposed solutions of DL-based trackers.

Despite recent advances, small target tracking remains an area that requires further research and development to improve the accuracy and robustness of visual tracking systems. Although most visual trackers are intended for general object tracking, small target tracking has not received sufficient attention, despite its prevalence in various applications. Accurately tracking a target in the video captured by a drone-mounted camera, for instance,

requires advanced small target tracking capabilities that are often lacking in current visual trackers.

Furthermore, it is worth noting that most visual tracking benchmark datasets and evaluation metrics have been investigated, and the various state-of-the-art trackers have been compared on seven visual tracking datasets. Continued research and innovation in small target tracking capabilities are crucial to enable better performance in various real-world applications, such as drone surveillance, autonomous vehicles, and robotics. Improved accuracy and robustness of visual tracking systems can be achieved by integrating advanced DL-based trackers and developing novel evaluation metrics that are tailored to specific applications.

Data-driven visual tracking algorithms rely heavily on models trained on vast amounts of labeled data. However, manual labeling of the target in each video frame using a bounding box is an expensive and time-consuming process.

The abundance of unlabeled data available on the internet presents a significant opportunity to improve the performance of visual trackers. Exploring ways to leverage this unlabelled data, such as unsupervised and self-supervised learning techniques, can potentially enhance the accuracy and robustness of visual tracking systems without incurring the high cost of manual labeling.

Therefore, developing innovative approaches to effectively use unlabeled data in training data-driven visual tracking algorithms could significantly reduce the reliance on expensive labeled data and accelerate the development of more accurate and efficient visual trackers. This would have a profound impact on various real-world applications, including surveillance, autonomous vehicles, and robotics.

Long-term visual tracking is a critical area of research in com-

puter vision. Unlike short-term tracking, long-term visual tracking aims to track the target for extended periods, requiring the visual tracker to understand the target's semantic properties beyond its initial appearance in the video. Incorporating the target's semantic information into visual tracking algorithms can improve their accuracy and robustness.

As many modern visual tracking algorithms fall into the "tracking by detection" category, the boundary between object tracking and object detection is becoming less explicit. Thus, integrating the target's semantic information in visual tracking algorithms could bridge this gap and enhance the system's ability to track targets over extended periods.

Overall, addressing the challenges associated with long-term visual tracking will require the development of novel techniques for integrating semantic information and improving the robustness of visual tracking algorithms. These advancements will play a critical role in enabling better performance and wider applicability of visual tracking in various real-world applications.

To advance both fields of object tracking and object detection, it is essential to identify their similarities and differences and effectively combine them into real-world applications. Currently, most state-of-the-art visual trackers rely on deep neural networks, which are the primary tool used by researchers in the field of AI. However, these networks are good at memorizing information but lack the ability to reason and draw inferences.

For instance, if a person being tracked suddenly gets into a car and drives away, the target becomes invisible, and visual trackers find it challenging to infer that the person is now in the car. Humans can quickly reason that the target has moved to the car, but most visual trackers fail to track the target in such scenarios.

Therefore, improving the reasoning abilities of AI systems is crucial to enhancing the performance of visual trackers. Novel techniques such as integrating explicit reasoning mechanisms can bridge the gap between memorization and reasoning and improve the ability of visual trackers to track targets effectively in complex real-world scenarios.

Continued research and innovation in AI and visual tracking will play a critical role in enabling the development of more robust and accurate visual tracking systems. These systems will have a broader applicability in various real-world applications, including surveillance, autonomous vehicles, and robotics.

In future work, it will be interesting to investigate the extent to which the approach can be applied to other more complex architectures such as multi-branch models like Siamese networks or models designed for time-series processing, such as those that include recurrent memory cells (Recurrent Neural Network - RNN). For instance, in multi-branch networks, one should check whether learning proceeds similarly along each of the branches. Also, it would be useful to study whether similar techniques can be applied to visual transformers.

Bibliography

[1] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In *2005 IEEE computer society conference on computer vision and pattern recognition (CVPR'05)*, volume 1, pages 886–893. Ieee, 2005.

[2] David G Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60:91–110, 2004.

[3] Stephen O'Hara and Bruce A Draper. Introduction to the bag of features paradigm for image classification and retrieval. *arXiv preprint arXiv:1101.3354*, 2011.

[4] Ran Tao, Efstratios Gavves, and Arnold WM Smeulders. Siamese instance search for tracking. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1420–1429, 2016.

[5] Yibing Song, Chao Ma, Xiaohe Wu, Lijun Gong, Linchao Bao, Wangmeng Zuo, Chunhua Shen, Rynson WH Lau, and Ming-Hsuan Yang. Vital: Visual tracking via adversarial learning. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8990–8999, 2018.

[6] Hyeonseob Nam and Bohyung Han. Learning multi-domain convolutional neural networks for visual tracking. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4293–4302, 2016.

[7] Ke Song, Wei Zhang, Weizhi Lu, Zheng-Jun Zha, Xiangyang Ji, and Yibin Li. Visual object tracking via guessing and matching. *IEEE Transactions on Circuits and Systems for Video Technology*, 30(11):4182–4191, 2019.

[8] Martin Danelljan, Goutam Bhat, Fahad Shahbaz Khan, and Michael Felsberg. Atom: Accurate tracking by overlap maximization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4660–4669, 2019.

[9] Guangting Wang, Chong Luo, Xiaoyan Sun, Zhiwei Xiong, and Wenjun Zeng. Tracking by instance detection: A meta-learning approach. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 6288–6297, 2020.

[10] Sangdoo Yun, Jongwon Choi, Youngjoon Yoo, Kimin Yun, and Jin Young Choi. Action-driven visual object tracking with deep reinforcement learning. *IEEE transactions on neural networks and learning systems*, 29(6):2239–2252, 2018.

[11] Qiang Wang, Li Zhang, Luca Bertinetto, Weiming Hu, and Philip HS Torr. Fast online object tracking and segmentation: A unifying approach. In *Proceedings of the IEEE/CVF conference on Computer Vision and Pattern Recognition*, pages 1328–1338, 2019.

[12] Zongpu Zhang, Yang Hua, Tao Song, Zhengui Xue, Ruhui Ma, Neil Robertson, and Haibing Guan. Tracking-assisted weakly supervised online visual object segmentation in unconstrained videos. In *Proceedings of the 26th ACM international conference on Multimedia*, pages 941–949, 2018.

[13] Alan Lukezic, Jiri Matas, and Matej Kristan. D3s-a discriminative single shot segmentation tracker. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 7133–7142, 2020.

[14] Rui Yao, Guosheng Lin, Shixiong Xia, Jiaqi Zhao, and Yong Zhou. Video object segmentation and tracking: A survey. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 11(4):1–47, 2020.

[15] In Su Kim, Hong Seok Choi, Kwang Moo Yi, Jin Young Choi, and Seong G Kong. Intelligent visual surveillance—a survey. *International Journal of Control, Automation and Systems*, 8:926–939, 2010.

[16] Javed Ahmed, Mubarak Shah, Andrew Miller, Don Harper, and M Noman Jafri. A vision-based system for a ugv to handle a road intersection. In *Proceedings of the National Conference on Artificial Intelligence*, volume 22, page 1077. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2007.

[17] Benjamin Coifman, David Beymer, Philip McLauchlan, and Jitendra Malik. A real-time computer vision system for vehicle tracking and traffic surveillance. *Transportation Research Part C: Emerging Technologies*, 6(4):271–288, 1998.

[18] Jen-Chao Tai, Shung-Tsang Tseng, Ching-Po Lin, and Kai-Tai Song. Real-time image tracking for automatic traffic monitoring and enforcement applications. *Image and Vision Computing*, 22(6):485–501, 2004.

[19] Sian Barris and Chris Button. A review of vision-based motion analysis in sport. *Sports Medicine*, 38:1025–1043, 2008.

[20] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[21] James Supancic III and Deva Ramanan. Tracking as online decision-making: Learning a policy from streaming videos with reinforcement learning. In *Int. Conf. on Computer Vision*, pages 322–331, 2017.

[22] Weining Wang, Yan Huang, and Liang Wang. Language-driven temporal activity localization: A semantic matching reinforcement learning model. In *Proc. of the Conf. on Computer Vision and Pattern Recognition*, pages 334–343, 2019.

[23] Juan C Caicedo and Svetlana Lazebnik. Active object localization with deep reinforcement learning. In *Proc. of the Int. Conf. on Computer Vision*, pages 2488–2496, 2015.

[24] Wenhao Wu, Dongliang He, Xiao Tan, Shifeng Chen, and Shilei Wen. Multi-agent reinforcement learning based frame sampling for effective untrimmed video recognition. In *Proc. of the Int. Conf. on Computer Vision*, pages 6222–6231, 2019.

[25] Junwei Han, Le Yang, Dingwen Zhang, Xiaojun Chang, and Xiaodan Liang. Reinforcement cutting-agent learning for video object segmentation. In *Computer Vision and Pattern Recognition*, pages 9080–9089, 2018.

[26] Giorgio Cruciata, Liliana Lo Presti, and Marco La Cascia. Iterative multiple bounding-box refinements for visual tracking. *Journal of Imaging*, 8(3):61, 2022.

[27] Giorgio Cruciata, Liliana Lo Presti, and Marco La Cascia. On the use of deep reinforcement learning for visual tracking: A survey. *IEEE Access*, 9:120880–120900, 2021.

[28] Rudolph Emil Kalman. A new approach to linear filtering and prediction problems. *Journal of basic Engineering*, 82(1):35–45, 1960.

[29] Andrew Blake, Rupert Curwen, and Andrew Zisserman. A framework for spatiotemporal control in the tracking of visual contours. *Int. Journal of Computer Vision*, 11(2):127–145, 1993.

[30] Jianguang Lou, Hao Yang, Wei Ming Hu, Tieniu Tan, et al. Visual vehicle tracking using an improved ekf. In *Proc. Asian Conf. Computer Vision*, pages 296–301, 2002.

[31] Eric A Wan and Rudolph Van Der Merwe. The unscented kalman filter for nonlinear estimation. In *Proc. of the IEEE 2000 Adaptive Systems for Signal Processing, Communications, and Control Symposium (Cat. No. 00EX373)*, pages 153–158. Ieee, 2000.

[32] Neil J Gordon, David J Salmond, and Adrian FM Smith. Novel approach to nonlinear/non-gaussian bayesian state es-

timation. In *IEE proceedings F (radar and signal processing)*, volume 140, pages 107–113. IET, 1993.

[33] Peihua Li, Tianwen Zhang, and Bo Ma. Unscented kalman filter for visual curve tracking. *Image and vision computing*, 22(2):157–164, 2004.

[34] Michael Isard and Andrew Blake. Condensation—conditional density propagation for visual tracking. *Int. journal of computer vision*, 29(1):5–28, 1998.

[35] Apurva Bedagkar-Gala and Shishir K Shah. A survey of approaches and trends in person re-identification. *Image and Vision Computing*, 32(4):270–286, 2014.

[36] Boris Babenko, Ming-Hsuan Yang, and Serge Belongie. Robust object tracking with online multiple instance learning. *IEEE transactions on pattern analysis and machine intelligence*, 33(8):1619–1632, 2010.

[37] Zdenek Kalal, Krystian Mikolajczyk, and Jiri Matas. Tracking-learning-detection. *IEEE transactions on pattern analysis and machine intelligence*, 34(7):1409–1422, 2011.

[38] Frédéric Jurie, Michel Dhome, et al. Real time robust template matching. In *BMVC*, pages 1–10, 2002.

[39] Rafael C Gonzales and Richard E Woods. Digital image processing, 2002.

[40] David S Bolme, Bruce A Draper, and J Ross Beveridge. Average of synthetic exact filters. In *Computer Vision and Pattern Recognition*, pages 2105–2112. IEEE, 2009.

[41] David S Bolme, J Ross Beveridge, Bruce A Draper, and Yui Man Lui. Visual object tracking using adaptive correlation filters. In *Computer Vision and Pattern Recognition*, pages 2544–2550. IEEE, 2010.

[42] Martin Danelljan, Gustav Hager, Fahad Shahbaz Khan, and Michael Felsberg. Learning spatially regularized correlation filters for visual tracking. In *Int. Conf. on computer vision*, pages 4310–4318, 2015.

[43] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. Deep learning book. *MIT Press*, 521(7553):800, 2016.

[44] Yann LeCun, Bernhard E Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne E Hubbard, and Lawrence D Jackel. Handwritten digit recognition with a back-propagation network. In *Advances in neural information processing systems*, pages 396–404, 1990.

[45] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[46] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[47] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proc. of the Conf. on computer vision and pattern recognition*, pages 1–9, 2015.

[48] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proc. of the Conf. on computer vision and pattern recognition*, pages 3431–3440, 2015.

[49] Dana H Ballard. Modular learning in neural networks. In *AAAI*, pages 279–284, 1987.

[50] Sumit Chopra, Raia Hadsell, Yann LeCun, et al. Learning a similarity metric discriminatively, with application to face verification. In *CVPR (1)*, pages 539–546, 2005.

[51] John J Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proc. of the national academy of sciences*, 79(8):2554–2558, 1982.

[52] Zachary C Lipton, John Berkowitz, and Charles Elkan. A critical review of recurrent neural networks for sequence learning. *arXiv preprint arXiv:1506.00019*, 2015.

[53] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[54] Joe Yue-Hei Ng, Matthew Hausknecht, Sudheendra Vijayanarasimhan, Oriol Vinyals, Rajat Monga, and George Toderici. Beyond short snippets: Deep networks for video classification. In *Proc. of the Conf. on computer vision and pattern recognition*, pages 4694–4702, 2015.

[55] Wongun Choi and Silvio Savarese. A unified framework for multi-target tracking and collective activity recognition. In *European Conference on Computer Vision*, pages 215–230. Springer, 2012.

[56] Yibing Song, Chao Ma, Xiaohe Wu, Lijun Gong, Linchao Bao, Wangmeng Zuo, Chunhua Shen, Rynson WH Lau, and Ming-Hsuan Yang. Vital: Visual tracking via adversarial learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8990–8999, 2018.

[57] Hyeonseob Nam and Bohyung Han. Learning multi-domain convolutional neural networks for visual tracking. In *Proc. of the Conf. on computer vision and pattern recognition*, pages 4293–4302, 2016.

[58] Ilchae Jung, Jeany Son, Mooyeol Baek, and Bohyung Han. Real-time mdnet. In *Proc. of the Eur. Conf. on Computer Vision (ECCV)*, pages 83–98, 2018.

[59] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *Proc. of Int. Conf. on computer vision*, pages 2961–2969, 2017.

[60] Ross Girshick. Fast r-cnn. In *Proc. of the Int. Conf. on computer vision*, pages 1440–1448, 2015.

[61] Lijun Wang, Wanli Ouyang, Xiaogang Wang, and Huchuan Lu. Stct: Sequentially training convolutional networks for visual tracking. In *Conf. on computer vision and pattern recognition*, pages 1373–1381, 2016.

[62] Martin Danelljan, Gustav Hager, Fahad Shahbaz Khan, and Michael Felsberg. Convolutional features for correlation filter based visual tracking. In *Int. Conf. on Computer Vision Workshops*, pages 58–66, 2015.

[63] Chao Ma, Jia-Bin Huang, Xiaokang Yang, and Ming-Hsuan Yang. Hierarchical convolutional features for visual tracking. In *Proc. of the Int. Conf. on computer vision*, pages 3074–3082, 2015.

[64] Martin Danelljan, Goutam Bhat, Fahad Shahbaz Khan, and Michael Felsberg. Atom: Accurate tracking by overlap maximization. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4660–4669, 2019.

[65] Goutam Bhat, Martin Danelljan, Luc Van Gool, and Radu Timofte. Learning discriminative model prediction for tracking. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 6182–6191, 2019.

[66] Martin Danelljan, Luc Van Gool, and Radu Timofte. Probabilistic regression for visual tracking. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7183–7192, 2020.

[67] Ran Tao, Efstratios Gavves, and Arnold WM Smeulders. Siamese instance search for tracking. In *Proc. of CVPR*, pages 1420–1429, 2016.

[68] Luca Bertinetto, Jack Valmadre, Joao F Henriques, Andrea Vedaldi, and Philip HS Torr. Fully-convolutional siamese networks for object tracking. In *Eur. Conf. on computer vision*, pages 850–865. Springer, 2016.

[69] Kai Chen and Wenbing Tao. Once for all: a two-flow convolutional neural network for visual tracking. *IEEE Trans. on Circuits and Systems for Video Technology*, 28(12):3377–3386, 2017.

[70] David Held, Sebastian Thrun, and Silvio Savarese. Learning to track at 100 fps with deep regression networks. In *Eur. Conf. on Computer Vision*, pages 749–765. Springer, 2016.

[71] Bo Li, Junjie Yan, Wei Wu, Zheng Zhu, and Xiaolin Hu. High performance visual tracking with siamese region proposal network. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8971–8980, 2018.

[72] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015.

[73] Bo Li, Wei Wu, Qiang Wang, Fangyi Zhang, Junliang Xing, and Junjie Yan. Siamrpn++: Evolution of siamese visual tracking with very deep networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4282–4291, 2019.

[74] Tianyu Yang and Antoni B Chan. Recurrent filter learning for visual tracking. In *Int. Conf. on Computer Vision*, pages 2010–2019, 2017.

[75] Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives. *IEEE Trans. on PAMI*, 35(8):1798–1828, 2013.

[76] Peixia Li, Dong Wang, Lijun Wang, and Huchuan Lu. Deep visual tracking: Review and experimental comparison. *Pattern Recognition*, 76:323–338, 2018.

[77] Yi Wu, Jongwoo Lim, and Ming-Hsuan Yang. Online object tracking: A benchmark. In *Proc. of CVPR*, pages 2411–2418, 2013.

[78] Sangdoo Yun, Jongwon Choi, Youngjoon Yoo, Kimin Yun, and Jin Young Choi. Action-decision networks for visual tracking with deep reinforcement learning. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2711–2720, 2017.

[79] Yifan Jiang, David K Han, and Hanseok Ko. Relay dueling network for visual tracking with broad field-of-view. *IET Computer Vision*, 13(7):615–622, 2019.

[80] Liangliang Ren, Xin Yuan, Jiwen Lu, Ming Yang, and Jie Zhou. Deep reinforcement learning with iterative shift for visual tracking. In *Proc. of the ECCV*, pages 684–700, 2018.

[81] Chen Huang, Simon Lucey, and Deva Ramanan. Learning policies for adaptive tracking with deep feature cascades. In *Proc. of the IEEE ICCV*, pages 105–114, 2017.

[82] Wei Zhang, Ke Song, Xuewen Rong, and Yibin Li. Coarse-to-fine uav target tracking with deep reinforcement learning. *IEEE Trans. on Automation Science and Engineering*, 16(4):1522–1530, 2018.

[83] Boyu Chen, Dong Wang, Peixia Li, Shuang Wang, and Huchuan Lu. Real-time'actor-critic'tracking. In *Proceedings of the European conference on computer vision (ECCV)*, pages 318–334, 2018.

[84] Yifan Jiang, Hyunhak Shin, and Hanseok Ko. Precise regression for bounding box correction for improved tracking based

on deep reinforcement learning. In *Int. Conf. on Acoustics, Speech and Signal Processing*, pages 1643–1647. IEEE, 2018.

[85] Ken Chatfield, Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Return of the devil in the details: Delving deep into convolutional nets. *arXiv preprint arXiv:1405.3531*, 2014.

[86] Arnold WM Smeulders, Dung M Chu, Rita Cucchiara, Simone Calderara, Afshin Dehghan, and Mubarak Shah. Visual tracking: An experimental survey. *IEEE transactions on pattern analysis and machine intelligence*, 36(7):1442–1468, 2013.

[87] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018.

[88] Leslie Pack Kaelbling, Michael L Littman, and Andrew W Moore. Reinforcement learning: A survey. *Journal of artificial intelligence research*, 4:237–285, 1996.

[89] Laëtitia Matignon, Guillaume J Laurent, and Nadine Le Fort-Piat. Reward function and initial values: better choices for accelerated goal-directed reinforcement learning. In *Int. Conf. on Artificial Neural Networks*, pages 840–849. Springer, 2006.

[90] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.

[91] Martin Riedmiller. Neural fitted q iteration–first experiences with a data efficient neural reinforcement learning

method. In *Eur. Conf. on Machine Learning*, pages 317–328. Springer, 2005.

[92] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *Int. conf. on machine learning*, pages 1928–1937, 2016.

[93] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

[94] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *Proc. of Conf. on artificial intelligence*, 2016.

[95] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Van Hasselt, Marc Lanctot, and Nando De Freitas. Dueling network architectures for deep reinforcement learning. *arXiv preprint arXiv:1511.06581*, 2015.

[96] Shixiang Gu, Timothy Lillicrap, Ilya Sutskever, and Sergey Levine. Continuous deep q-learning with model-based acceleration. In *Int. Conf. on Machine Learning*, pages 2829–2838, 2016.

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

[98] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. 2014.

[99] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[100] Scott Fujimoto, Herke van Hoof, and David Meger. Addressing function approximation error in actor-critic methods. *arXiv preprint arXiv:1802.09477*, 2018.

[101] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *Int. conf. on machine learning*, pages 1889–1897, 2015.

[102] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[103] Fang Liu and Jianbo Su. Reinforcement learning-based feature learning for object tracking. In *Proc. of the Int. Conf. on Pattern Recognition*, volume 2, pages 748–751. IEEE, 2004.

[104] Qing Guo, Ruize Han, Wei Feng, Zhihao Chen, and Liang Wan. Selective spatial regularization by reinforcement learned decision making for object tracking. *IEEE Transactions on Image Processing*, 2019.

[105] Andrew D Bagdanov, Alberto Del Bimbo, Walter Nunziati, and Federico Pernici. A reinforcement learning approach to active camera foveation. In *Proc. of Int. workshop on video*

surveillance and sensor networks, pages 179–186. ACM, 2006.

[106] Andre Cohen and Vladimir Pavlovic. Reinforcement learning for robust and efficient real-world tracking. In *Int. Conf. on Pattern Recognition*, pages 2989–2992. IEEE, 2010.

[107] Kourosh Meshgi, Maryam Sadat Mirzaei, and Shigeyuki Oba. Long and short memory balancing in visual co-tracking using q-learning. *arXiv preprint arXiv:1902.05211*, 2019.

[108] Sarang Khim, Sungjin Hong, Yoonyoung Kim, and Phill kyu Rhee. Adaptive visual tracking using the prioritized q-learning algorithm: Mdp-based parameter learning approach. *Image and Vision Computing*, 32(12):1090–1101, 2014.

[109] Yu Xiang, Alexandre Alahi, and Silvio Savarese. Learning to track: Online multi-object tracking by decision making. In *Proc. of the Int. Conf. on computer vision*, pages 4705–4713, 2015.

[110] Chenglong Wu, Hao Sun, Hongqi Wang, Kun Fu, Guangluan Xu, Wenkai Zhang, and Xian Sun. Online multi-object tracking via combining discriminative correlation filters with making decision. *IEEE Access*, 6:43499–43512, 2018.

[111] Yifan Jiang, David K Han, and Hanseok Ko. Relay dueling network for visual tracking with broad field-of-view. *IET Computer Vision*, 13(7):615–622, 2019.

[112] Zhu Teng, Baopeng Zhang, and Jianping Fan. Three-step action search networks with deep q-learning for real-time object tracking. *Pattern Recognition*, page 107188, 2020.

[113] Minghao Guo, Jiwen Lu, and Jie Zhou. Dual-agent deep reinforcement learning for deformable face tracking. In *Proc. of the Eur. Conf. on Computer Vision (ECCV)*, pages 768–783, 2018.

[114] Detian Huang, Lingke Kong, Jianqing Zhu, and Lixin Zheng. Improved action-decision network for visual tracking with meta-learning. *IEEE Access*, 7:117206–117218, 2019.

[115] Bineng Zhong, Bing Bai, Jun Li, Yulun Zhang, and Yun Fu. Hierarchical tracking by reinforcement learning-based searching and coarse-to-fine verifying. *IEEE Transactions on Image Processing*, 28(5):2331–2341, 2018.

[116] Ming-xin Jiang, Chao Deng, Zhi-geng Pan, Lan-fang Wang, and Xing Sun. Multiobject tracking in videos based on lstm and deep reinforcement learning. *Complexity*, 2018, 2018.

[117] Alejandro Newell, Kaiyu Yang, and Jia Deng. Stacked hourglass networks for human pose estimation. In *Eur. Conf. on computer vision*, pages 483–499. Springer, 2016.

[118] Sangdoo Yun, Jongwon Choi, Youngjoon Yoo, Kimin Yun, and Jin Young Choi. Action-driven visual object tracking with deep reinforcement learning. *IEEE transactions on neural networks and learning systems*, 29(6):2239–2252, 2018.

[119] Boyu Chen, Dong Wang, Peixia Li, Shuang Wang, and Huchuan Lu. Real-time actor-critic tracking. In *ECCV*, pages 318–334, 2018.

[120] Shengjie Zheng and Huan Wang. Real-time visual object tracking based on reinforcement learning with twin delayed

deep deterministic algorithm. In *Int. Conf. on Intelligent Science and Big Data Engineering*, pages 165–177. Springer, 2019.

[121] Matteo Dunnhofer, Niki Martinel, Gian Luca Foresti, and Christian Micheloni. Visual tracking by means of deep reinforcement learning and an expert demonstrator. In *Proc. of the Int. Conf. on Computer Vision Workshops*, pages 0–0, 2019.

[122] Qianqian Wang, Liansheng Zhuang, Ning Wang, Wengang Zhou, and Houqiang Li. Learning motion-aware policies for robust visual tracking. In *Int. Conf. on Multimedia and Expo*, pages 1786–1791. IEEE, 2019.

[123] Da Zhang, Hamid Maei, Xin Wang, and Yuan-Fang Wang. Deep reinforcement learning for visual object tracking in videos. *arXiv preprint arXiv:1701.08936*, 2017.

[124] Xiaobai Liu, Qian Xu, Thuan Chau, Yadong Mu, Lei Zhu, and Shuicheng Yan. Revisiting jump-diffusion process for visual tracking: a reinforcement learning approach. *IEEE Transactions on Circuits and Systems for Video Technology*, 2018.

[125] Wenhan Luo, Peng Sun, Fangwei Zhong, Wei Liu, Tong Zhang, and Yizhou Wang. End-to-end active object tracking and its real-world deployment via reinforcement learning. *IEEE transactions on pattern analysis and machine intelligence*, 2019.

[126] Fangwei Zhong, Peng Sun, Wenhan Luo, Tingyun Yan, and Yizhou Wang. Ad-vat: An asymmetric dueling mechanism for learning visual active tracking. 2018.

[127] Wenhan Luo, Peng Sun, Fangwei Zhong, Wei Liu, Tong Zhang, and Yizhou Wang. End-to-end active object tracking via reinforcement learning. *Proc. of the Int. Conf. on Machine Learning*, 2018.

[128] Fangwei Zhong, Peng Sun, Wenhan Luo, Tingyun Yan, and Yizhou Wang. Ad-vat+: An asymmetric dueling mechanism for learning and understanding visual active tracking. *IEEE transactions on pattern analysis and machine intelligence*, 2019.

[129] Xingping Dong, Jianbing Shen, Wenguan Wang, Yu Liu, Ling Shao, and Fatih Porikli. Hyperparameter optimization for tracking with continuous deep q-learning. In *Proc. of the Conf. on Computer Vision and Pattern Recognition*, pages 518–527, 2018.

[130] Xingping Dong, Jianbing Shen, Wenguan Wang, Ling Shao, Haibin Ling, and Fatih Porikli. Dynamical hyperparameter optimization via deep reinforcement learning in tracking. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2019.

[131] Hamed Kiani Galoogahi, Ashton Fagg, and Simon Lucey. Learning background-aware correlation filters for visual tracking. In *Proc. of the Int. Conf. on Computer Vision*, pages 1135–1143, 2017.

[132] Mohamed H Abdelpakey and Mohamed S Shehata. Dp-siam: Dynamic policy siamese network for robust object tracking. *IEEE Transactions on Image Processing*, 29:1479–1492, 2019.

[133] Liangliang Ren, Xin Yuan, Jiwen Lu, Ming Yang, and Jie Zhou. Deep reinforcement learning with iterative shift for visual tracking. In *Proc. of the Eur. Conf. on Computer Vision*, pages 684–700, 2018.

[134] Lijun Wang, Wanli Ouyang, Xiaogang Wang, and Huchuan Lu. Visual tracking with fully convolutional networks. In *Proc. of the Int. Conf. on computer vision*, pages 3119–3127, 2015.

[135] Zhao Zhong, Zichen Yang, Weitao Feng, Wei Wu, Yangyang Hu, and Cheng-Lin Liu. Decision controller for object tracking with deep reinforcement learning. *IEEE Access*, 7:28069–28079, 2019.

[136] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. In *Eur. Conf. on computer vision*, pages 21–37. Springer, 2016.

[137] Ke Song, Wei Zhang, Weizhi Lu, Zheng-Jun Zha, Xiangyang Ji, and Yibin Li. Visual object tracking via guessing and matching. *IEEE Transactions on Circuits and Systems for Video Technology*, 2019.

[138] Wenfeng Song, Shuai Li, Tao Chang, Aimin Hao, Qinping Zhao, and Hong Qin. Cross-view contextual relation transferred network for unsupervised vehicle tracking in drone videos. In *The IEEE Winter Conference on Applications of Computer Vision*, pages 1707–1716, 2020.

[139] Xiao Wang, Chenglong Li, Bin Luo, and Jin Tang. Sint++: robust visual tracking via adversarial positive instance gen-

eration. In *Proc. of the Conf. on Computer Vision and Pattern Recognition*, pages 4864–4873, 2018.

[140] Chen Huang, Simon Lucey, and Deva Ramanan. Learning policies for adaptive tracking with deep feature cascades. In *Proc. of the Int. Conf. on Computer Vision*, pages 105–114, 2017.

[141] Yuanpei Liu, Xingping Dong, Wenguan Wang, and Jianbing Shen. Teacher-students knowledge distillation for siamese trackers. *arXiv preprint arXiv:1907.10586*, 2019.

[142] Yanchun Xie, Jimin Xiao, Kaizhu Huang, Jeyarajan Thiyagalingam, and Yao Zhao. Correlation filter selection for visual tracking using reinforcement learning. *arXiv preprint arXiv:1811.03196*, 2018.

[143] Wenju Huang, Yuwei Wu, and Yunde Jia. Tracker-level decision by deep reinforcement learning for robust visual tracking. In *Int. Conf. on Image and Graphics*, pages 442–453. Springer, 2019.

[144] Janghoon Choi, Junseok Kwon, and Kyoung Mu Lee. Real-time visual tracking by deep reinforced decision making. *Computer Vision and Image Understanding*, 171:10–19, 2018.

[145] Mingjie Sun, Jimin Xiao, Eng Gee Lim, Bingfeng Zhang, and Yao Zhao. Fast template matching and update for video object tracking and segmentation. *CVPR*, 2020.

[146] Daniel Bolya, Chong Zhou, Fanyi Xiao, and Yong Jae Lee. Yolact: Real-time instance segmentation. In *Proceedings*

of the IEEE international conference on computer vision, pages 9157–9166, 2019.

[147] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.

[148] Gaurav Menghani. Efficient deep learning: A survey on making deep learning models smaller, faster, and better. *arXiv preprint arXiv:2106.08962*, 2021.

[149] Pavlo Molchanov, Arun Mallya, Stephen Tyree, Iuri Frosio, and Jan Kautz. Importance estimation for neural network pruning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11264–11272, 2019.

[150] Tejalal Choudhary, Vipul Mishra, Anurag Goswami, and Jagannathan Sarangapani. Heuristic-based automatic pruning of deep neural networks. *Neural Computing and Applications*, 34(6):4889–4903, 2022.

[151] Ryad Zemouri, Nabil Omri, Farhat Fnaiech, Noureddine Zerhouni, and Nader Fnaiech. A new growing pruning deep learning neural network algorithm (gp-dlnn). *Neural Computing and Applications*, 32:18143–18159, 2020.

[152] Yihui He, Xiangyu Zhang, and Jian Sun. Channel pruning for accelerating very deep neural networks. In *Proceedings of the IEEE international conference on computer vision*, pages 1389–1397, 2017.

[153] Sheng Xu, Hanlin Chen, Xuan Gong, Kexin Liu, Jinhu Lü, and Baochang Zhang. Efficient structured pruning based on

deep feature stabilization. *Neural Computing and Applications*, 33(13):7409–7420, 2021.

[154] Xueli Xiao, Thosini Bamunu Mudiyanselage, Chunyan Ji, Jie Hu, and Yi Pan. Fast deep learning training through intelligently freezing layers. In *2019 International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*, pages 1225–1232. IEEE, 2019.

[155] Shiwei Liu, Iftitahu Ni'mah, Vlado Menkovski, Decebal Constantin Mocanu, and Mykola Pechenizkiy. Efficient and effective training of sparse recurrent neural networks. *Neural Computing and Applications*, 33:9625–9636, 2021.

[156] Jiaqi Zhang, Xiangru Chen, Mingcong Song, and Tao Li. Eager pruning: Algorithm and architecture support for fast training of deep neural networks. In *2019 ACM/IEEE 46th Annual International Symposium on Computer Architecture (ISCA)*, pages 292–303. IEEE, 2019.

[157] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[158] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[159] Yann LeCun, John Denker, and Sara Solla. Optimal brain damage. *Advances in neural information processing systems*, 2, 1989.

[160] Babak Hassibi, David G Stork, and Gregory J Wolff. Optimal brain surgeon and general network pruning. In *IEEE international conference on neural networks*, pages 293–299. IEEE, 1993.

[161] Jian-Hao Luo, Jianxin Wu, and Weiyao Lin. Thinet: A filter level pruning method for deep neural network compression. In *Proceedings of the IEEE international conference on computer vision*, pages 5058–5066, 2017.

[162] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.

[163] Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. *Advances in neural information processing systems*, 28, 2015.

[164] Andreas Veit, Michael J Wilber, and Serge Belongie. Residual networks behave like ensembles of relatively shallow networks. *Advances in neural information processing systems*, 29, 2016.

[165] Adam Polyak and Lior Wolf. Channel-level acceleration of deep face representations. *IEEE Access*, 3:2163–2175, 2015.

[166] Shi Chen and Qi Zhao. Shallowing deep networks: Layer-wise pruning based on feature representations. *IEEE transactions on pattern analysis and machine intelligence*, 41(12):3048–3056, 2018.

[167] Pengtao Xu, Jian Cao, Fanhua Shang, Wenyu Sun, and Pu Li. Layer pruning via fusible residual convolutional block for deep neural networks. *arXiv preprint arXiv:2011.14356*, 2020.

[168] Sara Elkerdawy, Mostafa Elhoushi, Abhineet Singh, Hong Zhang, and Nilanjan Ray. To filter prune, or to layer prune, that is the question. In *Proceedings of the Asian Conference on Computer Vision*, 2020.

[169] Dayu Tan, Weimin Zhong, Xin Peng, Qiang Wang, and Vladimir Mahalec. Accurate and fast deep evolutionary networks structured representation through activating and freezing dense networks. *IEEE Transactions on Cognitive and Developmental Systems*, 2020.

[170] Yi Wu, Jongwoo Lim, and Ming-Hsuan Yang. Online object tracking: A benchmark. In *Computer Vision and Pattern Recognition*, 2013.

[171] Matej Kristan, Jiri Matas, Aleš Leonardis, Tomas Vojir, Roman Pflugfelder, Gustavo Fernandez, Georg Nebehay, Fatih Porikli, and Luka Čehovin. A novel performance evaluation methodology for single-target trackers. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 38(11):2137–2155, Nov 2016.

[172] Yi Wu, Jongwoo Lim, and Ming-Hsuan Yang. Object tracking benchmark. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 37(9):1834–1848, 2015.

[173] Heng Fan, Liting Lin, Fan Yang, Peng Chu, Ge Deng, Sijia Yu, Hexin Bai, Yong Xu, Chunyuan Liao, and Haibin Ling.

Lasot: A high-quality benchmark for large-scale single object tracking. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5374–5383, 2019.

[174] Matthias Muller, Adel Bibi, Silvio Giancola, Salman Alsubaihi, and Bernard Ghanem. Trackingnet: A large-scale dataset and benchmark for object tracking in the wild. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 300–317, 2018.

[175] Pengpeng Liang, Erik Blasch, and Haibin Ling. Encoding color information for visual tracking: Algorithms and benchmark. *IEEE Transactions on Image Processing*, 24(12):5630–5644, 2015.

[176] Lianghua Huang, Xin Zhao, and Kaiqi Huang. Got-10k: A large high-diversity benchmark for generic object tracking in the wild. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2019.

[177] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *Int. journal of computer vision*, 115(3):211–252, 2015.

[178] Arnold WM Smeulders, Dung M Chu, Rita Cucchiara, Simone Calderara, Afshin Dehghan, and Mubarak Shah. Visual tracking: An experimental survey. *IEEE transactions on pattern analysis and machine intelligence*, 36(7):1442–1468, 2013.

[179] Luka Čehovin, Aleš Leonardis, and Matej Kristan. Visual object tracking performance measures revisited. *IEEE Transactions on Image Processing*, 25(3):1261–1274, 2016.

[180] Lianghua Huang, Xin Zhao, and Kaiqi Huang. Got-10k: A large high-diversity benchmark for generic object tracking in the wild. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 43(5):1562–1577, 2021.

[181] Bineng Zhong, Bing Bai, Jun Li, Yulun Zhang, and Yun Fu. Hierarchical tracking by reinforcement learning-based searching and coarse-to-fine verifying. *IEEE Transactions on Image Processing*, 28(5):2331–2341, 2018.

[182] Guangting Wang, Chong Luo, Xiaoyan Sun, Zhiwei Xiong, and Wenjun Zeng. Tracking by instance detection: A meta-learning approach. In *Proc. of the IEEE/CVF Conf. on CVPR*, pages 6288–6297, 2020.

[183] Bo Li, Wei Wu, Qiang Wang, Fangyi Zhang, Junliang Xing, and Junjie Yan. Siamrpn++: Evolution of siamese visual tracking with very deep networks. In *Proc. of CVPR*, pages 4282–4291, 2019.

[184] Martin Danelljan, Goutam Bhat, Fahad Shahbaz Khan, and Michael Felsberg. Eco: Efficient convolution operators for tracking. In *Proc. of the IEEE Conf. on CVPR*, pages 6638–6646, 2017.

[185] Shengjie Zheng and Huan Wang. Real-time visual object tracking based on reinforcement learning with twin delayed deep deterministic algorithm. In *International Conference on Intelligent Science and Big Data Engineering*, pages 165–177. Springer, 2019.

[186] Matteo Dunnhofer, Niki Martinel, Gian Luca Foresti, and Christian Micheloni. Visual tracking by means of deep reinforcement learning and an expert demonstrator. In *Proceedings of The IEEE/CVF ICCVW*, pages 0–0, 2019.

[187] Matej Kristan et al. The visual object tracking vot2016 challenge results. Springer, 2016.

[188] Matej Kristan et al. The sixth visual object tracking vot2018 challenge results, 2018.

[189] Matej Kristan et al. The seventh visual object tracking vot2019 challenge results, 2019.

[190] Alan Lukezic, Jiri Matas, and Matej Kristan. D3s-a discriminative single shot segmentation tracker. In *Proc. of the IEEE/CVF Conf. on CVPR*, pages 7133–7142, 2020.

[191] Lichao Zhang, Abel Gonzalez-Garcia, Joost van de Weijer, Martin Danelljan, and Fahad Shahbaz Khan. Learning the model update for siamese trackers. In *Proc. of the IEEE/CVF ICCV*, pages 4010–4019, 2019.

[192] Goutam Bhat, Martin Danelljan, Luc Van Gool, and Radu Timofte. Learning discriminative model prediction for tracking. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 6182–6191, 2019.

[193] Guangting Wang, Chong Luo, Zhiwei Xiong, and Wenjun Zeng. Spm-tracker: Series-parallel matching for real-time visual object tracking. In *Proc. of the IEEE/CVF Conf. on CVPR*, pages 3643–3652, 2019.

[194] Martin Danelljan, Andreas Robinson, Fahad Shahbaz Khan, and Michael Felsberg. Beyond correlation filters: Learning continuous convolution operators for visual tracking. In *European conference on computer vision*, pages 472–488. Springer, 2016.

[195] Zhipeng Zhang, Houwen Peng, Jianlong Fu, Bing Li, and Weiming Hu. Ocean: Object-aware anchor-free tracking. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XXI 16*, pages 771–787. Springer, 2020.

[196] Goutam Bhat, Joakim Johnander, Martin Danelljan, Fahad Shahbaz Khan, and Michael Felsberg. Unveiling the power of deep tracking. In *Proc. of ECCV*, pages 483–498, 2018.

[197] Chong Sun, Dong Wang, Huchuan Lu, and Ming-Hsuan Yang. Correlation tracking via joint discrimination and reliability learning. In *Proc. of the IEEE Conf. on CVPR*, pages 489–497, 2018.

[198] Qiang Wang, Li Zhang, Luca Bertinetto, Weiming Hu, and Philip HS Torr. Fast online object tracking and segmentation: A unifying approach. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1328–1338, 2019.

[199] Guangting Wang, Chong Luo, Xiaoyan Sun, Zhiwei Xiong, and Wenjun Zeng. Tracking by instance detection: A meta-learning approach. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6288–6297, 2020.

[200] Heng Fan and Haibin Ling. Siamese cascaded region proposal networks for real-time visual tracking. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7952–7961, 2019.

[201] Shengjing Tian, Xiuping Liu, Meng Liu, Shuhua Li, and Baocai Yin. Siamese tracking network with informative enhanced loss. *IEEE Transactions on Multimedia*, 2020.

[202] Martin Danelljan, Goutam Bhat, Fahad Shahbaz Khan, and Michael Felsberg. Eco: Efficient convolution operators for tracking. In *Proc. of the Conf. on computer vision and pattern recognition*, pages 6638–6646, 2017.

[203] Tianyang Xu, Zhen-Hua Feng, Xiao-Jun Wu, and Josef Kittler. Joint group feature selection and discriminative filter learning for robust visual object tracking. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 7950–7960, 2019.

[204] Mingxin Jiang, Tao Hai, Zhigeng Pan, Haiyan Wang, Yinjie Jia, and Chao Deng. Multi-agent deep reinforcement learning for multi-object tracker. *IEEE Access*, 7:32400–32407, 2019.

[205] Laëtitia Matignon, Guillaume J Laurent, and Nadine Le Fort-Piat. Reward function and initial values: Better choices for accelerated goal-directed reinforcement learning. In *International Conference on Artificial Neural Networks*, pages 840–849. Springer, 2006.

[206] Qi Feng, Vitaly Ablavsky, Qinxun Bai, Guorong Li, and Stan Sclaroff. Real-time visual object tracking with natural

language description. In *The IEEE Winter Conference on Applications of Computer Vision*, pages 700–709, 2020.

[207] Richard S. Sutton Shangtong Zhang. A deeper look at experience replay. In *arXiv:1712.01275*.

[208] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015.

[209] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Hasselt, Marc Lanctot, and Nando Freitas. Dueling network architectures for deep reinforcement learning. In *International conference on machine learning*, pages 1995–2003, 2016.

[210] Matteo Hessel, Joseph Modayil, Hado Van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. Rainbow: Combining improvements in deep reinforcement learning. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[211] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015.

[212] Marc G Bellemare, Will Dabney, and Rémi Munos. A distributional perspective on reinforcement learning. *arXiv preprint arXiv:1707.06887*, 2017.

[213] Meire Fortunato, Mohammad Gheshlaghi Azar, Bilal Piot, Jacob Menick, Ian Osband, Alex Graves, Vlad Mnih, Remi

Munos, Demis Hassabis, Olivier Pietquin, et al. Noisy networks for exploration. *arXiv preprint arXiv:1706.10295*, 2017.

[214] John Schulman, Xi Chen, and Pieter Abbeel. Equivalence between policy gradients and soft q-learning. *arXiv preprint arXiv:1704.06440*, 2017.

[215] Vijay R Konda and John N Tsitsiklis. Actor-critic algorithms. In *Advances in neural information processing systems*, pages 1008–1014, 2000.

[216] Shalabh Bhatnagar, Mohammad Ghavamzadeh, Mark Lee, and Richard S Sutton. Incremental natural actor-critic algorithms. In *Advances in neural information processing systems*, pages 105–112, 2008.

[217] Simone Parisi, Voot Tangkaratt, Jan Peters, and Mohammad Emtiyaz Khan. Td-regularized actor-critic methods. *Machine Learning*, 108(8-9):1467–1501, 2019.

[218] Pickle - python object serialization. `https://github.com/python/cpython/blob/3.11/Lib/pickle.py`.

[219] Numpy. `https://numpy.org/`.

[220] Pytorch. `https://pytorch.org/`.

[221] Yann LeCun, Yoshua Bengio, et al. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361(10):1995, 1995.

[222] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[223] Jeremy Howard. Imagenette. `https://github.com/fastai/imagenette/`.

[224] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115:211–252, 2015.