Ph.D. DISSERTATION

Robust Visual Tracking via Learned Model Adaptation

학습된 모델 갱신 기법을 사용한 강인한 물체 추적

BY

JANGHOON CHOI

February 2021

DEPARTMENT OF ELECTRICAL ENGINEERING AND
COMPUTER SCIENCE
COLLEGE OF ENGINEERING
SEOUL NATIONAL UNIVERSITY

Robust Visual Tracking via Learned Model Adaptation

학습된 모델 갱신 기법을 사용한 강인한 물체 추적

지도교수 이 경 무

이 논문을 공학박사 학위논문으로 제출함

2021 년 2 월

서울대학교 대학원

전기컴퓨터공학부

최 장 훈

최장훈의 공학박사 학위논문을 인준함

2021 년 2 월

위 원 장 : 한보형

부위원장 : 이경무

위 원 : 김선주

위 원 : 권준석 Junsoo

위 원 : 김영민

Abstract

In this dissertation, we address the model adaptation problem of visual tracking algorithms. Conventional tracking algorithms regard the visual tracking problem as a *tracking-by-detection* problem, which can be solved by formulating a target-specific detection model at the initial frame of a given video, and evaluating the model for the subsequent video frames. However, various challenges are associated with the model due to changes in circumstances such as target deformation, scale change, occlusion, illumination change, background clutter, etc. To deal with the aforementioned challenges, conventional tracking algorithms incorporate a model adaptation strategy to provide the model with new information regarding the target appearance and background distractor objects. Nonetheless, since these approaches are often conducted on a handful of self-labeled training examples through solving an optimization task involving hand-crafted regularization schemes, the risk of overfitting and error accumulation persist throughout the course of the tracking process.

In order to address the aforementioned problems, we introduce novel approaches to the model adaptation strategy for the visual tracking problem. Three types of model adaptation approaches are proposed, based on the following: (1) reinforcement learning based exemplar selection, (2) deep meta-learning based feature space update, (3) deep adaptive continual meta-learning based adaptation. The proposed approaches introduce deep neural network based meta-learners that can handle vari-

ous scenes and circumstances with reduced overfitting and error accumulation, while the meta-learners are designed to be light-weight and can achieve real-time speeds for the overall visual tracking framework.

First, we propose a deep reinforcement learning based exemplar selection method that incorporates a policy network for its meta-learner. The policy network is trained to make decisions on selecting the adequate target exemplar that can be used to locate the target given a scene. Next, a deep meta-learning based method, which utilizes a meta-learner network to construct the target-specific feature space using the loss gradient information, is proposed. The meta-learner network provides the tracker with new information in the form of adaptive weights and channel attention. Finally, a deep continual meta-learning based method simultaneously models the initial and online adaptations under the adaptive continual meta-learning framework. The meta-learner is trained to adaptively regulate the learning process where the tracker can choose between learning new examples and retaining the previous knowledge.

Applying the proposed methods to visual tracking algorithms, significant performance gains are achieved and the effectiveness is validated by the extensive experimental evaluations and component-wise ablation analyses. Additionally, comparisons on well-known, widely used visual tracking benchmarks demonstrate the competitive performance against other state-of-the-art tracking algorithms, while efficiently running at real-time speeds.

Key words: Visual tracking, Object tracking, Tracking-by-detection, Reinforcement learning, Meta learning, Continual learning, Semi-supervised learning

Student number: 2013-20896

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Visual tracking, with practical applications such as automated surveillance, robotics, AR/VR applications, and image stabilization, is one of the fundamental problems among the fields under computer vision research. Given initial bounding box coordinates of the target object along with the first frame of a video sequence, visual tracking algorithms aim to precisely locate the target object in the subsequent frames of the given video sequence. Tracking algorithms are designed to successfully track the target object under various challenging circumstances such as scale change, illumination change, deformation, occlusion, background clutter, motion blur, and more, as shown in Figure 1.1.

To deal with the aforementioned challenges arising from various circumstances, conventional tracking algorithms incorporate a model adaptation approach in a semi-supervised manner, where the tracker is provided with new information regarding the appearance of the target object and potential background distractor objects. Previous approaches for model adaptation include incremental linear subspace [92, 57], sparse representation [73, 127] for generative tracking methods, and ridge regres-

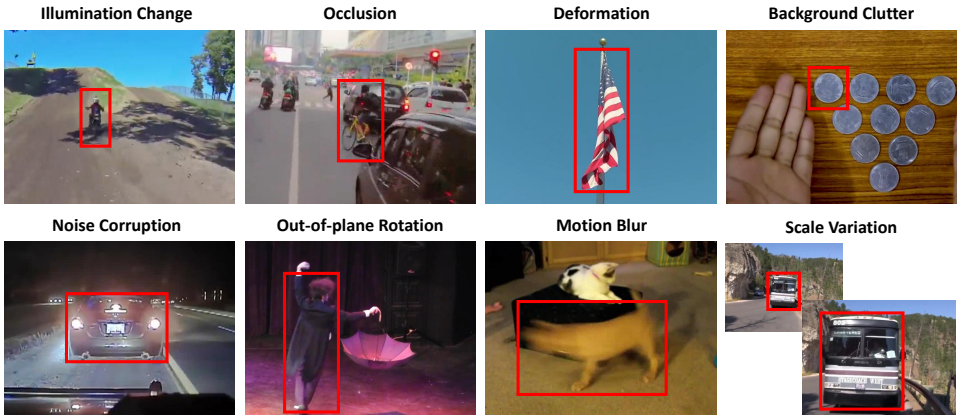


Figure 1.1: **Challenging circumstances in visual tracking problem.**

sion [44, 24], stochastic gradient descent (SGD) [82, 8, 20] for discriminative tracking methods.

However, these approaches for model adaptation are often conducted on only a handful of self-labeled training examples and the adaptation is performed through solving an optimization task involving hand-crafted regularization schemes. Thus, the risk of the model overfitting to a small number of training examples and error accumulation due to mislabeled examples persists throughout the overall course of the tracking process. Additionally, since these adaptation processes often require recursive iterations and demand extra computational load, trackers run in slower sub-real-time speeds, limiting their use in various real-time applications. To avoid these issues and achieve faster speed, recent Siamese network based trackers [7, 60, 59, 132] only rely on the powerful representation capacity of deep convolutional neural networks (CNN), completely eliminating the adaptation processes at the cost of lower performance.

In this dissertation, in order to address the aforementioned problems of model overfitting, error accumulation and slow tracking speed, novel approaches to the

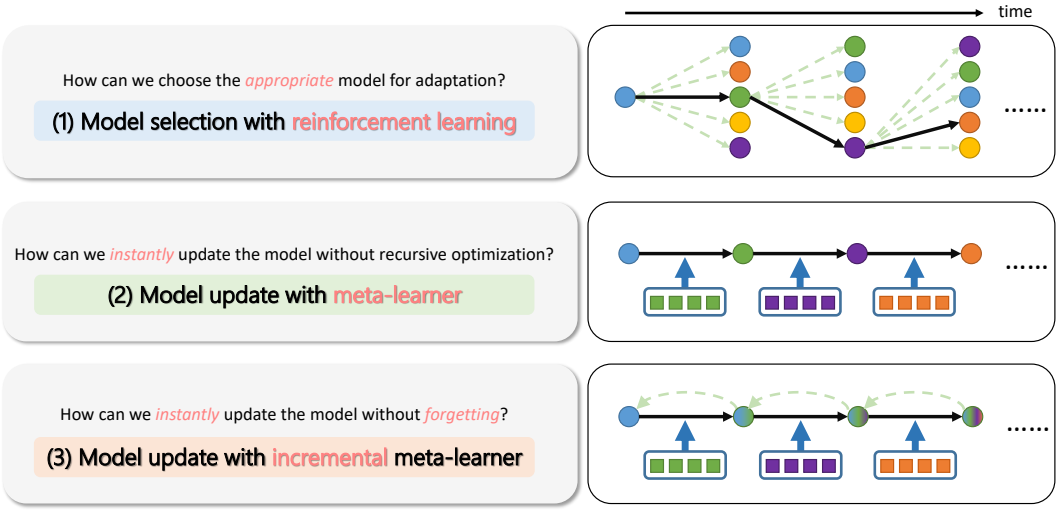


Figure 1.2: **Summary of the methods proposed in this dissertation.**

model adaptation strategy for the visual tracking problem are introduced. Three approaches for the model adaptation are proposed, based on the following: (1) reinforcement learning based exemplar model selection, (2) deep meta-learning based feature space update, (3) deep adaptive continual meta-learning based adaptation. The proposed approaches introduce external, deep neural network based meta-learner networks that can handle model adaptations under various scenes and circumstances with reduced overfitting and error accumulation, while the meta-learners are designed to be light-weight and can achieve real-time speeds for the overall visual tracking framework. A summary of the proposed methods introduced in this dissertation is shown in Figure 1.2.

In Chapter 2, a deep reinforcement learning based exemplar selection method is introduced [15]. We introduce a novel real-time visual tracking algorithm based on a template selection strategy constructed by deep reinforcement learning methods. The tracking algorithm utilizes this strategy to select the appropriate template for

tracking a given frame. The template selection strategy can be learned by utilizing a simple policy gradient method on numerous training episodes randomly generated from a tracking benchmark dataset. Our proposed reinforcement learning framework is generally applicable to other confidence map based tracking algorithms. The experimental results show that our tracking algorithm runs at a real-time speed of 43 fps, and the proposed policy network effectively decides the appropriate template for successful visual tracking.

In Chapter 3, a deep meta-learning based method, which utilizes a meta-learner network to construct the target-specific feature space using the loss gradient information, is proposed [16]. Our proposed tracking algorithm incorporates and utilizes a meta-learner network to provide the matching network with new information on the appearance of the target objects by adding a target-aware feature space. The parameters for the target-specific feature space, which are given in the form of convolutional kernels and channel attention weights, are obtained instantly from a single forward pass of the meta-learner network given loss gradient information as input. By eliminating the necessity of continuously solving complex optimization tasks in the course of tracking, experimental results demonstrate that our algorithm performs at a real-time speed while maintaining competitive performance among other state-of-the-art tracking algorithms.

In Chapter 4, the deep continual meta-learning based method simultaneously models the initial and online adaptations under the adaptive continual meta-learning framework. In contrast to conventional meta-learning based approaches that regard visual tracking as an instance detection problem with a focus on finding good weights for model initialization, we consider both the initialization and online update processes simultaneously under our adaptive continual meta-learning framework. The

proposed adaptive meta-learning strategy dynamically generates the hyperparameters needed for fast initialization and online update to achieve more robustness through adaptively regulating the learning process. In addition, our continual meta-learning approach based on a knowledge distillation scheme helps the tracker adapt to new examples while retaining its knowledge on previously seen examples. We apply our proposed framework to two deep learning-based tracking algorithms [17] to obtain noticeable performance gains and competitive results against recent state-of-the-art tracking algorithms while performing at real-time speeds.

Conclusion of the dissertation is provided in Chapter 6, where we summarize the proposed methods and their contributions, along with suggestions for possible future research directions for further improvement.

Chapter 2

Model Selection by Reinforcement Learning

In this chapter, we introduce a model selection method based on reinforcement learning and policy gradient method for visual tracking algorithms. We aim to solve the issues of previous deep neural network based tracking algorithms where erroneous model adaptation process causes tracker drift and recursive adaptation processes hinder real-time applications. We propose a fast adaptation process based on learned model selection strategy from the policy network, where the overall framework can achieve real-time tracking speed. The policy network is trained to choose the adequate target template that is expected to finish a simulated tracking episode without failure from tracker drift.

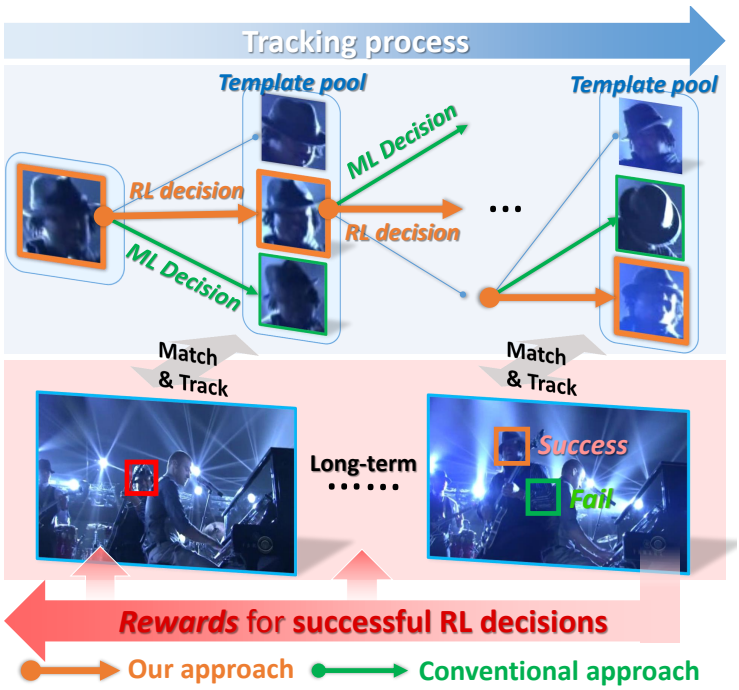


Figure 2.1: **Motivation for the proposed visual tracking algorithm.** Our tracking algorithm formulates the visual tracking problem as a consecutive decision making task that can be self-learned through a reinforcement learning scheme. Rather than simply using the template with the maximum likelihood (ML decision), our algorithm strategically chooses the best template from the template pool in terms of localization and long-term success (RL decision).

2.1 Introduction

For several years, visual tracking problem has been regarded as a *tracking-by-detection* problem, where the visual tracking task is formulated as an object detection task performed on consecutive video frames. Tracking algorithm is often composed of a combination of appearance and motion models of the object. Especially, the appearance model is carefully designed to be robust to numerous appearance variations of the target object, where common challenges arise from changes in illumination, motion blur, deformation and occlusion from surrounding objects [62].

To solve the aforementioned challenges, two approaches are mainly utilized to

cover the appearance variations of the target object. One approach is to update the appearance model of the target on-line in the tracking process [52, 57, 92, 127, 42, 35], gaining new examples on the way. This approach considers the visual tracking problem as a semi-supervised learning task where the initial sample is labeled while other samples are not. However, inaccurate and erroneous update often causes the tracker to fail and drift to the background [72, 125]. The other approach is to utilize a feature representation scheme that is more robust to appearance perturbations while maintaining the discriminability between the target object and background objects [112, 111, 82, 110, 40]. This approach shares a common objective of other computer vision tasks such as object detection and semantic segmentation.

Recently, with growing attention on deep neural networks, especially convolutional neural networks (CNN) [58], there have been several approaches to utilize the powerful representation capabilities of CNNs for the visual tracking task. These methods showed successful results in covering the target appearance variations in short video segments. However, we also focus on the other aspect of visual tracking. Our proposed real-time visual tracking algorithm aims to utilize the deep neural network for revising the on-line update by making decisions concerning which template is the most adequate for localizing the target in a new frame. Our method formulates the visual tracking task as a consecutive decision making process where given past target appearance samples, the tracker has to decide which sample is the best for localizing the target for a new frame. Figure 2.1 illustrates the motivation of this research.

While there are large image datasets such as ImageNet [93] with ground truth labels available for obtaining a powerful feature representation under supervised learning environment, on-line update and selection of the target appearance model

for visual tracking should be adequately tuned according to the tracking environment and the capacity of the feature representation that is used. This results in an absence of explicit labels on when and how to update the appearance model, which makes supervised learning infeasible. To resolve this problem, we adopt a reinforcement learning environment where given sequential states, an agent is prompted to make actions that can maximize the future reward. To achieve this learning task, we adopt deep neural networks for efficient state representation. Then we utilize policy gradient methods used in [116, 74] and experience replay memory as in [75], motivated by their recent success in playing the game of Go and ATARI video games in [95, 76]. We train our policy network using randomly generated episodes from VOT-2015 tracking benchmark dataset [11]. We build our tracking algorithm based on a Siamese matching network by [13] for its simplicity and real-time tracking speed while having a powerful representation capacity. To our knowledge, our work is one of the first to utilize a deep reinforcement learning methodology for on-line update in visual tracking.

2.2 Related Work

Conventional visual tracking algorithms can be largely categorized into two approaches. One approach constructs a generative model from previously observed samples and utilizes this model to find the region in the new frame where it can be described by the model best. The other uses a discriminative model where a classifier is learned to distinguish the target object region from the surrounding background region.

Generative approaches for visual tracking often utilize sparse representation as

in [128, 127, 73] or linear subspace for incremental learning as in [57, 92]. Using these criteria, they try to find the target region where it can be described by the model. The model is constructed from target appearance samples collected from previously tracked frames. [92] uses principal component analysis on previous templates to construct a incremental subspace that can be used to reconstruct the target appearance. The target is localized by finding the location with the lowest reconstruction error. Discriminative approaches for visual tracking often utilize classifiers as in [4, 35, 52] or correlation filters as in [42, 69, 45, 25]. These approaches try to build a model that can distinguish the target appearance from the background region by using classification or regression. The model is trained from target and background appearance samples together. [35] uses structured SVM to find the transformation vectors for patches obtained from the vicinity of the target, solving the label ambiguity problem of the binary classification assumption. Other than the generative and discriminative methods, there are hybrid methods as in [134, 12] that aim to utilize the advantages of both models. [12] adopted two components for the appearance model; with one descriptive and the other discriminative. Both components are integrated through a single optimization task.

Recently, there have been approaches to utilize deep representations for the visual tracking task. Convolutional neural networks (CNN) [58] have shown outstanding performance in a wide range of computer vision applications including image classification [56], object detection [91] and much more. Their powerful representation capacity motivated visual tracking approaches such as [112, 69, 111, 82, 110]. [112] was the first to introduce deep representation learning to visual tracking problem. They build a stacked denoising autoencoder and utilize its intermediate representation for visual tracking. In [69], hierarchical correlation filters learned on the feature

maps of VGG-19 network [96] are efficiently integrated. [110] also utilizes the feature maps generated from the VGG network to obtain multi-level information. [82] used the structure of low-level kernels of VGG-M network [96] and trained on visual tracking datasets to obtain multi-domain representation for a robust target appearance model.

Based on deep representations, some outstanding performances were shown by using two-flow Siamese networks on stereo matching problem in [124] and patch-based image matching problem in [34]. Accordingly, approaches to solve the visual tracking problem as a patch matching problem have emerged in [104, 7, 13, 40]. [104] and [40] train the Siamese networks using videos to learn a patch similarity matching function that shares an invariant representation. [7] and [13] further expand this notion and proposes a more end-to-end approach to similarity matching where a Siamese architecture can localize an exemplar patch inside a search image using shared convolutional layers. In particular, [7] proposes a fully-convolutional architecture that adopts a cross-correlation layer to obtain invariance to spatial transitions inside the search image, lowering the complexity of the training process significantly.

However, approaches such as [40, 13] use a naive on-line update strategy that cannot revise erroneous updates and recover from heavy occlusions. Moreover, approaches [104, 7] do not update the initial template, solely relying on the representation power of the pre-trained CNN. This approach may be effective for short-term video segments with no distractors, but the tracker can be attracted towards a distractor with a similar appearance to the target. Our proposed algorithm is aimed to solve both problems of the previous approaches, by utilizing previously seen examples to adapt to the recent appearance of the target and choosing the most adequate template for localizing the target, ruling out erroneously updated templates.

There also have been some recent approaches that employ deep reinforcement learning methodology on visual tracking algorithms. [123] trained a policy network to generate actions for state transition in order to localize the target in a given frame. [99] used YouTube videos to interactively learn a Q-value function where it makes decisions for the tracker to reinitialize, update or to keep tracking with the same appearance model. However, both trackers run at 3 fps and 10 fps respectively, both lacking the speed of real-time performance. Our algorithm runs at a real-time speed of 43 fps while maintaining a competitive performance. We achieve this by incorporating more lightweight and optimized structures for matching network and policy network.

2.3 Tracking with Reinforced Decisions

In the following subsections, we first show a brief overview of our proposed visual tracking algorithm. Then we describe the details of the proposed method. We show the theoretical background for the reinforcement learning formulation. Next we describe the architectures for visual tracking and its training scheme.

2.3.1 Proposed Tracking Algorithm

Our tracking system can be divided into two parts where the first part is the matching network that produces prediction heatmaps as a result of localizing the target templates inside a given search image. And the second part is the policy network that produces the normalized scores of prediction maps obtained from the matching network. Figure 2.2 shows the overall diagram of our tracking system.

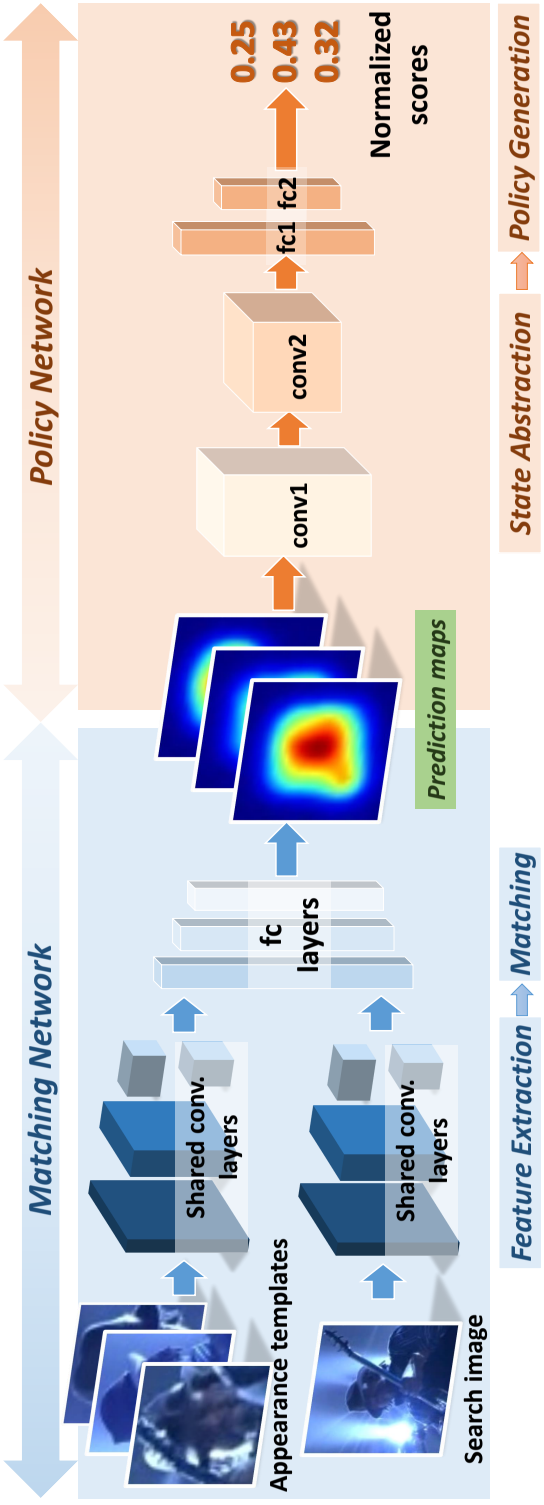


Figure 2.2: **Overall architecture of the proposed system.** Matching network is a Siamese network which consists of shared convolutional layers as feature extractors and fully connected layers for matching. Matching result is passed to the policy network where it also consists of convolutional layers for state abstraction and fully connected layers for policy generation.

Assuming the networks are trained and its weights are fixed, we can perform the visual tracking task on arbitrary sequences. For a given video frame, we crop and obtain a search image based on the target's previous bounding box information. Search image is cropped with its center location being the target's previous center location and its scale being double the target's previous scale. Using the appearance templates obtained from previously tracked frames, the matching network produces prediction maps for each appearance template. Then each of the prediction maps are fed to the policy network where it produces the normalized scores for each prediction map. The prediction map with the maximum score is chosen and its corresponding template is used to track and localize the target. The position in the prediction map with the maximum value corresponds to the next position of the target in the search image. To obtain a more precise scale and position of the target, we construct a scale pyramid of 3 images using the search image to adapt to the scale change of the target. Scale with the maximum response value is chosen. Then we additionally use 4 shifted versions of the search image to obtain a more fine-level position of the target. Size of the x, y-axis shift are fixed proportionally to the target width/height respectively. Maximum positions in each response map are found and we calculate the mean position as the final location of the target.

Appearance templates are obtained on a regular time interval during the tracking process. We intentionally use this simplistic method for updating the template pool in order to promote robustness in the decision making process. This encourages the policy network to be trained on various situations where the policy network is expected to avoid choosing an outlier template (i.e. dark, blurry, occluded template) which may be present in the template pool. The overall flow of our tracking algorithm is described in algorithm 1.

Algorithm 1: Visual tracking with reinforced decision making

input : Tracking sequence of length L
Initial target location x_0
output: Tracked target locations x_t

// For every frame in a sequence
for $t = 1$ *to* L **do**
  // For all N templates
  for $i = 1$ *to* N **do**
    Produce prediction maps s_i with each template i;
    Obtain normalized scores for each prediction map using policy network $\pi(a_i|s_i;\theta)$;
  end
  Find the prediction map s_t with maximum score, corresponding template is chosen for localizing the target;
  Construct scale pyramid, obtain response maps and choose best scale with maximum value;
  Localize the target x_t with shifted search images, choose mean position as target location x_t;
  Add a template to template pool every K frames, discarding the oldest one;
end

2.3.2 Reinforcement Learning Overview and Application to Visual Tracking

We consider a general reinforcement learning setting where the agent interacts with an environment through sequential states, actions and rewards. Given an environment $\mathcal{E}$ and its representative state s_t at time step t, an agent must perform an action a_t selected from a set $\mathcal{A}$ of every possible actions. Action a_t is determined by the policy $\pi(a_t|s_t)$ where action a_t can be chosen with deterministic or stochastic manner. In return for the action, the agent receives a scalar reward r_t and observes the next state s_{t+1}. This recurrent process continues until the agent reaches a terminal state. The goal of the agent is to select actions that maximizes the discounted sum of expected future rewards, where we define the action-value function as $Q^{\pi}(s, a)$

Algorithm 2: Training the policy network for a single episode

input : Randomly generated episode of length L, Policy network weights θ^-
output: Updated policy network weights θ^+

// For every frame in an episode
for $t = 1$ *to* L **do**
  // For all N templates
  for $i = 1$ *to* N **do**
    Produce prediction maps s_t with each template i;
    Obtain normalized scores for each prediction map using policy network $\pi(a_i|s_t;\theta^-)$;
  end
  Choose some $a_t \in \{a_i, ..., a_N\}$ stochastically, with probabilities proportional to the normalized scores;
  Obtain update gradient $\nabla_\theta \log \pi(a_t|s_t;\theta^-)$;
  Accumulate gradients according to eq. 2.2, obtain $\Delta\theta$;
  Localize the target position x_t as in Alg. 1;
  Add a template to template pool every K frames, discarding an oldest template;
end
if *episode successful* **then**
  Update weights $\theta^+ = \theta^- + \Delta\theta$;
else
  Update weights $\theta^+ = \theta^- - \Delta\theta$;
end
Obtain $4N$ samples from experience replay memory, calculate gradients and add to θ^+;

[100].

To achieve the goal mentioned above, there are mainly two approaches to reinforcement learning; value-based methods and policy-based methods. Value-based method assumes that there exists an optimal action-value function $Q^*(s,a) = \max_\pi Q^\pi(s,a)$ that gives the maximum action-value for a state-action pair, given some implicit policy (e.g. ϵ-greedy) [115].

On the other hand, policy-based methods [101] aim to directly model the policy function $\pi(a|s;\theta)$ without the assumption of intermediate action-value function,

removing the need of evaluating the values of possible actions for a given state. Approximation of the policy function can be achieved by maximizing the objective return function R_t using stochastic gradient ascent algorithms. One simplest example of the method is the REINFORCE algorithm introduced in [116] where gradient ascent is used on expected reward $R_t = \mathbb{E}[r_t]$ as in

$$\Delta\theta = \alpha \nabla_\theta \log \pi(a_t|s_t;\theta) r_t, \tag{2.1}$$

where α is the learning rate. First, actions are sampled from the policy distribution π, then the performed actions are evaluated and rewards are given from interacting with the environment. Using this information, we can refine the policy through a parameter update. Samples from an updated policy is expected to give us a higher reward. For our work, we use a variant of policy-based reinforcement learning method since it is commonly known to have better convergence properties and capability of learning stochastic policies [74].

To apply this reinforcement learning method to our visual tracking environment, we define state s_t of the tracker as the overall combination of prediction maps which are obtained from the matching network using the corresponding appearance templates gained in the course of tracking. The action a_t is defined as selecting a single template from current template pool to locate the target in a given frame. This can be simulated by drawing a sample from a discrete probability distribution of normalized scores where each score is produced by the policy network using the corresponding prediction map. At training time, we can train the policy network to assign higher scores to prediction maps (templates) that result in successful tracking using a rewarding scheme.

Reward r_t is given at the end of a tracking episode depending on success or failure of a tracking episode. Positive reward will be given when the tracker successfully tracks the target, producing a bounding box overlap score over a predefined threshold. This will encourage the policy network to perform more actions (choosing templates) that may result in a successful tracking episode. On the other hand, negative reward will be given when the tracker loses the target as a result of performing a chain of poor actions. This will help the tracker to avoid performing those actions in the future, thus resulting in more successful tracking episodes hereafter. At test time, we select the prediction map (and its corresponding appearance template) with the highest normalized score to locate the target in a given frame.

2.3.3 Network Architectures

Architecture of the Matching Network: We borrow the Siamese architecture design from [13], using it as our baseline tracker. The Siamese architecture has 2 input branches and uses 3 shared convolutional layers for extracting the common representations from the object patch and the search patch. Then these features are concatenated and fed to 3 fully connected layers to produce a Gaussian prediction map, where the maximum point is the relative location of the object patch inside the search patch.

Architecture of the Policy Network: Our policy network sees the output prediction maps produced by the matching network and makes the decision whether the matching result is reliable or not. By following this decision, we can always choose the most appropriate template for locating the target object in a given frame. The policy network consists of 2 convolutional layers to produce an adequate representation of the state and 2 fully connected layers for deciding whether this state is reliable

or not. Then the outputs are fed through sigmoid function to produce probabilities. Finally, we choose the activation with the highest value and its corresponding template as the best candidate for tracking.

2.3.4 Training the Policy Network

To train the policy network $\pi(a|s;\theta)$ introduced above, we use a variant of REINFORCE algorithm with accumulated policy gradients. We randomly generate numerous tracking episodes with varying lengths from VOT-2015 [11] video dataset. Then we perform tracking on each training episodes with stochastically sampled action roll-outs produced by the policy network to ensure exploration of state space. For each decisions in an episode, we temporarily assume each decision was optimal and perform backpropagation to obtain gradients for all weights inside the policy network. We accumulate these gradients for all decisions in a single episode as in

$$\Delta\theta = \alpha \sum_{t=1}^{L} \nabla_\theta \log \pi(a_t|s_t;\theta)\beta^{L-t}, \tag{2.2}$$

where L is the length of an episode, α is the learning rate and $\beta \in (0, 1]$ is a discounting factor inserted to give more weight to decisions made later in the episode. If an episode terminates, weights in the policy network are updated according to the success or failure of that episode. If an episode was successful, gradient is updated accordingly. If an episode was a failure, negative gradient is applied. The overall algorithm for training the policy network for a single episode is described in algorithm 2.

We also keep an experience replay memory of state-action-reward gained from previous episodes. When gradients are applied to the policy network after an episode,

Table 2.1: Architecture of the matching network. The convolutional layer parameters are denoted as conv(kernel size, stride)-(number of channels) and fully connected layer parameters are denoted as fc-(number of units). Max pooling layer is denoted as pool-(kernel size, stride)

<table>
<tr><td colspan="2">input (48 × 48, 120 × 120 RGB image)</td></tr>
<tr><td colspan="2">conv7,3-16</td></tr>
<tr><td colspan="2">pool2,2, relu</td></tr>
<tr><td>conv3,1-32</td><td rowspan="4">conv1,1-4</td></tr>
<tr><td>pool2,2, relu</td></tr>
<tr><td>conv3,1-64</td></tr>
<tr><td>pool2,2</td></tr>
<tr><td colspan="2">relu, reshape, concat</td></tr>
<tr><td colspan="2">fc-2048, relu</td></tr>
<tr><td colspan="2">fc-2048, relu</td></tr>
<tr><td colspan="2">fc-961</td></tr>
<tr><td colspan="2">sigmoid, reshape</td></tr>
<tr><td colspan="2">output (31 × 31, prediction map)</td></tr>
</table>

experiences are randomly sampled from the experience replay memory and applied concurrently. Successful experiences and failure experiences are kept separately in each replay memories. At every training step, total update gradient $\Delta\theta$ is a sum of $5L$ gradients where L gradients are obtained by decisions from a single episode and $4L$ gradients are sampled from the decisions in the experience memory ($2L$ from each success and failure memory). By using these sampled experiences from the experience replay memory, we can remove the correlation in incoming data sequence and reduce the variance of the update to obtain more stable convergence.

2.4 Experiments

2.4.1 Implementation Details

Matching network parameters: Input to the matching network are 48×48 appearance template and 120×120 search image. Matching network consists of the three convolutional layers with a skip connection layer connecting the first and the third layer. Activations are then fed to the 3 fully connected layers with dropout

rate of 0.8, followed by a sigmoid function. Output is a 31×31 prediction map. More detailed network parameters are shown in table 2.1.

Policy network parameters: Input to the policy network are 31×31 prediction maps. The first convolutional layer has a 5×5 sized kernel with 4 output channels and it is applied with a stride of 3, then the activations are 2×2 max-pooled. The second convolutional layer has a 3×3 sized kernel with 8 output channels and it is applied with a stride of 1. Then the activations are fed to fully-connected layers, each with 128 hidden activations and 1 activation. For fully-connected layers, dropout regularization [98] is used with keep probability of 0.7. All layers are initialized from Gaussian normal distribution with zero mean and variance of 0.1 and each convolutional layer is followed by rectified linear unit (ReLU) activation functions.

Training parameters: To train the matching network, batch size of 64 is sampled from the ImageNet [93] dataset. For optimization, Adam optimizer [54] with learning rate of 10^{-4} is used. For the policy network, Adagrad optimizer [28] with learning rate of 10^{-4} is used and $\beta = 0.95$ is used. We train our policy network using 50,000 episodes randomly sampled from the VOT-2015 [11] benchmark dataset. Overlapping sequences with OTB dataset were removed before training. Length of each episode is between 30 and 300 frames and a new template is added to the template pool every 50 frames. Success or failure of an episode is determined by the mean intersection-over-union (IoU) ratio of last 20 bounding box predictions compared to the ground truth bounding boxes. If the mean IoU is under 0.2, we consider the episode as a failure. To lower the variance of each update, we keep an experience replay memory for 5000 successful samples and 5000 failure samples. Setting the episode length too short will result a deficiency in number of failure experiences to

learn from. For each update, 40 samples are sampled from each experience replay memory and gradient is applied concurrently.

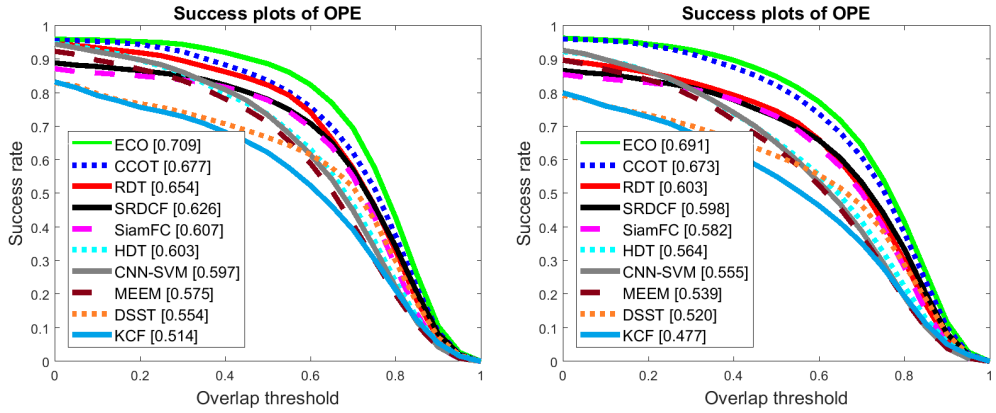
Tracking parameters: For practical reasons, we also average the location predictions obtained from 4 slightly shifted (upward, downward, left and right) search images to obtain a more accurate localization. Shift sizes are fixed to 20% of the target's width/height. Additionally, to cover the scale space, we use 3 scaled versions of the search image to find the best scale that fits the template. Scale parameters used are $1.05, 1.00$ and 1.05^{-1}. We used a maximum of 4 templates including the initial template for tracking, and template pool is updated every 50 frames, replacing the oldest template. Increasing the number of maximum templates beyond this limit did not show noticeable performance gain since the tracker tended to utilize a small subset of templates that were updated more recently. This tendency is coherent with the results shown in [125] where number of actively utilized snapshots was limited. *All parameters were fixed* throughout the evaluation of the benchmark dataset.

Implementation environment: We implement our tracker in Python using TensorFlow [1] library. The implementation runs on an Intel Core i7-4790K 4GHz CPU with 24GB of RAM and the neural network is computed and trained on GeForce GTX TITAN X GPU with 12GB of VRAM. Our implemented tracker runs at an average of 43 frames per second (FPS) on OTB-2015 [119] video dataset.

2.4.2 Evaluation on OTB dataset

2.4.2.1 Quantitative Results

Object tracking benchmark (OTB-2015) [119] is a well-known visual tracking benchmark dataset that contains a total of 100 fully annotated sequences. OTB-2013 is a



(a) OTB-2013 Result (b) OTB-2015 Result

Figure 2.3: OPE result comparison on (a) OTB-2013 and (b) OTB-2015 benchmark dataset [119]. The numbers in the legend box indicate the average area-under-curve (AUC) scores for each tracker.

subset of the OTB dataset which contains 50 selected sequences from the original dataset. We compare our tracking algorithm with 9 other tracking algorithms, including 5 real-time algorithms. SiamFC [7], SRDCF [23], HDT [88], CNN-SVM [44], MEEM [125], DSST [21], KCF [42], ECO [19] and CCOT [24] tracking algorithms are used for comparison. Success plots for both OTB-2013 and OTB-2015 sequences are shown on figure 2.3 where the proposed algorithm is denoted as RDT. Success rate evaluation metric is calculated by comparing the predicted bounding boxes with the ground truth bounding boxes to obtain the IoU scores and measuring the proportion of scores larger than a given threshold value. Final score is calculated by measuring the area-under-curve (AUC) for each tracker. The result shows that our proposed algorithm achieves a competitive performance alongside with other trackers, despite using a simple patch matching framework. Our tracker also performs at a real-time speed of 43 FPS with relatively high performance (Table 2.2), compared to other deep representation based trackers such as HDT [88] running at 10 FPS, SRDCF [23] running at 5 FPS and ECO [19] running at 8 FPS.. The real-time processing

Table 2.2: Tracking performance on OTB-2015 dataset and speed comparison between trackers.

Tracker	**AUC**	**FPS**
ECO [19]	0.691	8
RDT (Ours)	**0.603**	**43**
SRDCF [23]	0.598	5
CSR-DCF [68]	0.587	13
SiamFC [7]	0.582	58
CFNet-conv2 [106]	0.568	75
HDT [88]	0.564	10
DSST [21]	0.520	24
KCF [42]	0.477	172
GOTURN [40]	0.427	100
TLD [52]	0.406	20

Table 2.3: Tracking performance on OTB-2015 benchmark dataset under different template update intervals (frames).

Frames	20	40	50	80	100	150	200	300
AUC	0.615	0.605	0.603	0.595	0.599	0.596	0.587	0.586

speed of our tracker is a favorable characteristic for training the policy network using numerous training episodes.

We further analyze the performance of our tracker for 8 different challenge attributes labeled for each sequence, where sequences with background clutter, illumination variation, in-plane rotation, low resolution, occlusion, out-of-plane rotation, out of view and scale variation are evaluated. As shown in figure 2.4, our tracker shows competitive results on most of the attributes compared to the other trackers.

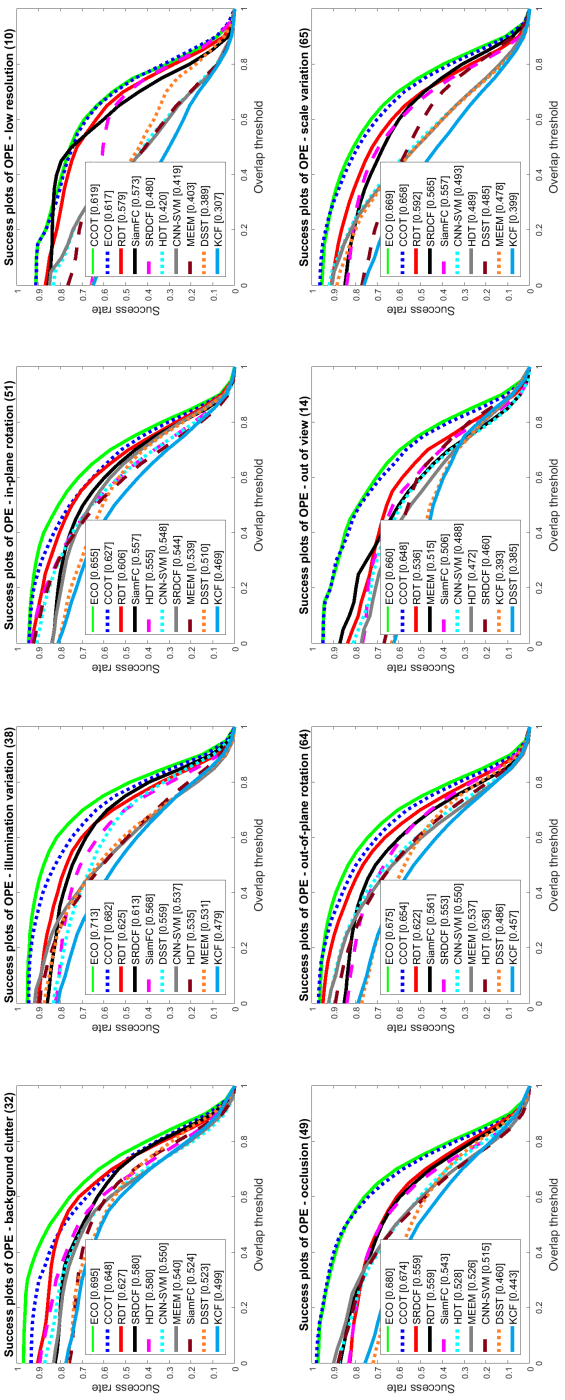
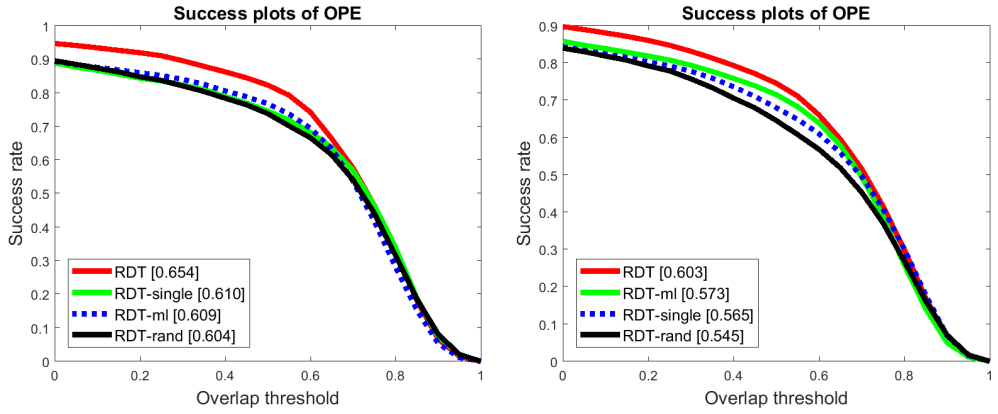


Figure 2.4: Success plots for 8 challenge attributes: background clutter, illumination variation, in-plane rotation, low resolution, occlusion, out-of-plane rotation, out of view and scale variation.



(a) OTB-2013 Result (b) OTB-2015 Result

Figure 2.5: OPE result comparisons of (a) OTB-2013 and (b) OTB-2015 benchmark dataset [119] for the internal comparisons.

To show the effectiveness of our reinforced template selection strategy, we also perform internal comparisons between different baselines of our algorithm. Figure 2.5 shows the OPE plots and Table 2.4 shows the SRE and TRE results between four algorithms where RDT is the original algorithm, RDT-rand is a variant where the template is selected at random, RDT-ml is a variant where the template with the maximum likelihood score is selected and RDT-single is a variant where only the original template is used with no update. We were able to obtain a OPE performance gain of roughly 10% for both OTB-2013 and OTB-2015 sequences by using the proposed template selection strategy, proving that our policy network chooses the more adequate template for tracking a given frame.

The performance gain in TRE is smaller in comparison to OPE and SRE results. However, by the nature of TRE evaluation where an original OTB sequence is divided into 20 uniform time-segments and tested 20 times with initialization from each segment, overall performance is measured with sequences that are much shorter than their original sequences. Since our algorithm's performance gain comes from the

Table 2.4: SRE and TRE tracking performances measured in AUC on OTB-2015 benchmark dataset for internal comparisons.

	RDT	**RDT-ml**	**RDT-rand**	**RDT-single**
SRE	0.601	0.565	0.544	0.543
TRE	0.646	0.625	0.605	0.615

appropriate update strategy in long-term sequences (i.e. `blurFace, basketball, panda, human6, faceOcc1, girl2, blurCar1, doll, ...`), smaller performance gain in short-term sequences is reasonable.

We also perform an additional experiment with different update intervals (frames) where the results are shown in Table 2.3. Performance is measured by AUC tested on OTB-2015 dataset. Decreasing the update interval gives some performance gain while increasing the interval results loss in performance. We speculate that templates that are more recent are effective in localizing the target more precisely, while older templates are less effective and thus less utilized. This is consistent with our experiment with increased template pool where increasing the template over certain limit did not show noticeable performance gain.

2.4.2.2 Qualitative Results

Figure 2.7 shows the snapshots of tracking results produced by the proposed algorithm with SiamFC [7], SRDCF [23], HDT [88], CNN-SVM [44] and DSST [21]. Trackers were tested on some challenging OTB-2015 sequences (*box, carScale, ironman, jump, matrix, singer1, soccer* and *bolt2*) where selected frame numbers are denoted in yellow on the top-left corners respectively. Our proposed tracking algorithm performs robustly, without losing track of the target under challenging conditions such as occlusion in *box* and *soccer*, scale change in *carScale*, *singer1* and *jump*, illumination variation in *ironman* and *matrix*. From the qualitative results, it

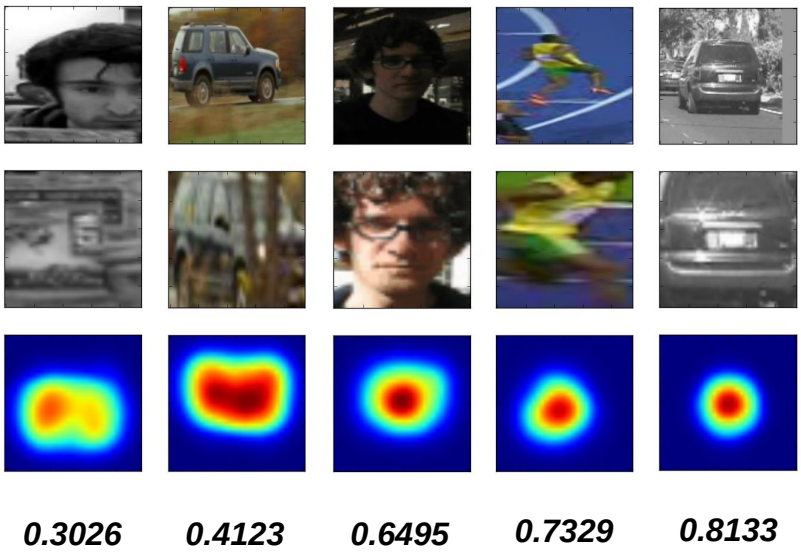


Figure 2.6: Example input search image (1st row), template image (2nd row), prediction maps (3rd row) and their corresponding output scores of the policy network (4th row). Outputs are shown in values [0,1]

is shown that our tracker successfully utilizes both the deep representation power and the template selection strategy for tracking the target.

We also show some example input/output pairs for the policy network in figure 2.6 to show what the policy network has learned from numerous tracking episodes. Interestingly, the policy network has self-learned to avoid ambiguous decisions with high uncertainty (left) while preferring decisions with low ambiguity (right) by assigning a higher score on the prediction map. Our tracker avoids utilizing erroneously updated templates (e.g. occluded samples) by choosing the best template in terms of uncertainty, resulting a more successful tracking episode.

Figure 2.7: Qualitative results of the proposed method on challenging sequences from OTB benchmark dataset (in vertical order, *box, carScale, ironman, jump, matrix, singer1, soccer* and *bolt2*)

2.5 Summary

In this chapter, we proposed a novel tracking algorithm based on a template selection strategy constructed by deep reinforcement learning methods, especially policy gradient methods. Our goal was to construct a policy network that can choose the appropriate template from a template pool for tracking an arbitrary frame where the policy network is trained from numerous training episodes randomly generated from a tracking benchmark dataset. Experimental results show that we achieved a noteworthy performance gain in tracking under challenging scenarios, proving that our learned policy effectively chooses the appearance template that is more appropriate for a given tracking scenario. Our algorithm also performs at a real-time speed of 43 fps while maintaining a competitive performance compared to other real-time visual tracking algorithms.

Chapter 3

Model Update by Meta-Learning

In the previous chapter, we introduced a reinforcement learning based model selection strategy for visual tracking, where a model is represented by a target template acquired throughout the course of tracking. However, since a target template only represents a single positive example of the target object, the model lacks in encompassing large variations of the target appearance as well as crucial information on background objects.

In this chapter, to solve the aforementioned issues, we propose a meta-learning based approach to update the target feature representation to better discriminate the target and the background regions. The meta-learner provides the tracker with updated feature space using the loss gradient information while achieving real-time speed. The meta-learner network is trained to eliminate the recursive optimization process and additional hyperparemeter tuning with reduced overfitting to small number of online training examples.

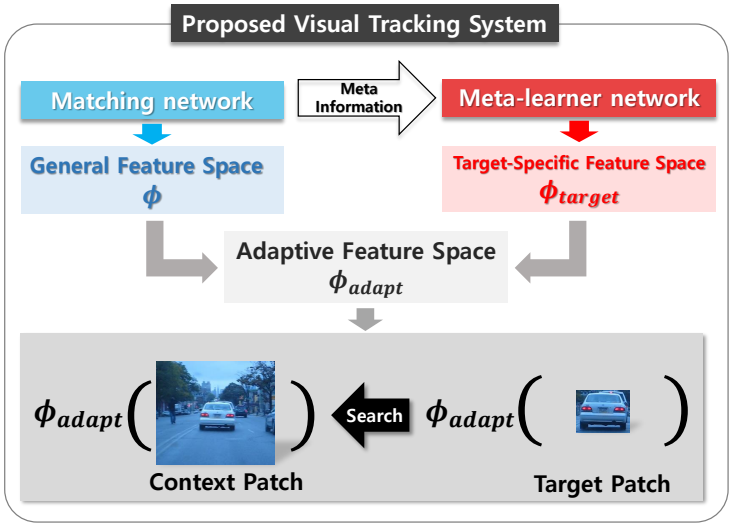


Figure 3.1: **Motivation of the proposed visual tracker.** Our framework incorporates a meta-learner network along with a matching network. The meta-learner network receives meta information from the matching network and provides the matching network with the adaptive target-specific feature space needed for robust matching and tracking.

3.1 Introduction

Recently, with the increasing use of deep learning and convolutional neural networks (CNN) [58] in computer vision applications for their rich representation power and generalization capabilities [56, 91, 96], there have been numerous studies on utilizing the rich and general feature representation of the CNNs for visual tracking task [112, 69, 82, 110, 104]. Most algorithms incorporate the deep convolutional features used in object recognition systems [56, 96, 91]. On top of these feature representations, additional classifiers or correlation filters are trained for on-line adaptation to the target object [82, 44, 110, 69, 24, 68, 22, 129].

While these methods were successful in obtaining high performance metrics in well-known benchmarks and datasets [11, 119] using deep representations, the majority of these algorithms were not designed as an integrated structure, where two

different systems (i.e. deep feature network system and target classifier systems) are built and trained separately, not being closely associated. This causes several problems when the framework is naively applied to the problem of visual tracking, where the classifier system is in constant need of being updated in order to adapt to the appearance changes of the target object, while the number of positive samples are highly limited.

Since an update operation requires solving complex optimization problems for a given objective function using methods such as stochastic gradient descent (SGD) [82], Lagrange multipliers [44], and ridge regression [44, 24, 22], most tracking algorithms with deep representations run at low speeds under 20 fps, thus making real-time applications unrealizable. Moreover, since the updates are often achieved by utilizing a handful of target appearance templates obtained in the course of tracking, while this strategy is inevitable, classifiers are prone to overfitting and losing generalization capabilities due to insufficient positive training samples. To deal with this prevalent overfitting problem, most algorithms incorporate a hand-crafted regularization term with a training hyper-parameter tuning scheme to achieve better results.

Our approach tackles the aforementioned problems by building a visual tracking system incorporating a Siamese *matching network* for target search and a *meta-learner network* for adaptive feature space update. We use a fully-convolutional Siamese network structure analogous to [7] for searching the target object in a given frame, where target search can be done fast and efficiently using the cross-correlation operations between feature maps. For the meta-learner network, we propose a parameter prediction network inspired by recent advances in the meta learning methodology for few-shot learning problems [81, 108, 32].

The proposed meta-learner network is trained to provide the matching network

with additional convolutional kernels and channel attention information so that the feature space of the matching network can be modified adaptively to adopt new appearance templates obtained in the course of tracking without overfitting. The meta-learner network only sees the gradients from the last layer of the matching network, given new training samples for the appearance. We also employ a novel training scheme for the meta-learner network to maintain the generalization capability of the feature space by preventing the meta-learner network from generating new parameters that cause overfitting of the matching network. By incorporating our meta-learner network, the target-specific feature space can be constructed instantly with a single forward pass without any iterative computation for optimization and is free from innate overfitting, improving the performance of the tracking algorithm. Fig.3.1 illustrates the motivation of the proposed visual tracking algorithm. We show the effectiveness of our method by showing consistent performance gains in 5 different visual tracking datasets [119, 29, 65, 78, 11] while maintaining a real-time speed of 48 fps.

3.2 Related Work

General visual tracking approaches: Conventional tracking algorithms can be largely grouped into two approaches. One approach builds a generative appearance model of the target based on previously observed examples. This generative model can be used to find the target in the upcoming frames by finding the region that can be best described by the model, where sparse representation and linear subspace representation is often utilized [92, 127, 128, 73]. The other approach aims to build a discriminative classifier to distinguish the target region from the background

region. This discriminative classifier can be used to find the target region in the upcoming frames by solving a binary classification problem [42, 45, 23, 125, 35, 52]. Recently, correlation filters have gained great popularity among the visual tracking methods since the seminal works of [10] and [42] due to their simplicity and computational efficiency in the Fourier frequency domain. Many new approaches have been proposed based on the correlation filter learning framework such as color attribute features [25], using multi-resolution feature maps [88, 24], accurate scale estimation [21], spatial regularization [21], and factorized convolution operators [19].

Visual tracking methods using deep representations: With the growing popularity of the application of deep convolutional networks to a wide range of computer vision tasks, many novel visual tracking algorithms make use of the powerful representation capabilities of the convolutional neural networks (CNN). Starting with [112], where the encoder representation of the denoising autoencoder was used,[82] used feature representations of the VGG-M network [96] and [110] also used VGG feature maps. Many correlation filter-based tracking algorithms also utilize the powerful representation capacity of the CNN by training the correlation filters on the feature maps of the network. Recent approaches include hierarchical correlation filters [69], adaptive hedging of correlation filters [88], continuous convolutional operators [24], sequential training of features [111], and spatial regularization [22]. Other than correlation filter-based algorithms, approaches to design an end-to-end framework for visual tracking have recently emerged. They employ the two-flow Siamese architecture networks commonly used in stereo matching [124] and patch-matching [34] problems. [104] and [40] trained a Siamese network to learn the two-patch similarity function that shares the convolutional representation. [7] proposed a more end-to-end approach to visual tracking where the Siamese network can localize

an exemplar patch inside a search patch. They use a fully convolutional architecture that adopts a cross-correlation layer which significantly lowers the computational complexity. Based on the framework of [7], recent approaches incorporate triplet loss [26], region proposal networks [60], distractor-aware features for suppressing semantic distractors [137] and two-fold Siamese networks for semantic and appearance features [36].

Meta learning methods for few-shot image recognition task and visual tracking: There are recent approaches for learning to classify from a few given examples using meta learning methodologies [108, 32, 89, 81]. In [108], authors proposed a network architecture that employs characteristics of non-parametric nearest-neighbor models to solve N-way, k-shot learning tasks, where a small support set is given. [32] made use of a pre-trained network as a good initialization and then trained the meta-learner to effectively fine-tune the network based on few given examples. In [81], a two-level structure of meta-learner and base-learner both equipped with fast and slow weights was used. The meta-learner acquires the meta information from the base-learner in the form of loss gradients, and then provides the based-learner with fast parameterization while preserving generalization capabilities. Recently, [84] proposed a meta-learner based optimizer analogous to [32] for the visual tracking task, where they chose [97] and [82] as the baseline algorithms to show the effectiveness of their update step, decreasing the number of training iterations thus improving the speed of the baseline methods. Whereas the aim of our method is to update the network using the meta-learner with a single iteration at real-time speeds, providing the network with new adaptive kernels and feature space representation without overfitting, which can be achieved by our regularizing training scheme.

3.3 Tracking with Meta-Learner

In the following subsections, we first provide an overview of our proposed visual tracking framework, where a brief explanation of our framework and the visual tracking procedure are given. Then we describe the implementation and training details for the components of our framework.

3.3.1 Overview of Proposed Method

3.3.1.1 Components

Our framework is largely composed of two components, a matching network and a meta-learner network. The matching network is a fully-convolutional Siamese network that takes two images as inputs where x is denoted as an image patch of the target and z is an image patch of the larger context area that contains the target. The matching network takes these inputs, extracts the feature maps using the N-layer feature extractor CNN network $\phi_{\mathbf{w}}(\cdot)$, and produces the final response map $f_{\mathbf{w}}(x, z)$ by cross-correlation operation between the feature maps. This process can be expressed as follows,

$$f_{\mathbf{w}}(x, z) = \phi_{\mathbf{w}}(x) * \phi_{\mathbf{w}}(z), \tag{3.1}$$

where $*$ represents the cross-correlation operator between two feature maps and $\mathbf{w} = \{w_1, w_2, ..., w_N\}$ represents the set of trained kernel weights for each layer of the feature extractor CNN. To train the feature extractor CNN, we minimize a differentiable loss function given as $\ell(f_{\mathbf{w}}(x, z), y)$, where the loss function measures the inaccuracy in predictions of $f_{\mathbf{w}}$, given y as the ground-truth response map.

The meta-learner network provides the matching network with target-specific

weights given an image patch of the target x with context patches $\mathbf{z}_\delta = \{z_1, ..., z_M\}$ previously obtained and cropped around the target's vicinity. To adapt the weights to the target patch, we use the averaged negative gradient δ of the loss function for the last layer of the matching network taken as,

$$\delta = \sum_{i=1}^{M} -\frac{1}{M} \frac{\partial \ell(f_{\mathbf{w}}(x, z_i), \tilde{y}_i)}{\partial w_N}, \tag{3.2}$$

where $\tilde{y}_i$ is the generated binary response map *assuming* the target is located at the correct position inside the context patch z_i. The meta-learner network is designed based on the fact that the characteristic of δ is empirically different according to a target. Then, given δ as an input, the meta-learner network $g_\theta(\cdot)$ can generate target-specific weights w^{target} corresponding to the input as,

$$w^{target} = g_\theta(\delta), \tag{3.3}$$

where θ is the parameter for the meta-learner network. The new weights are used to update the matching network's original weights as in,

$$f_{\mathbf{w}^{adapt}}(x, z) = \phi_{\mathbf{w}^{adapt}}(x) * \phi_{\mathbf{w}^{adapt}}(z), \tag{3.4}$$

where $\mathbf{w}^{adapt} = \{w_1, w_2, ..., [w_N, w^{target}]\}$, concatenating w^{target} to w_N of the last layer for feature extraction. The meta-learner network also generates channel-wise sigmoid attention weights for each channel of the feature map to further adjust the feature representation space where the weights can be applied by channel-wise multiplication. Fig.3.2 shows an overview of the proposed method.

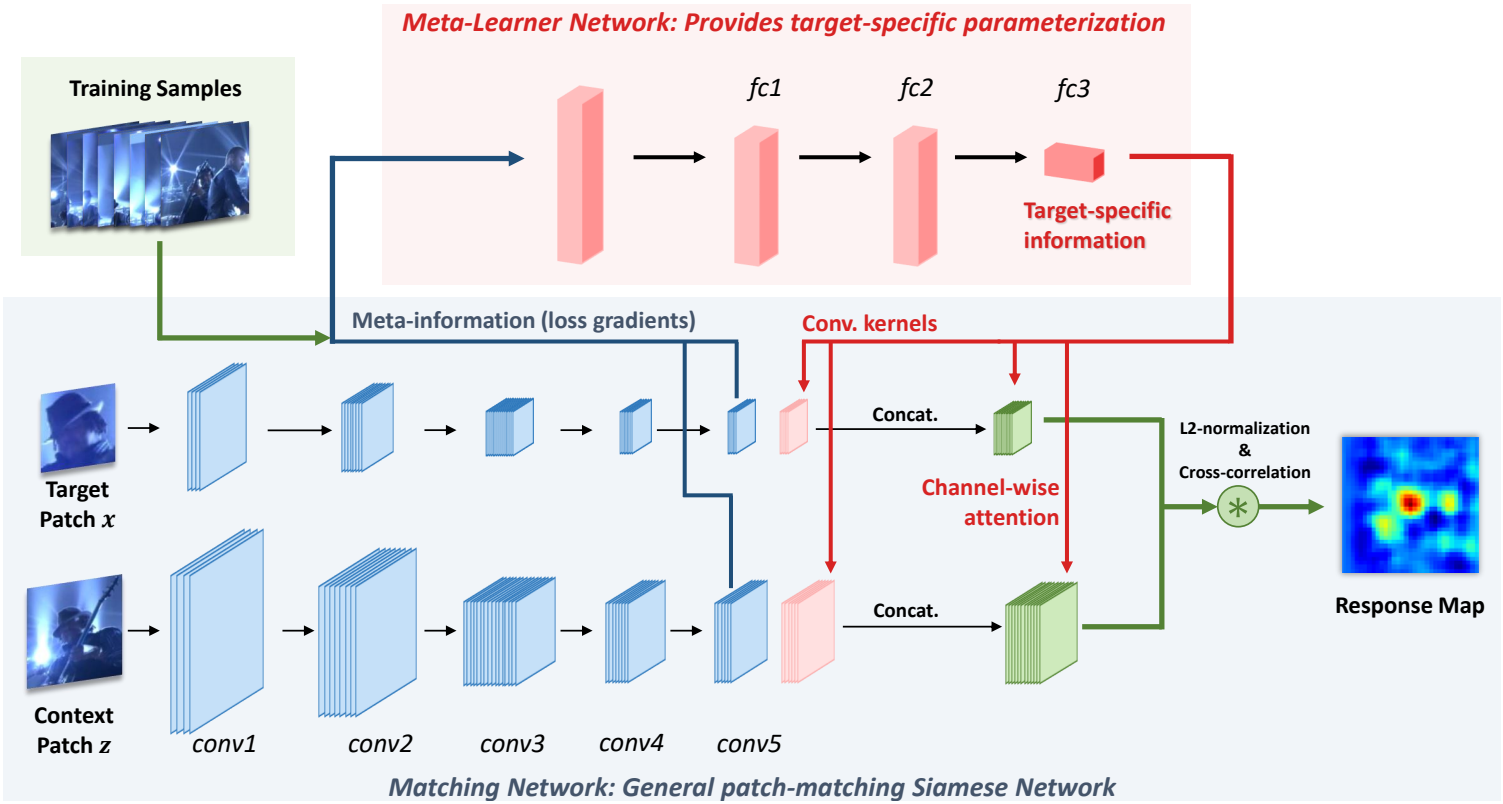


Figure 3.2: **Overview of proposed visual tracking framework.** The matching network provides the meta-learner network with meta-information in the form of loss gradients obtained using the training samples. Then the meta-learner network provides the matching network with target-specific information in the form of convolutional kernels and channel-wise attention.

3.3.1.2 Tracking algorithm

Tracking is performed in a straightforward and simplistic manner to ensure fast performance. Given a target patch x and its previous state, a context image z in a new frame can be cropped based on the previous state. Processing both images through the matching network, the estimated response map $\hat{y} = f_{\mathbf{w}^{adapt}}(x, z)$ is obtained. The new position of the target can be found by finding the maximum position in the response map $\hat{y} \otimes h$, where $\otimes$ is an element-wise multiplication operator and h is a cosine window function for penalizing large displacements. Scale variation of the target can be covered by using multiple size crops of z matched with x. Scale changes are also penalized and damped by a constant to ensure smooth changes of target size over time.

During the course of tracking, we keep a memory of the context images as $\mathbf{z}_{mem} = \{z_1, ..., z_K\}$ along with the corresponding estimated response maps used for tracking $\hat{\mathbf{y}}_{mem} = \{\hat{y}_1, ..., \hat{y}_K\}$. We store a context image z to the memory only if it is considered to be confident, where the maximum response value in the corresponding map $\hat{y}$ is over a certain threshold τ. To update the appearance model of the target, we choose M samples from this memory under the minimum entropy criterion on $\hat{\mathbf{y}}_{mem}$ as in [125] without replacement. This criterion is used to avoid ambiguous response maps where false positive samples may exist in the corresponding context image. Finding the response map with the minimum entropy can be defined as,

$$\underset{\hat{y}_i \in \hat{\mathbf{y}}_{mem}}{\operatorname{argmin}} - \sum_{\mathrm{p} \in \mathcal{P}} \rho(\hat{y}_i[\mathrm{p}]) \log(\rho(\hat{y}_i[\mathrm{p}])), \tag{3.5}$$

where p corresponds to a position in a set of all possible positions $\mathcal{P}$ in the response map and $\rho(\cdot)$ is the normalization function. Using the chosen M appearance samples

Algorithm 3: Visual tracking with meta-learner network

input : Tracking sequence of length L
Initial target state $\mathbf{s}_1$
Corresponding initial target template x
output: Tracked target states $\mathbf{s}_t$

// For every frame in a tracking sequence
for $t = 2$ *to* L **do**
  Obtain a candidate context image z' based on the previous target state $\mathbf{s}_{t-1}$;
  Obtain a response map $\hat{y}$ using the matching network as in eq.(3.1) or eq.(3.4);
  Apply cosine window h to $\hat{y}$, find the position and scale with maximum response, and obtain a new state $\mathbf{s}_t$;

  // Store context image in the memory if confident
  if $\hat{y}[\mathbf{s}_t] > \tau$ **then**
    Obtain new context image z_t based on $\mathbf{s}_t$ and store it in the memory $\mathbf{z}_{mem}$;
  end

  // Update weights every T frames
  if $(t \mod T) == 0$ **then**
    Choose M samples $\mathbf{z}_\delta$ from memory $\mathbf{z}_{mem}$ under minimum entropy metric (3.5);
    Obtain a loss gradient δ as in eq.(3.2);
    Obtain target-specific adaptive weights w^{target} as in eq.(3.3)
    Update $\mathbf{w}^{adapt}$ for the matching network in (3.4)
  end
end

$\mathbf{z}_\delta$, target-adaptive weights w^{target} are obtained using the meta-learner network as in (3.2) and (3.3), and then the matching network is updated as in (3.4), and it is used to track the object in subsequent frames. Since updating the model too frequently is unnecessary and cumbersome for the performance, we only update the model every T frames as in other algorithms [19]. The overall tracking process is described in Algorithm 3.

3.3.2 Network Implementation and Training

3.3.2.1 Matching Network

The matching network consists of a shared feature extractor CNN $\phi(\cdot)$, channel-wise attention step, feature ℓ^2-normalization step, and cross-correlation step. For feature extraction, we use a CNN with 5 convolutional layers and 2 pooling layers of kernel size 3 and stride 2 are applied after the first two convolutional layers. Batch normalization layer is inserted after each convolutional layer. The overall structure of the CNN is analogous to [7], where kernel size and input/output dimensions for each layer are $w_1 : 11 \times 11 \times 3 \times 128$, $w_2 : 5 \times 5 \times 128 \times 256$, $w_3 : 3 \times 3 \times 256 \times 384$, $w_4 : 3 \times 3 \times 384 \times 256$, $w_5 : 1 \times 1 \times 256 \times 192$. For inputs, we use a RGB image of size $127 \times 127 \times 3$ for x and a RGB image of size $255 \times 255 \times 3$ for z, and the matching network produces a response map of size 17×17.

To train the matching network, we used ILSVRC 2015 [93] object detection from video dataset with additional training data from ILSVRC 2017 dataset, which contains objects of 30 classes in 4000 videos in the training set and 1314 videos in the validation set, with a total of 11566 independent object trajectories. Each frame of the video is annotated with bounding box notations of objects appearing in the video. We only used videos in the training set to train the matching network. At training time, pairs of (x, z) are randomly sampled from an object trajectory in a chosen video clip. Then, a ground-truth response map $y \in \{-1, +1\}^{17 \times 17}$ is generated where the value is $+1$ at the position of the target and -1 otherwise. For the loss function $\ell(f_{\mathbf{w}}(x, z), y)$, we use the logistic loss function defined as,

$$\ell(f_{\mathbf{w}}(x, z), y) = \frac{1}{|\mathcal{P}|} \sum_{\mathrm{p} \in \mathcal{P}} \zeta(y[\mathrm{p}]) \cdot \log\left(1 + \exp(-f_{\mathbf{w}}(x, z)[\mathrm{p}] \cdot y[\mathrm{p}])\right), \tag{3.6}$$

where p represents a position in the set of every possible positions $\mathcal{P}$ in the response map and $\zeta(y[\mathrm{p}])$ is a weighting function for alleviating label imbalance. The loss function is optimized with Adam [54] optimizer with the learning rate of 10^{-4} using batch size of 8, and run for 95000 iterations.

3.3.2.2 Meta-Learner Network

We then train the meta-learner network subsequent to pre-training the matching network. The meta-learner network $g_\theta(\cdot)$ consists of 3 fully-connected layers with 2 intermediate layers of 512 units. Each intermediate layer is followed by a dropout layer with the keep probability of 0.7 when training. For input, gradient δ of size $1 \times 1 \times 256 \times 192$ is used and output w^{target} of size $1 \times 1 \times 256 \times 32$ is generated. These new kernels are used to update the weights of the matching network by concatenating w^{target} to the kernels w_5 of the last layer of the Siamese matching network to provide the additional feature space needed for updates, resulting in new kernels $[w_5, w^{target}]$ of size $1 \times 1 \times 256 \times (192 + 32)$.

To train the meta-learner network, we use 1314 videos in the validation set of the ILSVRC video dataset. The training process is described hereafter. First, an anchor target image x is randomly sampled from an object trajectory. Then, M' context patches are randomly sampled from the same object's trajectory as in $\mathbf{z}_{reg} = \{z_1, ..., z_{M'}\}$ where $M' \geq M$. Then M patches are chosen from $\mathbf{z}_{reg}$ to form $\mathbf{z}_\delta$, where we can perform matching these samples with the target image x to obtain gradient δ by (3.2) using generated binary response map $\tilde{y}_i$, assuming the target is located at the center of z_i. Standard data augmentation techniques (*e.g.* horizontal flip, noise, Gaussian blur, translation) are applied when sampling z_i. We can train the meta-learner network $g_\theta(\delta)$ by minimizing the loss function with respect to

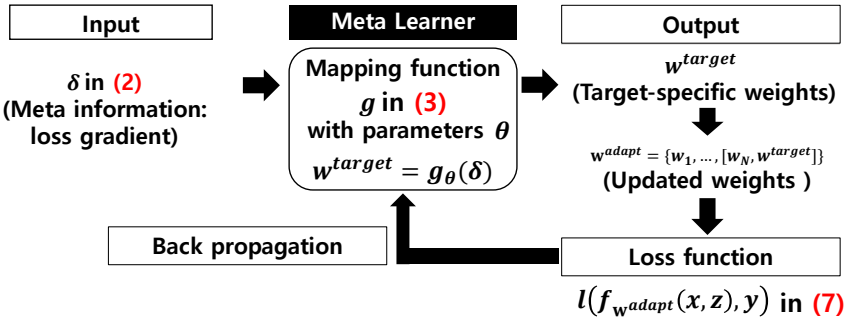


Figure 3.3: **Training scheme of meta-learner network.** The meta-learner network uses loss gradients δ in (3.2) as meta information, derived from the matching network, which explains its own status in the current feature space [81]. Then, the function $g(\cdot)$ in (3.3) learns the mapping from this loss gradient to adaptive weights w^{target}, which describe the target-specific feature space. The meta-learner network can be trained by minimizing the loss function in (3.7), which measures how accurate the adaptive weights w^{target} were at fitting new examples $\{z_1, ..., z_{M'}\}$ correctly.

parameter θ:

$$\underset{\theta}{\operatorname{argmin}} \sum_{z_i \in \mathbf{z}_{reg}} \ell(f_{\mathbf{w}^{adapt}}(x, z_i), y), \text{ where } \mathbf{w}^{adapt} = \{w_1, w_2, ..., [w_N, g_\theta(\delta)]\}. \quad (3.7)$$

Training the meta-learner network to generate new weights $w^{target} = g_\theta(\delta)$ that only fit examples in $\mathbf{z}_\delta$ (*i.e.* $M' = M$) can lead the meta-learner network to generate weights that will make the matching network overfit to samples in $\mathbf{z}_\delta$. To prevent this overfitting problem, a regularization scheme is needed when training. For natural regularization, $M' = 2M$ is chosen so that the weights can fit a larger set of examples $\mathbf{z}_{reg}$ instead of a smaller set $\mathbf{z}_\delta$. This encourages much better generalization properties for the matching network when tracking. For the experiments, $M = 8$ and $M' = 16$ are used and Adam optimizer with the learning rate of 10^{-4} with batches of 8 videos are used. Training is performed for 11000 iterations. Fig.3.3 shows the training scheme of the meta-learner network.

3.4 Experimental Results

3.4.1 Evaluation Environment

Our algorithm was implemented in Python using TensorFlow 1.8.0 [1] library and executed on a system with Intel Core i7-4790K 4GHz CPU with 32GB of RAM with GeForce GTX TITAN X (Maxwell) GPU with 12GB of VRAM. The algorithm ran at an average of 48.1 fps on 100 videos in the OTB-2015 [119] dataset. We considered 3 scale variations of $[1.00, 1/1.035, 1.035]$ to adapt to the scale change of the target, where changes in scale are penalized by a constant of 0.97 and damped by a constant of 0.59. Cosine window h was applied with the penalization factor of 0.25. The meta-learner network updated the weights every $T = 30$ frames, and threshold of $\tau = 0.5$ was used for choosing confident samples. All parameters were fixed during the entire evaluation process for all datasets.

3.4.2 Experiments and Analysis

Object Tracking Benchmark (OTB) [119] is a visual tracking benchmark that is widely used to evaluate the performance of a visual tracking algorithm. The dataset contains a total of 100 sequences and each is annotated frame-by-frame with bounding boxes and 11 challenge attributes. OTB-2013 dataset contains 51 sequences and the OTB-2015 dataset contains all 100 sequences of the OTB dataset. As evaluation metric, we used OPE success rate evaluation metric that compares the predicted bounding boxes with the ground truth bounding boxes to obtain intersection over union (IoU) scores, and measure the proportion of predictions having larger score than a given varying threshold score value. The final score is calculated by measuring the area-under-curve (AUC) for each tracker.

Large-scale Single Object Tracking (LaSOT) [29] dataset is a recently introduced large-scale visual tracking dataset containing 1400 sequences with average length of 2512 frames (83 secs) and minimum of 1000 frames per sequence, with the total of 3.52 million frames where every frame is annotated with a bounding box annotation. It contains 70 object categories with each containing 20 sequences. Compared to OTB, LaSOT contains 14 times more sequences and 59 times the total number of frames, with more various object categories. For evaluation protocols, *Protocol I* employs all 1400 sequences for evaluation and *Protocol II* uses the testing subset of 280 videos, where AUC of success plot is used for the performance metric for both protocols.

TrackingNet [80] is a large-scale tracking benchmark with over $30,000$ videos sequences gathered from YouTube, of which 511 video sequences are designated as the test split. Similar to other tracking benchmarks, AUC of the success plot are shown as the performance metric.

We also perform internal comparisons on TC-128 [65], UAV20L [78] and VOT-2016 [11] datasets to show the effectiveness of our meta-learner update scheme. VOT-2016 is the dataset used for the VOT Challenge [11] and it contains a total of 60 videos with bounding box annotations. Baseline experiment of the VOT dataset performs re-initialization of the tracker when it misses the target, while unsupervised experiment simply lets the tracker run from the first frame to the end. Temple Color-128 (TC-128) dataset [65] contains 128 real-world color videos annotated with bounding boxes, with 11 challenge factors. UAV20L [78] dataset contains 20 video sequences with an average length of 2933.5 frames, where some sequences have targets leaving the video frame (out-of-view). No explicit failure detection or re-detection scheme was used for all experiments.

	MLT	**MLT-*mt***	**MLT-*mt+ft***
OTB-2015	**0.611**	0.564	0.523
OTB-2013	**0.621**	0.571	0.510
LaSOT Protocol I	**0.368**	0.357	0.331
LaSOT Protocol II	**0.345**	0.330	0.305
TC-128	**0.498**	0.477	0.419
UAV20L	**0.435**	0.366	0.342
VOT-2016 Baseline	**0.537**	0.514	0.517
VOT-2016 Unsupervised	**0.421**	0.412	0.411

Table 3.1: **Internal comparison of tracking performance on OTB, LaSOT, TC-128, UAV20L and VOT-2016 datasets.** Proposed **MLT** shows consistent performance gains compared to **MLT-*mt*** and **MLT-*mt+ft*** throughout all datasets. For performance measures, AUC is shown for all experiments, with the exception of baseline experiment of VOT-2016 where A-R overlap score is shown. The best results were written in boldface.

3.4.2.1 Quantitative Analysis

Effect of meta-learner network: We performed an internal comparison between the proposed tracker (**MLT**) and the baseline trackers **MLT-*mt*** and **MLT-*mt+ft***, where **MLT-*mt*** is a variant that has only the matching network with fixed weights without the meta-learner network, and **MLT-*mt+ft*** performs on-line finetuning on `conv5` (kernel w_5) with training examples obtained while tracking. For fair comparison, the baseline trackers are pretrained on the whole ImageNet video detection dataset including the validation set, with the last convolutional layer of kernel size $1 \times 1 \times 256 \times 224$. For the **MLT-*mt+ft*** method, we finetune the matching network every 50 frames for 30 iterations using the Adam optimizer with the learning rate of 10^{-3}. As shown in Table 3.1, the meta-learner network improves the performance of the baseline matching network and produces better tracking results. **MLT** is consistently superior to **MLT-*mt*** and **MLT-*mt+ft*** in OTB, LaSOT, TC-128, UAV20L and VOT2016 datasets. The results demonstrate that the adaptive weights generated by meta-learner network are effective for inducing the customized feature space

Kernel Weights	Channel Attention	ℓ^2-Normalization	AUC
✓	✓	✓	0.611
	✓	✓	0.571
✓		✓	0.601
✓	✓		0.580
			0.564

Table 3.2: **Ablation study of individual components.** Experiments are performed on OTB-2015 dataset. Performance is denoted in AUC of OPE success plot.

for each target and for resulting in accurate visual tracking, showing performance gains in all 5 datasets. Also, results of **MLT-*mt+ft*** show that online finetuning without handpicked hyper-parameters and regularization scheme easily results in overfitting to handful of training samples, resulting in lower performance.

Ablation analysis of individual components: We show the results of the ablation experiments to analyze the impact of each component, which are kernel weights, channel attention, and ℓ^2-normalization. The result in Table 3.2 shows that while all components contribute to performance improvements, kernel weights generated by the meta-learner have the biggest impact on performance.

Comparison with other trackers: We compare our tracking algorithm MLT with 12 tracking algorithms on the OTB and LaSOT datasets, namely, SiamFC [7], StructSiam [131], DSiam [33], CFNet [106], SINT [104], SRDCF [23], PTAV [30], ECO-HC [19], $\text{STAPLE}_{\text{CA}}$ [79], BACF [53], DSST [21] and HDT [88], where most are real-time algorithms. As shown in Table 3.4, MLT achieves a competitive accuracy on OTB datasets compared to other tracking algorithms based on deep representation, and outperforms other algorithms on large-scale LaSOT experiments due to its robust meta-learner update. Especially, we were able to obtain a noticeable performance gain compared to SiamFC and its variants (StructSiam, DSiam and CFNet) on LaSOT where no variant was able to outperform the original SiamFC. Results on

	MLT	SiamFC	CFNet	ECO-HC	$\text{STAPLE}_{\text{CA}}$	BACF	SRDCF	SAMF	ASLA	DSST
TrackingNetTest	**0.581**	0.571	0.578	0.541	0.529	0.523	0.521	0.504	0.478	0.464

Table 3.3: **Quantitative results on TrackingNetTest dataset.** MLT denotes the proposed algorithm. The proposed algorithm shows similar performance rank as in LaSOT dataset experiments.

the test set of TrackingNet dataset are shown in Table 3.3 where they show similar tendencies as in LaSOT.

We also separately analyze the performance of MLT with respect to 8 different attributes of OTB videos. Each video has a different attribute such as in-plane rotation, out-of-plane rotation, motion blur, low resolution, scale variation, illumination variation, background clutter and occlusion. Fig. 3.4 shows that MLT is robust to low resolution, occlusion and scale variation and is competitive to the other trackers for most of the attributes. In blurred low resolution images, the appearance of a target is frequently indistinguishable from that of other objects in the background. MLT can distinguish between these appearances by customizing features spaces for each target with the meta-learner network. Also, MLT can learn from negative examples from the background and handle occlusions better than other trackers.

	MLT	SiamFC	StructSiam	DSiam	CFNet	SINT	SRDCF	PTAV	ECO-HC	STAPLE$_{CA}$	BACF	DSST	HDT
OTB-2015	0.611	0.582	0.621	-	0.586	0.580	0.598	0.635	**0.643**	0.598	0.630	0.520	0.564
OTB-2013	0.621	0.607	0.638	0.642	0.611	0.635	0.626	0.663	0.652	0.621	**0.678**	0.554	0.603
LaSOT Protocol I	**0.368**	0.358	0.356	0.353	0.296	0.339	0.339	0.269	0.311	0.262	0.277	0.233	-
LaSOT Protocol II	**0.345**	0.336	0.335	0.333	0.275	0.314	0.314	0.250	0.304	0.238	0.259	0.207	-
FPS	48	58	45	45	43	4	5	25	60	35	35	24	10

Table 3.4: **Quantitative results on OTB [119] and LaSOT [29] datasets.** MLT denotes the proposed algorithm. The proposed algorithm shows competitive performance on OTB datasets and outperforms other algorithms on large-scale LaSOT datasets, obtaining performance gains with the benefit of additional feature space provided by the meta-learner. AUC for OPE is used for the performance measures.

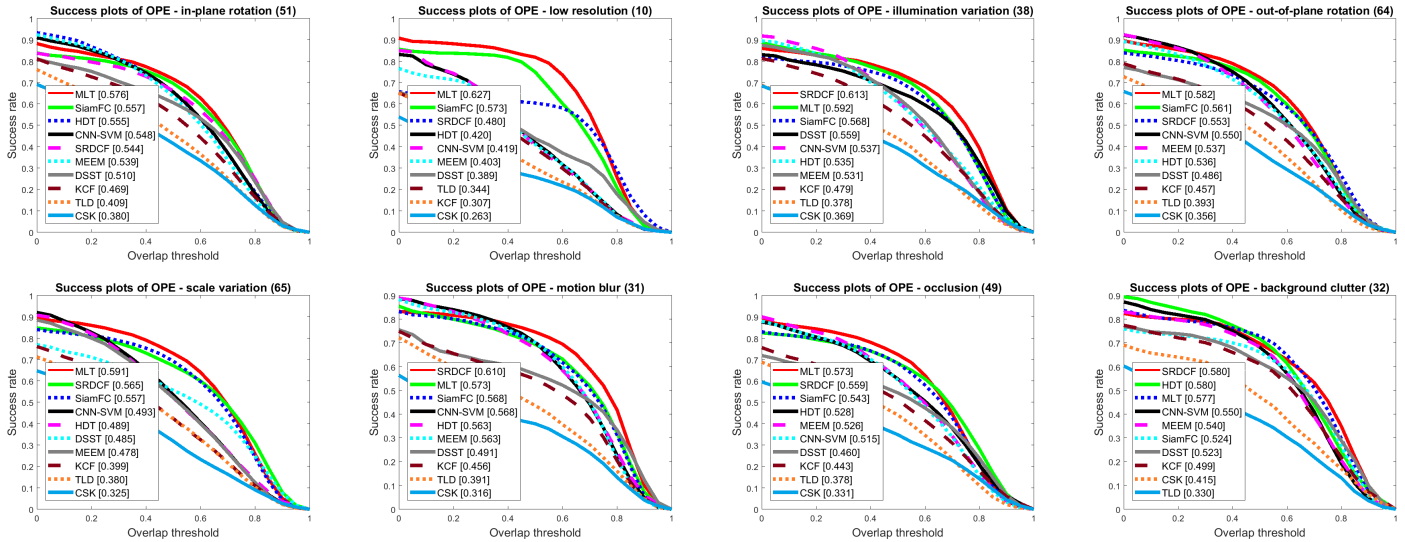


Figure 3.4: **Success plots for** 8 **challenge attributes** of the OTB-2015 dataset

3.4.2.2 Qualitative Analysis

Fig.3.5(a) shows qualitative tracking results produced by SiamFC, SRDCF, HDT, CNN-SVM, DSST, and the proposed algorithm MLT. All trackers were tested on all videos in the OTB-2015 dataset where tracking results for selected videos are shown in Fig.3.5(a) due to the limit in the length of the paper. MLT robustly and accurately tracks the target in spite of several challenging conditions such as occlusion in *Box* seq., pose variation in *Rubik* seq., background clutter in *Human3* seq., fast motion in *Girl2* seq., and scale variation in *Car24* seq.. These qualitative tracking results demonstrate that the proposed MLT successfully exploited the power of the meta-learner network and utilized the adaptive weights customized for each target to improve tracking accuracy, without losing the generalization capabilities. For additional reference, qualitative results on the test split of LaSOT dataset are also shown in Fig.3.5(b) where our proposed MLT successfully tracks the object under challenging scenarios.

In addition, Fig.3.7 shows some examples of how the target-specific feature space modifies the response maps, thus demonstrating how the meta-learner can be beneficial in the task of visual tracking. We show example image patches z where the target object fixed at the center of the image, with response maps before and after the target-specific feature space modification. The response maps show that the target-specific weights help the tracker to adapt to various target appearance changes and to locate the target, and are also effective in avoiding false positives by suppressing incorrect responses from distractors in the background.

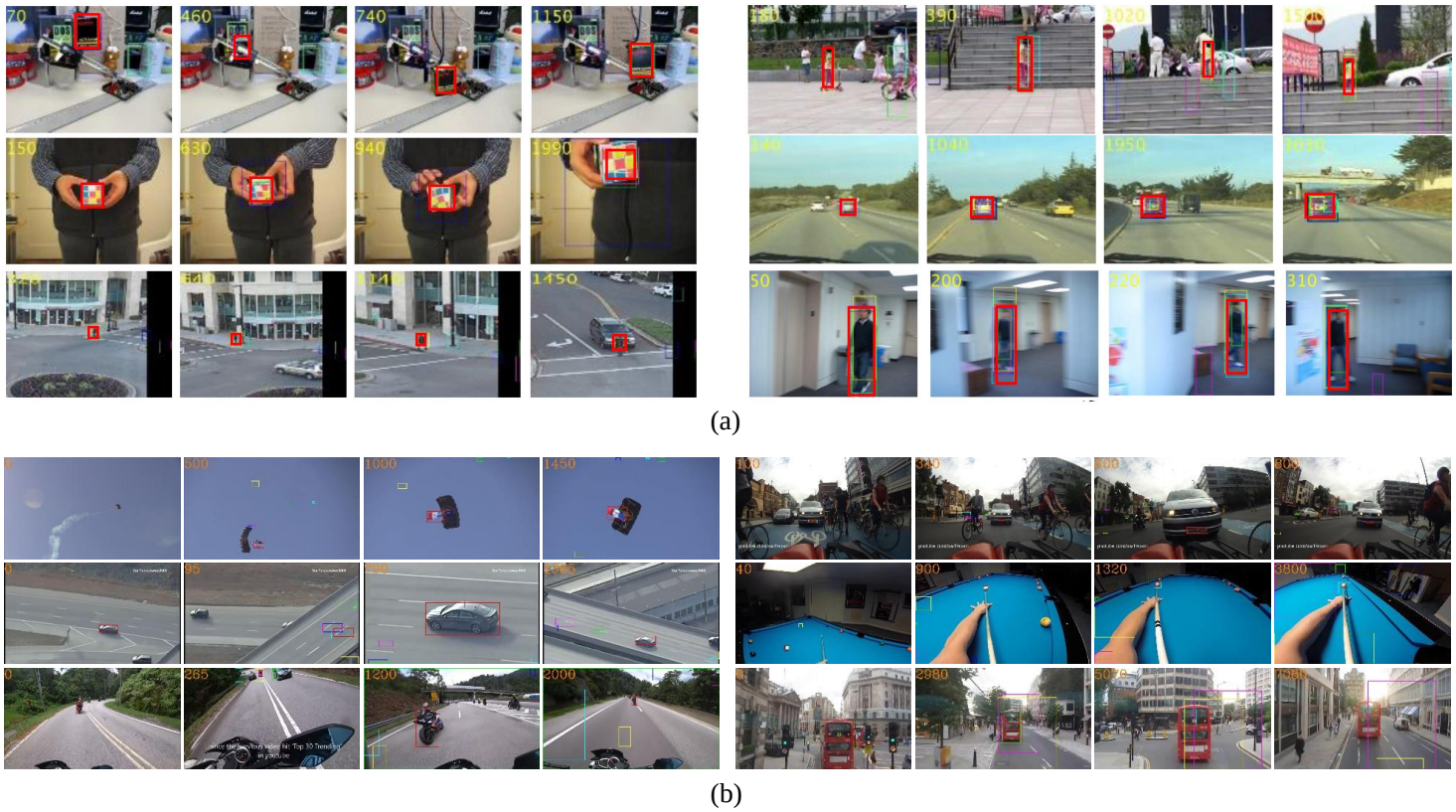


Figure 3.5: **Qualitative results.** Tracking results for (a) *box*, *girl2*, *rubik*, *car24*, *human3*, *blurBody* sequences from OTB-2015 dataset. Green, Blue, Cyan, Yellow, Violet, and Red bounding boxes denote tracking results of SiamFC, SRDCF, HDT, CNN-SVM, DSST, and MLT, respectively. (b) *flag-1*, *licenseplate-4*, *car-20*, *pool-19*, *motorcycle-4*, *bus-2* sequences from LaSOT dataset. The results of MLT, SiamFC, StructSiam, ECO-HC, Staple$_{\text{CA}}$, SRDCF are shown in the bounding boxes with colors red, green, blue, cyan yellow and magenta, respectively. Yellow numbers on the top-left corners indicate frame numbers.

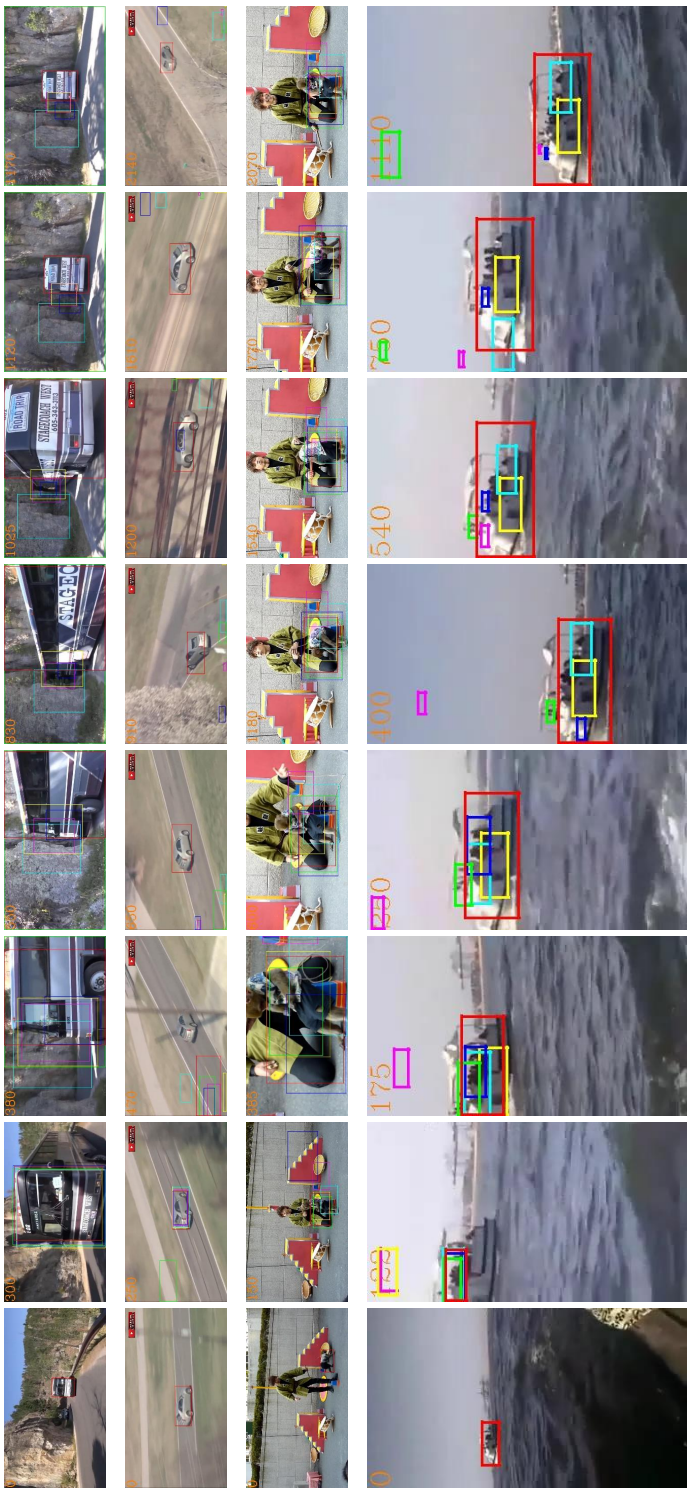


Figure 3.6: **Qualitative results.** Tracking results for *bus-1*, *car-18*, *monkey-17*, *boat-19* sequences from LaSOT dataset. The results of MLT, SiamFC, StructSiam, ECO-HC, Staple$_{CA}$, SRDCF are shown in the bounding boxes with colors red, green, blue, cyan yellow and magenta, respectively. Yellow numbers on the top-left corners indicate frame numbers.

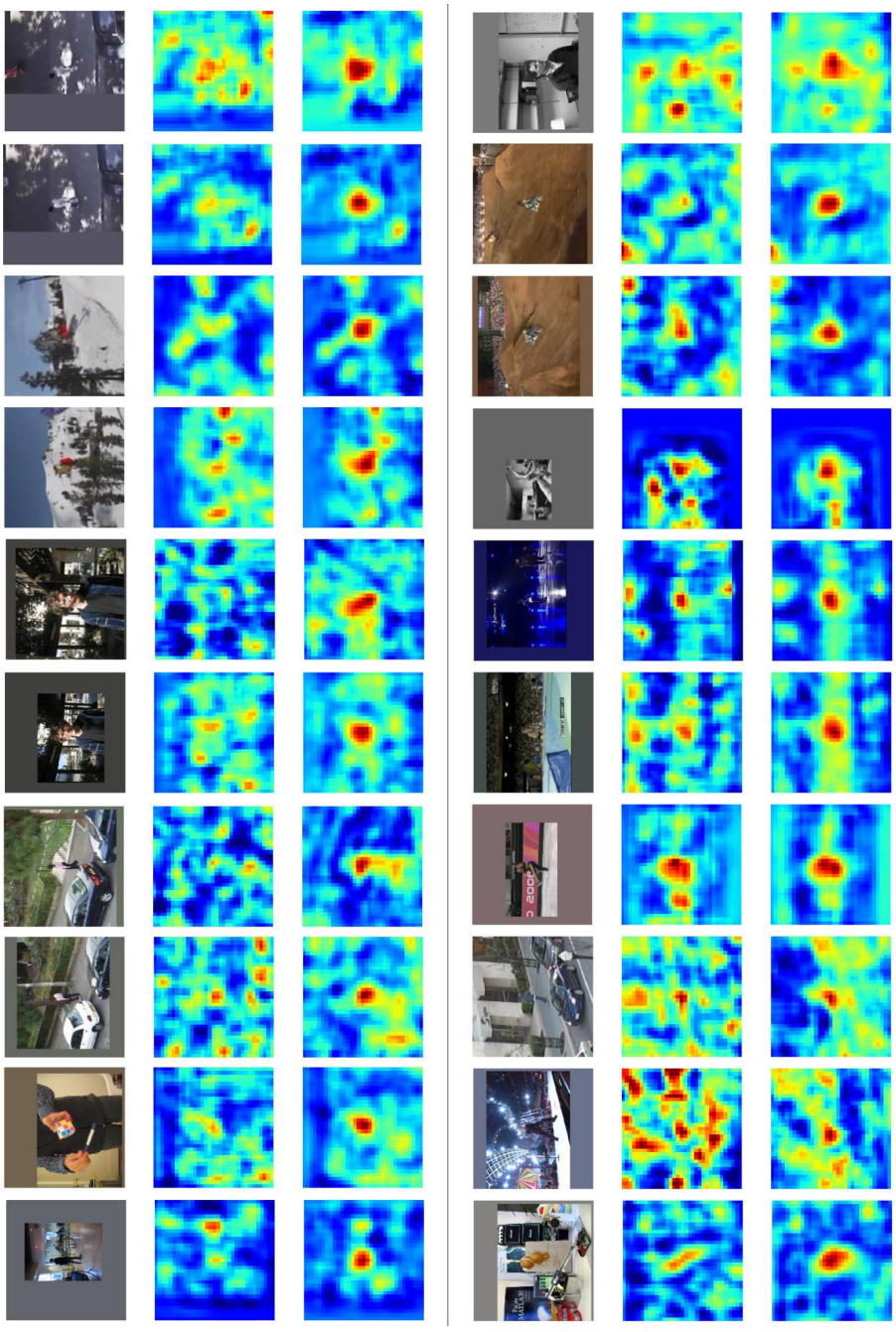


Figure 3.7: **Visualization for the effect of the target-specific feature space.** This shows some example image patches z (1^{st} and 4^{th} row) with the changes in response maps $\hat{y}$ ***before*** (2^{nd} and 5^{th} row) and ***after*** (3^{rd} and 6^{th} row) applying our adaptive weights w^{target} generated by our meta-learner.

3.5 Summary

In this chapter, we proposed a novel visual tracking algorithm based on the target-specific feature space constructed by the deep meta-learner network. The proposed tracking algorithm adapts to the target appearance by generating the target-specific adaptive weights with the meta-learner network, where the matching network provides the meta-information gradients as a learning signal. Our algorithm aims to customize the feature space to discriminate a specific target appearance from the background in order to accurately track the target without overfitting. Experimental results demonstrate that our algorithm achieves a noteworthy performance gain in visual tracking by using the proposed meta-learner network, achieving consistent performance gains on 5 tracking datasets including the large-scale tracking dataset LaSOT. Quantitatively and qualitatively the algorithm shows a competitive tracking performance on multiple visual tracking datasets with several challenging tracking conditions, compared to other visual tracking algorithms, while running at the real-time speed of 48 fps.

Chapter 4

Model Update without Forgetting by Continual Meta-Learning

In the previous chapter, we proposed a meta-learning based model update approach for visual tracking where the meta-learner provides the baseline tracker with updated feature space, making use of additional kernel and channel attention weights. However, since the preceding method relies on a limited set of external template pool when performing an online update, the tracker easily forgets previous examples if templates are discarded from the pool, resulting in a lack of long-term memory. Additionally, importance of initial model adaptation is overlooked, only focusing on modeling the online update.

In this chapter, we aim to solve the aforementioned issues by introducing an adaptive continual meta-learner that simultaneously models initial and online adap-

tations. Proposed meta-learning framework is based on the largely popular model-agnostic meta-learning (MAML) [32] where its aim is to find the default weights for fast adaptation. On top of MAML, in order to deal with label noise due to self-supervision, we employ an adaptive learning strategy where hyperparameters can dynamically change to regulate the learning process. In addition, long-term memory is embedded into the weights by employing the knowledge distillation scheme, where the adaptive learner also controls whether to forget or retain previous knowledge.

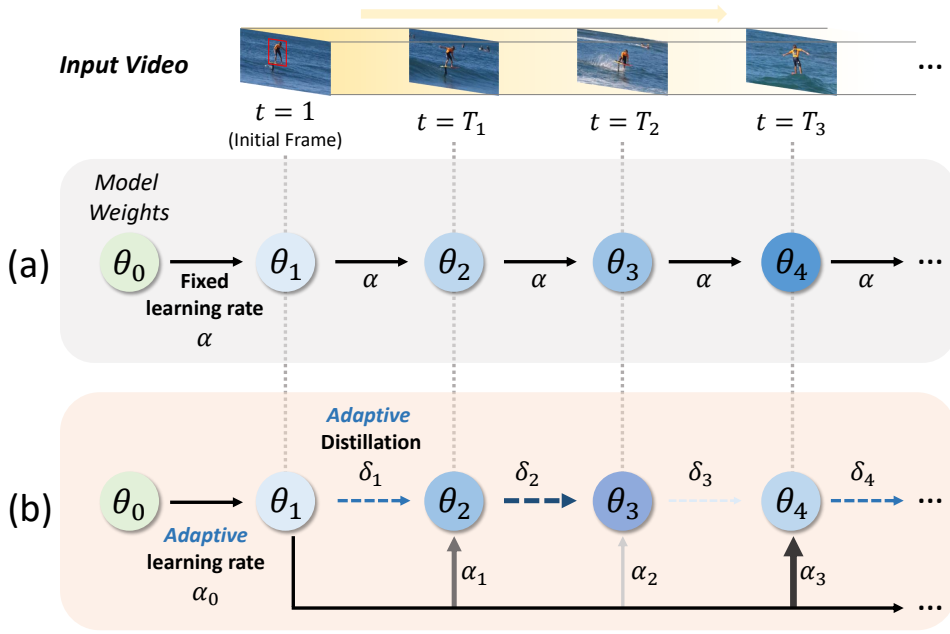


Figure 4.1: **Motivation for the proposed visual tracking framework.** Given an input video, (a) conventional tracking algorithm initializes and updates its model weights using fixed and predefined learning rates. (b) Our proposed tracking framework incorporates an adaptive learning scheme for both model initialization and online update using adaptive learning rate and adaptive knowledge distillation, which increases the flexibility of the tracker to learn new examples, while aiming to achieve robustness through retaining the memory on previously seen examples.

4.1 Introduction

Recently, with the advances in the application of deep convolutional neural networks (CNN) to image classification and object detection tasks [39, 56, 96, 91], visual tracking algorithms have also achieved large improvements in performance, owing to the representation power of their deep backbone networks [82, 7] and the object detection framework [60, 133]. However, there is a misalignment between goals of object detection and visual tracking problem, where object detection aims to locate all objects of same semantic class whereas visual tracking aims to locate a specific object instance. To overcome this gap, visual tracking algorithms employ some form

of domain adaptation process to the object detection framework, such as online network finetuning using stochastic gradient descent (SGD)-based methods [82, 50, 20, 8] or Siamese network structure [7, 60, 48] which generates a target-specific convolutional kernel from the initial frame.

While recent tracking algorithms were successful in achieving high performance metrics on several visual tracking benchmarks [119, 80, 49], the importance of the online adaptation process was often overlooked despite their crucial role in visual tracking. In particular, the tracker may need to update its model since the appearance of the target object constantly changes and similar distractor objects can appear in a given scene. Moreover, these aspects are further emphasized in long-term tracking scenarios [29, 107, 77] where the target can be absent for a prolonged time interval. Online update was often achieved by incorporating hand-crafted regularization schemes and meticulous hyperparameter selection, due to the lack of training samples and label uncertainty. In cases of most Siamese network-based trackers, online adaptation was completely ignored to achieve faster and real-time speeds. To address these aforementioned issues, several recent trackers employed meta-learning-based adaptation schemes [85, 16, 109, 51, 126, 18, 61, 47] in order to learn the adaptation process itself. However, a majority of them either focused only on finding a good initialization for the tracker [85, 109] or only regulating the online update process [16, 126, 18, 61].

In this chapter, we introduce a more generalized visual tracking framework, in which we model both adaptation and continual processes under our adaptive continual meta-learning framework. We adopt an adaptive scheme where hyperparameters can dynamically change to deal with various tracking scenarios. In addition, our continual meta-learning scheme employs adaptive knowledge distillation-based up-

date strategy to help the tracker adapt to newly obtained examples while retaining the necessary knowledge on previously seen examples. During offline training, our integrated framework trains (1) network weights that are good for both initialization and future online updates; (2) hyperparameter generator network which adaptively generates the learning rates, instance-wise weights, and focal loss hyperparmeter for controlling the adaptation process; and (3) knowledge distiller network for regulating the balance between learning new examples and retaining the knowledge from the previous step.

To validate the effectiveness of our proposed framework, we apply our method to two Siamese network-based tracking algorithms based on TACT [17], which is a two-stage detector-based tracker. We compare our method to other state-of-the-art trackers on test splits of large-scale visual tracking datasets, including LaSOT [29], OxUvA [107], TLP [77], TrackingNet [80], and GOT-10k [49]. We further demonstrate the effectiveness of our framework by component-wise ablation analyses on the LaSOT dataset. Our framework requires minimal computational overhead, achieving real-time speeds for both of trackers. Motivation for our tracking framework is shown in Figure 4.1.

4.2 Related Work

CNN-based tracking algorithms : Contemporary visual tracking algorithms solve visual tracking via tracking-by-detection, where they attempt to locate the target by finding the position where the classifier produces the highest classification score for the target class. With the powerful representation capacity of CNNs, recent trackers employ CNNs for feature extraction and classification. While features from

denoising autoencoder are used in [112], MDNet [82] used VGG [96] features with multi-task learning, which RT-MDNet has accelerated to real-time speed by using ROIAlign [37]. Correlation filter-based trackers [42, 10] have also been widely used on top of pretrained deep features, such as C-COT [24] and ECO [19], in which they use continuous convolutional operator for the fusion of multi-resolution CNN feature maps. Other approaches include spatially regularized filters [22], the fusion of multilevel features [9], and group feature selection [120].

Recently, Siamese network-based trackers have gained traction due to their simplicity and high performance [7, 60, 137, 59, 132, 33, 131, 31, 122, 14, 133]. SiamFC [7] introduced a fully convolutional end-to-end approach with increased speed and accuracy. SiamRPN [60] added a region proposal network for more accurate localization and size estimation while DaSiamRPN [137] enhanced its discriminability by introducing negative pairs during training to suppress distractors. Both [59] and [132] utilized deeper and wider feature extractors based on [39] and [102] for further performance gains. Other works include general transformation learning model [33], local pattern detection for structure-based prediction [131], cascaded region proposal for sequential refinement [31], and recurrent optimization based model [122].

Meta-learning for visual tracking : To overcome the issues of conventional model adaptation in visual tracking, noteworthy methods have been proposed to improve the tracking performance by employing meta-learning based adaptation at test-time. Among meta-learning algorithms, model-agnostic meta-learning (MAML) [32] based approaches [64, 2, 6, 83, 5] recently gained attention owing to its simplicity and versatility. MAML aims to find good model weights that can be trained to generalize well with a small number of SGD steps and a small amount of training data. Meta-Tracker [85] was one of the first to apply MAML on MDNet [82] for fast adap-

tation, reducing the number of SGD iterations. MetaRTT [51] extended the idea by simultaneously finding the learning rates for initialization and online update. In addition, [109] used MAML to convert a modern detection network into a tracker. However, all above methods used fixed learning rates for all tracking scenarios and thus lack the adaptiveness to deal with diverse individual scenarios, which are accompanied by various training examples with varying degree of label uncertainty. Additionally, they do not address the erroneous updates performed with uncertain and mislabeled examples. On the other hand, our proposed method is able to adaptively change the hyperparameters to deal with these scenarios.

Other meta-learning-based tracking algorithms introduce a separate meta-learner network to regulate the adaptation process. [61] and [16] used loss gradient information obtained during tracking to update the target feature representation. Moreover, [126] used a separate update module to acquire the updated accumulated template. An optimization-based architecture with a model predictor was used in [8] to predict the filter weights while a similar approach using recurrent neural optimizer is proposed in [122]. However, a majority of the aforementioned methods are mainly focused on short-term tracking scenarios, and are not designed for long-term tracking scenarios. With the exception of [18], where they used a meta-updater network that takes multiple cues as input to make the binary decision whether to update or not to update the baseline tracker.

Incremental object classification and detection : Conventional training setting for the classification problem assumes that abundant labeled training examples are always available for all classes at any point in training time. By contrast, incremental/continual setting assumes new examples and new classes are given in a sequential manner, thus the model has to be trained incrementally to prevent the

model from catastrophic forgetting, which is a phenomenon where the performance of the model on previously seen examples significantly degrades over time. Recent approaches for deep neural networks include iCaRL [90] which learns the classifier and representation simultaneously based on replay memory; EWC [55] which selectively slows down learning of weights based on their importance; and LwF [63] where task-specific parameters from previous tasks are utilized with knowledge distillation loss to prevent the network from forgetting, while improving the performance on a new task. Related to LwF, incremental learning of object classification and detection models based on the knowledge distillation [43] scheme to prevent catastrophic forgetting have recently emerged [94, 67, 46, 135].

Inspired by aforementioned approaches, we employ a knowledge distillation-based continual meta-learning scheme for our visual tracking framework. Different from conventional continual learning settings where new examples are given in a sequential manner along with their corresponding ground truth labels, these labels are not available under standard visual tracking setting. Since labels for new examples have to be obtained in a self-supervised manner, chance of adapting the model based on mislabeled examples persists. To alleviate this issue, we introduce two solutions. (1) When performing an online update at a certain time step, we always start from initially adapted weights where previous weights are used for knowledge distillation. This reduces error accumulation and overfitting to small number of training examples, while increasing the flexibility of the tracker. (2) We introduce an adaptive knowledge distiller network that predicts the importance weights for each previous frame where the magnitude of weights determine the degree of knowledge distilled from a certain frame. By controlling these weights, the tracker can choose between learning new examples and retaining the previous knowledge.

4.3 Tracking with Adaptive Continual Meta-Learner

Our proposed framework consists of two large components, which are the baseline tracking algorithm and the adaptive continual meta-learner module. The baseline tracking algorithm can be any deep neural network-based tracker or detector, where we choose trackers from Siamese network-based TACT [17] as baselines. In the following subsections, training procedure for our proposed adaptive continual meta-learner module is delineated.

Assuming a baseline tracker f_{θ_0} with its default weights θ_0, the meta-learner network g controls the learning process through adaptively generating the hyperparameters to modify the direction and magnitude of the loss gradients. The meta-learner network g contains four sub-networks, g_α, g_β, g_γ, and g_δ, where each sub-network generates the learning rate α, instance weight β, focal loss hyperparameter γ, and knowledge distillation hyperparameter δ, respectively. Our objective is to train the default weights θ_0 and network weights for g. To train both parameters, we construct a simulated tracking episode to perform the initial and online adaptation processes and assess how well these adaptations are conducted by evaluating the loss on the future frames. Our training scheme extends on the basic meta-learning formulation of dividing the training set into support and query sets, then performing inner-loop and outer-loop optimizations for meta-training analogous to such as in [32].

4.3.1 Meta-Training with Simulated Episodes

Tracker setting : The baseline tracking algorithm f_θ with weights θ takes a video frame image I^t as an input and outputs K candidate bounding boxes $b_\theta^t \in \mathbb{R}^{K\times 4}$

with corresponding confidence values $c_\theta^t \in \mathbb{R}^K$, which is denoted as,

$$b_\theta^t, c_\theta^t = f_\theta\left(I^t\right). \tag{4.1}$$

Tracking is performed by choosing the bounding box with the highest confidence value as an output. Online updates are conducted by training the tracker using this chosen output box, in which other boxes are labeled as positive if they have high overlap with the output box (IoU$> \tau_p$) and negative otherwise (IoU$< \tau_n$). We chose overlap threshold values $\tau_p = 0.5$ and $\tau_n = 0.3$ for training and testing.

Episode setting : Given a training video sequence $\mathcal{V}$ of length T with its frame images $\{I^1, I^2, ..., I^T\}$ and ground truth target bounding box annotations $\{b^1, b^2, ..., b^T\}$, the video sequence is divided into four time-ordered video segments $\mathcal{I}^1$, $\mathcal{I}^2$, $\mathcal{I}^3$, and $\mathcal{I}^4$ with corresponding bounding box label sets $\mathcal{B}^1$, $\mathcal{B}^2$, $\mathcal{B}^3$, and $\mathcal{B}^4$. Then, each video segment and label set are paired to form four datasets $\mathcal{D}^1 = (\mathcal{I}^1, \mathcal{B}^1)$, $\mathcal{D}^2 = (\mathcal{I}^2, \mathcal{B}^2)$, $\mathcal{D}^3 = (\mathcal{I}^3, \mathcal{B}^3)$, and $\mathcal{D}^4 = (\mathcal{I}^4, \mathcal{B}^4)$, respectively. Each video segment contains frame images with sizes of $|\mathcal{I}^1| = 1$, $|\mathcal{I}^2| = |\mathcal{I}^3| = N$, and $|\mathcal{I}^4| = T - 2N - 1$, where N is the number of frames used for online adaptation.

Given a baseline tracker f_{θ_0} with the default weights θ_0, the initial adaptation is first performed using the dataset $\mathcal{D}^1 = (\mathcal{I}^1, \mathcal{B}^1)$ to obtain adapted weights θ_1. Then, using the initialized tracker f_{θ_1} on images in $\mathcal{I}^2$, we can obtain estimated labels $\hat{\mathcal{B}}^2$ to form the dataset for self-supervised online update $\hat{\mathcal{D}}^2 = (\mathcal{I}^2, \hat{\mathcal{B}}^2)$ where the tracker is updated from θ_1 to θ_2. Lastly, using the adapted tracker f_{θ_2}, online update is performed again with dataset $\hat{\mathcal{D}}^3 = (\mathcal{I}^3, \hat{\mathcal{B}}^3)$ to obtain θ_3. After simulating tracking episodes, we obtain intermediate weights θ_0, θ_1, θ_2, and θ_3 for the tracker. To train our overall framework, we evaluate the tracker on different combination of

datasets based on each intermediate weight, then perform outer-loop optimization on the loss to train the default weights θ_0 and network weights for meta-learner g. The overview for the training process of our proposed framework is depicted in Figure 4.2.

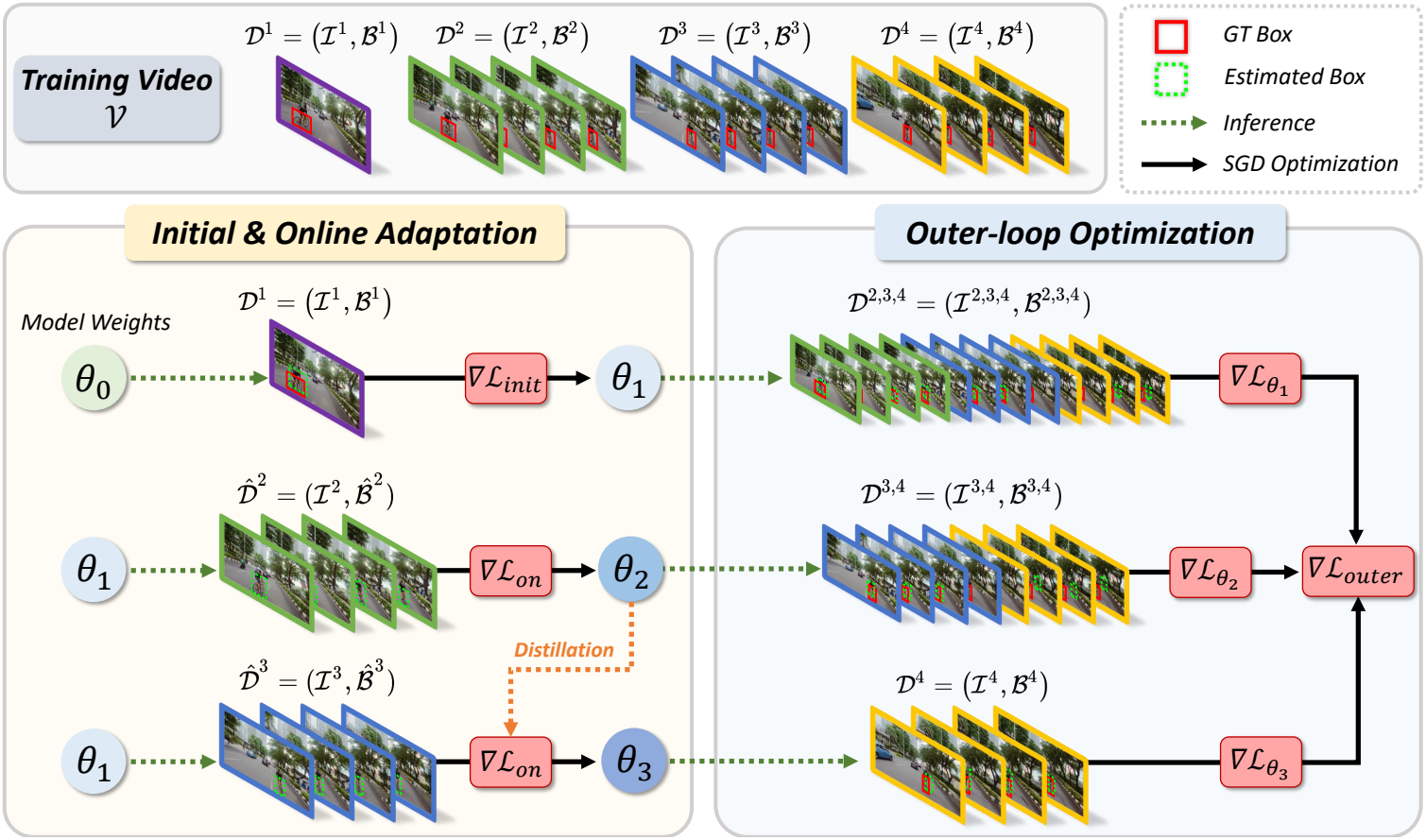


Figure 4.2: **Overview for the training process of proposed framework.** The training video $\mathcal{V}$ is divided into four datasets, $\{\mathcal{D}^1, \mathcal{D}^2, \mathcal{D}^3, \mathcal{D}^4\}$, where each dataset $\mathcal{D}^i$ contains frame images $\mathcal{I}^i$ and GT box labels $\mathcal{B}^i$. Initial adaptation is performed using the initial frame and label in $\mathcal{D}^1$, and online adaptations are conducted using self-supervised labels $\hat{\mathcal{B}}^2$, $\hat{\mathcal{B}}^3$ in $\hat{\mathcal{D}}^2$, $\hat{\mathcal{D}}^3$. Afterwards, outer loop optimization is performed to evaluate all adapted weights θ_i on $\mathcal{D}^{i+1,\cdots}$ for meta-training.

Initial adaptation : Our tracker first performs the initial adaptation process using the initial frame and label, $\mathcal{D}^1$. Starting from θ_0, the adapted weights θ_1 are obtained by,

$$\theta_1 = \theta_0 - \alpha_0 \odot \nabla_{\theta_0} \mathcal{L}_{init}\left(\mathcal{D}^1\right), \tag{4.2}$$

where $\alpha_0 = g_\alpha(\tau_0)$ is the predicted per-parameter learning rate and the input τ_0 is the learning state based on the layer-wise mean of gradients and kernels $[\bar{\nabla}_{\theta_0}\mathcal{L}, \bar{\theta_0}]$, as defined in [5]. The loss function for the initial adaptation $\mathcal{L}_{init}$ is defined as,

$$\mathcal{L}_{init}\left(\mathcal{D}^1\right) = \beta_0 \mathrm{FL}\left(c^1_{\theta_0}; \gamma_0\right), \tag{4.3}$$

where FL denotes the focal loss [66] evaluated using the initially given bounding box label $\mathcal{B}^1$, $\beta_0 = g_\beta\left(c^1_{\theta_0}\right)$ is the instance weight for the initial frame, and $\gamma_0 = g_\gamma\left(c^1_{\theta_0}\right)$ is the focusing hyperparameter used in focal loss to control the balance between losses of easy and hard samples. β_0 and γ_0 are both scalar values. In practice, we expand the initial dataset $\mathcal{D}^1$ by augmenting the scale of I^1 by constant factors of 1.04 and 1.08, resulting in five images and labels for $\mathcal{D}^1$.

Online adaptations : Using the initially adapted tracker f_{θ_1} and frames in $\mathcal{I}^2$, online adaptation is performed using $\hat{\mathcal{D}}^2$ to obtain updated parameters θ_2 as in,

$$\theta_2 = \theta_1 - \alpha_1 \odot \nabla_{\theta_1} \mathcal{L}_{on}\left(\hat{\mathcal{D}}^2\right), \tag{4.4}$$

where $\alpha_1 = g_\alpha(\tau_1)$. Loss function for the online update is defined as,

$$\mathcal{L}_{on}\left(\hat{\mathcal{D}}^2\right) = \frac{1}{|\mathcal{I}^2|} \sum_{I^i \in \mathcal{I}^2} \left\{\beta^i_1 \mathrm{FL}\left(c^i_{\theta_1}; \gamma^i_1\right) + \delta^i_1 \mathrm{KD}\left(c^i_{\theta_1}, c^i_{\theta_0}\right)\right\}, \tag{4.5}$$

where FL is evaluated using self-labeled bounding boxes in $\hat{\mathcal{B}}^2$ as labels, $\beta_1^i = g_\beta\left(c_{\theta_1}^i\right)$, and $\gamma_1^i = g_\gamma\left(c_{\theta_1}^i\right)$. KD is the knowledge distillation loss equivalent to the standard binary cross entropy loss and is used to measure discrepancy between predictions made by the model with parameters θ_0 and θ_1. To control the degree of knowledge distilled from a certain example, the knowledge distillation hyperparameter $\delta_1^i = g_\delta\left(c_{\theta_1}^i, c_{\theta_0}^i\right)$ is predicted, where δ_1^i is a scalar value.

Afterwards, further online adaptation is performed using $\hat{\mathcal{D}}^3$, where $\hat{\mathcal{B}}^3$ is obtained by evaluating the tracker f_{θ_2} on frames in $\mathcal{I}^3$. Updated parameters θ_3 can be acquired by evaluating the equations analogous to the previous step as in Eq. (4.4) and Eq. (4.5) where,

$$\theta_3 = \theta_1 - \alpha_2 \odot \nabla_{\theta_1} \mathcal{L}_{on}\left(\hat{\mathcal{D}}^3\right), \tag{4.6}$$

$$\mathcal{L}_{on}\left(\hat{\mathcal{D}}^3\right) = \frac{1}{|\mathcal{I}^3|} \sum_{I^i \in \mathcal{I}^3} \left\{\beta_2^i \mathrm{FL}\left(c_{\theta_1}^i; \gamma_2^i\right) + \delta_2^i \mathrm{KD}\left(c_{\theta_1}^i, c_{\theta_2}^i\right)\right\}, \tag{4.7}$$

where online adaptation is performed from the initially adapted parameters θ_1 rather than previous-step parameters θ_2 to reduce the effect of erroneous updates.

Outer-loop optimization : After completing the simulated tracking episode on input video sequence $\mathcal{V}$, we obtain intermediate tracker weights θ_0, θ_1, θ_2, and θ_3. We perform meta-training on our overall tracking framework by evaluating and performing outer-loop optimization for the tracker with each intermediate weight, using different combinations of held-out datasets with ground truth annotations to encourage reduced overfitting and better generalization performance of each adaptation process. Overall outer-loop loss function $\mathcal{L}_{outer}$ for outer-loop optimization is

given as following,

$$\mathcal{L}_{outer}(\mathcal{V}) = \lambda_0 \mathcal{L}_{\theta_0}\left(\mathcal{D}^{2,3,4}\right) + \lambda_1 \mathcal{L}_{\theta_1}\left(\mathcal{D}^{2,3,4}\right) + \lambda_2 \mathcal{L}_{\theta_2}\left(\mathcal{D}^{3,4}\right) + \lambda_3 \mathcal{L}_{\theta_3}\left(\mathcal{D}^{4}\right), \tag{4.8}$$

where λ_0, λ_1, λ_2, and λ_3 are stage-wise weighting hyperparameters with sum of 1 and superscript on $\mathcal{D}$ indicates a combination of respective datasets (*i.e.*, $\mathcal{D}^{i,j} = \mathcal{D}^i \cup \mathcal{D}^j$). Each individual loss term $\mathcal{L}_{\theta_i}$ with respect to each weight θ_i is defined as,

$$\mathcal{L}_{\theta_i}(\mathcal{D}) = \sum_{(I^j, b^j) \in \mathcal{D}} \mathrm{FL}\left(c^j_{\theta_i}; \gamma_{outer}\right), \tag{4.9}$$

where focal loss FL is evaluated for binary class predictions $c^j_{\theta_i}$ obtained from the tracker with weights θ_i, using the ground truth bounding box b^j and γ_{outer} is fixed to 0.5. Each loss $\mathcal{L}_{\theta_i}$, except for $\mathcal{L}_{\theta_0}$, is evaluated with dataset $\mathcal{D}^{i+1,\cdots}$ to measure the generalization performance on unseen future frames, assessing the quality of the adaptation process conducted from the previous weights θ_{i-1} using the meta-learner network g. It also serves to facilitate the tracker with weights θ_i to make more accurate predictions for subsequent frames in $\mathcal{I}^{i+1}$ for better future self-supervised update using the estimated labels, $\hat{\mathcal{D}}^{i+1} = (\mathcal{I}^{i+1}, \hat{\mathcal{B}}^{i+1})$. Note that for all aforementioned focal loss terms FL, additional IoU loss term evaluated on b^t_θ for bounding box regression is omitted for simplicity.

4.3.2 Proposed Tracking Algorithm

Herein, we propose the visual tracking with a novel adaptive continual meta-learner. The tracking process is purposely made simple to retain the speed of the original backbone tracking algorithm while requiring as small memory overhead as possible.

Algorithm 4: Visual tracking with meta-learner

Input : Tracking algorithm f_θ with default weights θ_0
Trained meta-learner network g
Tracking sequence of length L, $\{I^1, I^2, ..., I^L\}$
Initial target bounding box coordinates b^1
Output: Target bounding box coordinates for each frame

 # Initialization at $t = 1$
Form dataset $\mathcal{D}^1 = \left(I^1, b^1\right)$ for initial adaptation
Model initialization from θ_0 using $\mathcal{D}^1$ as in Eq. (4.2) and (4.3), updating $\theta \leftarrow \theta_1$

 # For later frames in tracking sequence
for $t = 2$ *to* L **do**
  Obtain candidate boxes $b^t_{\theta_i}$ and confidence scores $c^t_{\theta_i}$ from input frame I^t as in Eq. (4.1)
  Choose box with the highest confidence score as output $\hat{b}^t$
  If an output is confident (PSR $< \tau_{on}$), store corresponding frame and output box $(I^t, \hat{b}^t)$ to dataset for online update $\hat{\mathcal{D}}^{on} = (\hat{\mathcal{I}}^{on}, \hat{\mathcal{B}}^{on})$
  if $t \mod U = 0$ **and** $|\hat{\mathcal{I}}^{on}| \geq N$ **then**
    Online update from θ_1 using θ_i and N training samples from $\hat{\mathcal{D}}^{on}$ as in Eq. (4.6) and (4.7), updating $\theta \leftarrow \theta_{i+1}$ and $i \leftarrow i + 1$
    Clear buffer for dataset $\hat{\mathcal{D}}^{on}$
  end
end

Given an input tracking sequence of length L, the proposed initial adaptation process is performed using the initial frame I^1 and bounding box b^1 to obtain initial weights θ_1 for update. During the tracking process, frames that yield output confidence values with peak-to-sidelobe ratios (PSR) smaller than $\tau_{on} = 0.7$ are considered as confident frames and stored to the dataset for online update $\hat{\mathcal{D}}^{on}$. Online update is performed every $U = 100$ frames by employing N most confident frames from the dataset $\hat{\mathcal{D}}^{on}$ and initial weights θ_1, updating the weights θ_{i-1} to θ_i, and the buffer for $\hat{\mathcal{D}}^{on}$ is cleared after every update. The overall tracking procedure is organized and described in Algorithm 4.

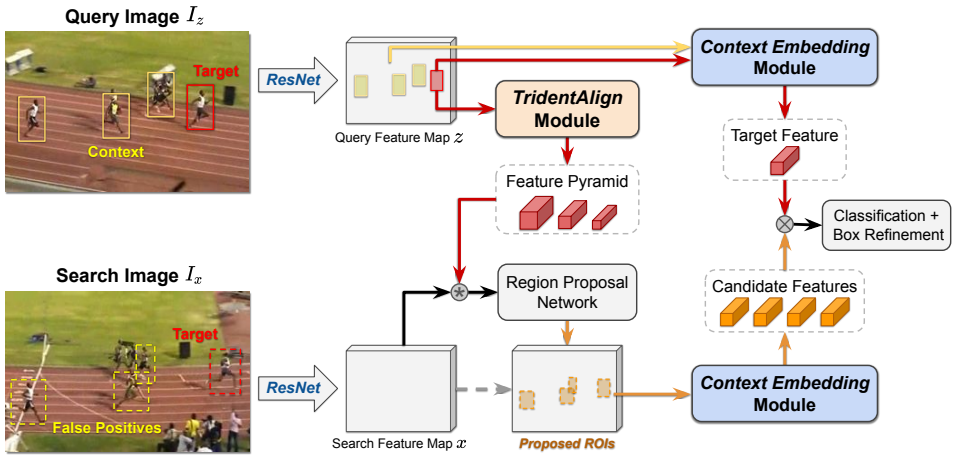


Figure 4.3: **Overview of the baseline tracking algorithm TACT.** TACT incorporates the TridentAlign module, which generates a feature pyramid representation of the target that can be utilized for better scale adaptability of the RPN. Moreover, the context embedding module modulates the locally pooled features to incorporate the global context information of a given frame, which encourages the discrimination of the target object from false positives.

4.3.3 Baseline Tracking Algorithm: TACT

In this section, we provide an overview on the backbone tracking algorithm TACT [17] used in the proposed meta-learning framework. Inspired by two-stage object detection networks [91, 38, 66], our framework largely includes two stages: the region proposal stage and classification stage. Given a pair of input RGB images, that is, the query image (initial frame) I_z and search image (current frame) I_x along with the shared backbone network $\varphi(\cdot)$, the respective feature maps $z = \varphi(I_z)$ and $x = \varphi(I_x)$ are obtained and then passed to the RPN. In the region proposal stage, the RPN generates proposals for target region-of-interest (RoIs) in the search image, given the target information obtained from the query image. Subsequently, in the classification stage, the classifier performs binary classification on the ROI proposals obtained in the previous stage, with a positive result indicating the target region and a negative indicating the background region. Figure 4.3 shows the overall flow

of the proposed method.

In the following subsections, we provide more detailed explanations of the proposed TridentAlign module used in the region proposal stage and context embedding module used in the classification stage. Then, we describe the online tracking procedure for TACT.

4.3.3.1 Region Proposal with Scale Adaptive TridentAlign

The initial target bounding box coordinates and input feature maps are given as $z, x \in \mathbb{R}^{h \times w \times c}$, where the channel dimension of the input feature maps are reduced from the original outputs of the ResNet backbone by employing 1×1 conv layers. Using these feature maps, the TridentAlign module performs multiple ROIAlign [37] operations on z with varying spatial dimensions to obtain target feature representations $z_i \in \mathbb{R}^{s_i \times s_i \times c}$, where s_i is the spatial dimension of the pooled features. These features form a feature pyramid denoted as $Z = \{z_1, z_2, ..., z_K\}$, where K is the total number of features in the feature pyramid. Then, the depth-wise cross-correlation between search feature map x and each target feature z_i in the feature pyramid Z is calculated as

$$\hat{x}_i = x \circledast z_i, \tag{4.10}$$

where $\circledast$ denotes the depth-wise cross-correlation operator with zero padding. As a result, each $\hat{x}_i \in \mathbb{R}^{h \times w \times c}$ is obtained for the corresponding z_i and is concatenated along the channel dimension to form the multi-scale correlation map $[\hat{x}_1, \hat{x}_2, ..., \hat{x}_K] = \hat{x} \in \mathbb{R}^{h \times w \times Kc}$. The correlation map is then refined as in $f_{att}(\hat{x}) \in \mathbb{R}^{h \times w \times c}$ using a self-attention block analogous to that employed in [117]. In this self-attention block, the adaptive channel and spatial attention weights are applied to focus on a specific

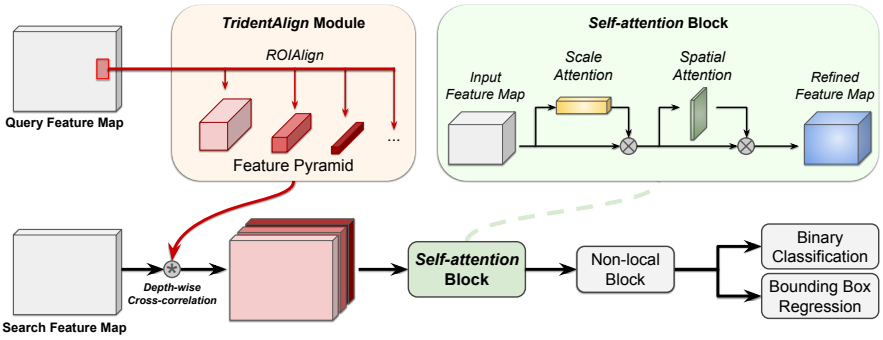


Figure 4.4: **Overview of the proposed region proposal network.** The feature pyramid representation of the target is constructed using our TridentAlign module, wherein each feature undergoes a depth-wise cross-correlation operation with the search feature map. The correlated feature maps are concatenated along the channel dimension; here, a self-attention block is used to focus more on a certain spatial area with a certain target scale. Followed by a non-local block [114] and binary classification/bounding box regression branches, ROI can be obtained.

position and target scale, followed by a 1×1 conv layer, thereby reducing the channel dimension back to c.

With the refined correlation map, we use the detection head module employed in [105], where each branch outputs binary classification labels and bounding box regression values. For a single location (i, j) inside the output map, classification labels $p_{i,j}$ with bounding box regression values $t_{i,j}$ are predicted from the respective branches. At training stage, if a location (i, j) is inside the ground-truth (GT) target bounding box, it is considered a positive sample and we assign the GT label $c^*_{i,j} = 1$ and GT regression target $t^*_{i,j} = (l^*, t^*, r^*, b^*)$, where l^*, t^*, r^*, and b^* are the distances from (i, j) to the four sides (left, top, right, bottom) of the bounding box, respectively. For negative samples, we assign $c^*_{i,j} = 0$. To train the overall RPN, we use the same loss as the one used in [105]:

$$L_{rpn}(\{p_{i,j}\}, \{t_{i,j}\}) = \frac{1}{N_{pos}} \sum_{i,j} L_{cls}(p_{i,j}, c^*_{i,j}) + \frac{\lambda}{N_{pos}} \sum_{i,j} \mathbf{1}_{\{c^*_{i,j} > 0\}} L_{reg}(t_{i,j}, t^*_{i,j}), \quad (4.11)$$

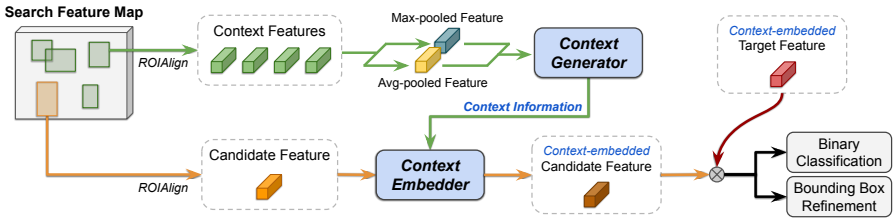


Figure 4.5: **Overview of our context embedding framework.** Given the candidate ROI and context regions, the respective feature representations are obtained by performing ROIAlign operations on each region. Using the context features, max-pooled and average-pooled features are obtained via element-wise maximum and averaging operations. The context generator receives these features to generate the global context information, which the context embedder embeds into the candidate features. Finally, the context-embedded candidate feature can be compared with the context-embedded target features for binary classification and bounding box refinement.

where N_{pos} is the number of positive samples, L_{cls} is the focal loss [66], and L_{reg} is the linear IoU loss. The loss is summed over all locations of the output map, and L_{reg} is only calculated for positive samples. Subsequently, a non-maximum suppression (NMS) operation is performed to obtain the top N candidate ROIs. The overall architecture of the RPN is illustrated in Figure 4.4.

4.3.3.2 Classification with Context-Embedded Features

Given the candidate ROIs obtained from the preceding region proposal stage, ROIAlign operations are performed on the search feature map x to obtain a set of candidate features $X = \{x_1, x_2, ..., x_N\}$, where each $x_i \in \mathbb{R}^{s \times s \times c}$ with N candidate regions. Using all of the candidate features in X to generate the global context information, we aim to modulate each feature x_i to obtain the context-embedded feature $\tilde{x}_i \in \mathbb{R}^{s \times s \times c}$. First, element-wise maximum and averaging operations are performed over all features in X to obtain max-pooled and average-pooled features, which are concatenated along the channel dimension as $x_{cxt} \in \mathbb{R}^{s \times s \times 2c}$. Then, inside the context embedding module, context generator $g_1(\cdot)$ receives x_{cxt} to generate the global

context information, and context embedder $g_2(\cdot)$ receives both the candidate feature x_i and context information from g_1 to generate the context-embedded feature $\tilde{x}_i$. The overall context embedding scheme is illustrated in Figure 4.5. For our context generator and embedder design, we test 4 variants: (1) simple concatenation, (2) simple addition, (3) CBAM [117], and (4) FILM [87] based modules. The details of each variant are listed in Table 4.1.

For the simple concatenation-based module (Table 4.1(1)), x_{cxt} is directly used as context information and the context embedder g_2 receives $[x_i, x_{cxt}]$ as input, where $[\cdot, \cdot]$ denotes concatenation along the channel dimension. In the simple addition-based module (Table 4.1(2)), context information is generated in a form of additive modulation δ, which is added to the original candidate feature x_i. For the CBAM-based module (Table 4.1(3)), context information is generated and applied to x_i as channel attention m_c and spatial attention m_s. Finally, the FILM-based module (Table 4.1(4)) modulates x_i by applying an affine transformation with coefficients γ and β.

Finally, each context-embedded candidate feature $\tilde{x}_i$ is compared with the context-embedded target feature $\tilde{z}_0 \in \mathbb{R}^{s \times s \times c}$ by element-wise multiplication as in $\tilde{x}_i \otimes \tilde{z}_0$. Binary classification and bounding box refinement operations are subsequently per-

Table 4.1: **Variants of the context embedding module.** We test 4 possible implementations of the context embedding module. For 3-layer CNNs, we use 1×1 kernels with output channels set to c, followed by ReLU activation.

	Generator $g_1(x_{cxt})$		Embedder $g_2(g_1(x_{cxt}), x_i)$
	Type	Output	Operation
(1) Simple concat.	Identity	$x_{cxt} \in \mathbb{R}^{s \times s \times 2c}$	3-layer CNN, $g_2([x_i, x_{cxt}])$
(2) Simple add.	3-layer CNN	$\delta \in \mathbb{R}^{s \times s \times c}$	$x_i + \delta$
(3) CBAM-based	3-layer CNN	$m_c \in \mathbb{R}^{1 \times 1 \times c}, m_s \in \mathbb{R}^{s \times s \times 1}$	$(x_i \otimes m_c) \otimes m_s$
(4) FILM-based	3-layer CNN	$\gamma, \beta \in \mathbb{R}^{s \times s \times c}$	$\gamma \otimes x_i + \beta$

Algorithm 5: TACT

Input : Tracking sequence of length L, $\{I^1, I^2, ..., I^L\}$
Initial target bounding box coordinates
Output: Target bounding box coordinates for each frame

 # Initialization at $t = 1$
Compute query feature map $z = \varphi(I^1)$ for initial frame I^1
Build target feature pyramid Z from z using TridentAlign
Using same z as search feature map; obtain candidate features using RPN
Obtain context-embedded target feature $\tilde{z}_0$

 # For later frames in tracking sequence
for $t = 2$ *to* L **do**
  Compute search feature map $x = \varphi(I^t)$ for frame I^t
  Using Z and x, obtain ROIs with candidate features X using RPN
  For every $x_i \in X$, calculate context-embedded feature $\tilde{x}_i$ to form $\tilde{X}$
  Compute $\tilde{x}_i \otimes \tilde{z}_0$ for every $\tilde{x}_i \in \tilde{X}$
  For every ROI, obtain softmax classification scores and box refinement values
  Choose refined ROI with highest classification score as output
end

formed. For every $\tilde{x}_i$, a classification label c_i and refined bounding box coordinates t_i are obtained. At training stage, the GT classification label $c_i^* = 1$ is assigned to candidate boxes with $\text{IoU}(t_i, t_i^*) > \tau_p$, where t_i^* is the GT box coordinates, and $c_i^* = 0$ is assigned to candidate boxes with $\text{IoU}(t_i, t_i^*) < \tau_n$. In our experiments, we use $(\tau_p, \tau_n) = (0.5, 0.4)$. To train our context embedding module and classifier, we minimize the loss function given as

$$L_{det}(\{c_i\}, \{t_i\}) = \frac{1}{N_{pos}} \sum_i L_{cls}(c_i, c_i^*) + \frac{\lambda}{N_{pos}} \sum_i \mathbf{1}_{\{c_i^* > 0\}} L_{reg}(t_i, t_i^*), \tag{4.12}$$

where the loss functions L_{cls} and L_{reg} are the same as those in Eq. (4.11).

4.3.3.3 TridentAlign and Context Embedding Tracker

Herein, we delineate **T**rident**A**lign and **C**ontext embedding **T**racker (**TACT**). The overall tracking procedure is organized and shown as Algorithm 5, where FILM-

based context embedding module is used for the final model. The tracking process is purposely made simple to achieve real-time speed. Furthermore, our tracking algorithm performs a full-frame search for every frame without any motion smoothness assumption based on the previous positions of the target; therefore, it is possible to run our tracker on a batch of multiple frames of offline videos. Increasing the batch size from 1 to 8 results in a large boost in tracking speed: we obtain 57 $\rightarrow$ 101 fps with the ResNet-18 backbone and 42 $\rightarrow$ 65 fps with the ResNet-50 backbone.

4.4 Experiments

In this section, we elaborate on the implementation details for our backbone trackers and the proposed meta-learning framework, followed by the experimental results to validate the performance gains obtained by our framework on five large-scale visual tracking benchmark datasets. We also demonstrate the results for attribute-wise and module-wise ablation experiments to further analyze the effectiveness of our proposed method.

4.4.1 Implementation Details

Evaluation environment : The proposed framework was implemented in Python 3.6 using the PyTorch 1.6 [86] machine learning library and learn2learn [3] meta-learning library. Run-time measurements were evaluated on a system with Intel i7-4790k CPU and a single Nvidia RTX 2080Ti GPU with 11GB of VRAM. Hyper-parameters were fixed for both ConTACT-18 and ConTACT-50, and for all benchmarks and experiments.

Backbone trackers : We employ two Siamese network-based backbone tracking

algorithms based on TACT [17], which are variants of a two-stage detection network. While freezing the weights of the feature extractor layers, the region proposal layers, and context embedding layers, we perform meta-training on the last ROI classification and refinement layers, starting from the original weights of TACT. We refer to these trackers **ConTACT-18** and **ConTACT-50** which are extensions of TACT-18 and TACT-50, respectively, with adaptive continual meta-learners.

Meta-learner architecture : For the meta-learner g, its sub-networks g_α, g_β, g_γ, and g_δ are all 3-layer MLPs with group normalization [118] and ReLU activation between the linear layers. The number of intermediate hidden units for each sub-network are 128, 256, 256, and 512, respectively. Assuming L is the number of layers that are involved in the adaptation process, the adaptive learning rate generator g_α takes $2L$-dimensional learning state as an input and returns L-dimensional layer-wise multipliers which are then multiplied to the per-parameter base learning rate α_{base}, similar to [64], and can be applied to each layer for SGD. The adaptive instance weight generator g_β and adaptive focusing hyperparameter generator g_γ both take confidence values $c^t_{\theta_i} \in \mathbb{R}^K$ obtained from a given frame as input and returns scalar values β^t_i and γ^t_i. The adaptive knowledge distiller g_δ takes two confidence vectors obtained from two different models as an input, $[c^t_{\theta_i}, c^t_{\theta_{i-1}}] \in \mathbb{R}^{2K}$, where $[\cdot,\cdot]$ denotes concatenation, and outputs a single scalar value δ^t_i.

Training details : Dimensions of input images and feature maps are the same as in [17] and the overall framework is trained with training splits of ImageNetVID [93], GOT-10k [49], LaSOT [29], and TrackingNet [80], from which a video sequence $\mathcal{V}$ is randomly chosen. $T = 13$ frames, in turn, are uniformly sampled inside a time window of 500 frames inside $\mathcal{V}$, along with their bounding box annotations. Among sampled frames, the first frame is used as $\mathcal{D}^1$. As for the remaining 12 frames,

$N = 4$ frames are assigned to each of $\mathcal{D}^2$, $\mathcal{D}^3$ and $\mathcal{D}^4$ in a sequential order. For all frames and annotations in $\mathcal{V}$, random image augmentations, such as Gaussian noise, blur, horizontal flips, and bounding box jittering with $\pm 5\%$ of its width and height are applied. For online adaptation, we choose a box with highest confidence from $K = 64$ candidate boxes for a given frame and use this box as self-supervision. For both initial and online adaptations, per-parameter base learning rate α_{base} are initialized to 10^{-3} and single-step SGD update is performed for faster speed. For the outer-loop optimization, Adam [54] optimizer with learning rate of 10^{-5} is used with weight decay of 10^{-5} and trained for 5×10^5 iterations with batch size of 4, where stage-wise weights $\lambda_0, \lambda_1, \lambda_2, \lambda_3$ are set to $(0.025, 0.325, 0.325, 0.325)$.

4.4.2 Quantitative Evaluation

Evaluation datasets and metrics: We conducted evaluations for our trackers on test splits of five large-scale visual tracking benchmark datasets: LaSOT [29], OxUvA [107], TLP [77], TrackingNet [80], and GOT-10k [49]. LaSOT, OxUvA, and TLP are long-term visual tracking benchmarks with average sequence lengths longer than 1 min., whereas TrackingNet and GOT-10k have shorter sequence lengths with larger number of sequences with more various semantic classes of objects.

LaSOT [29] dataset is a large-scale and long-term tracking dataset with $1,400$ video sequences for training and testing, with an average of $2,512$ frames (≈ 83 secs) in length, and are annotated with target bounding boxes. We evaluated our trackers on the test split (Protocol II) of 280 video sequences, and report the performance metrics of area-under-curve (AUC) of the success plot, location precision, and normalized precision for comparison.

OxUvA [107] dataset is focused on long-term tracking performance of a tracker

where its dev and test splits have 200 and 166 sequences, respectively, with an average length of 4,260 frames ($\approx$ 142 secs). Since target can leave and reappear in a frame under the long-term tracking scenario, trackers must report the target bounding boxes as well as whether the target is present or absent in a given frame. The performance metrics are the maximum geometric mean (MaxGM) over the true positive rate (TPR) and the true negative rate (TNR), with IoU thresholds of 0.5.

TLP [77] dataset also evaluates the long-term tracking performance where it contains 50 HD real-world videos, with average sequence length of 13,500 frames ($\approx$ 450 secs). AUC of the success plot is used as the performance metric.

TrackingNet [80] is a large-scale tracking dataset with more than $30,000$ videos gathered from YouTube, of which 511 sequences assigned as the test split. Similar to the other tracking benchmarks, location precision, normalized precision, and AUC of the success plot are used as performance metrics.

GOT-10k [49] is a dataset composed under the one-shot experiment setting where the training and test splits have disjoint set of object classes. It contains $10,000$ video sequences of which 420 are used in the test split. Performance metrics are calculated by the success rate (SR, with thresholds 0.5 and 0.75) and average overlap (AO).

Comparison to other trackers: Results for evaluation of our trackers on the LaSOT test set are provided in Table 4.2. Applying the proposed adaptive continual meta-learner on both variants of TACT, denoted as **ConTACT-18** and **ConTACT-50**, show consistent and noticeable gains on all performance metrics on both variants, while retaining real-time speeds of 50 fps and 37 fps. Both variants outperform many recent ResNet-based tracking algorithms, GlobalTrack [48], ATOM [20], DiMP [8], SiamRPN++ [59], SPLT [121], and Ocean [133]. For further

evaluation of the long-term tracking capabilities, we evaluated our tracker on the OxUvA test set and presented the results in Table 4.3. To detect the absence of the target, we simply used confidence threshold value of 0.97 to label target as absent if confidence is below this threshold. The proposed method shows substantial performance gains in MaxGM and TNR metrics compared to TACT, achieving a new state-of-the-art on OxUvA benchmark. Evaluation on relatively short-term, large-scale tracking benchmarks TrackingNet and GOT-10k are shown in Table 4.5 and 4.6. Both of our trackers show consistent performance gains on all metrics for both datasets, validating the effectiveness of our proposed meta-learner on both long-term and short-term tracking applications. Even without any temporal smoothing techniques (local search, bounding box interpolation, cosine window penalty, etc.) and hyperparameter tuning of the original tracker, our method was able to achieve competitive performance compared to other state-of-the-art trackers.

Table 4.2: Comparison on the **LaSOT** test set.

	TACT-18 → **ConTACT-18**	TACT-50 → **ConTACT-50**	GlobalTrack [48]	ATOM [20]	DiMP-50 [8]	SiamRPN++ [59]	LTMU [18]	SPLT [121]	FCOS-MAML [109]	Ocean [133]	SiamFC [7]	CLNet [27]
AUC	0.556 → 0.568	0.575 → **0.589**	0.521	0.518	0.569	0.496	0.572	0.426	0.523	0.560	0.336	0.499
Precision	0.583 → 0.597	0.607 → **0.627**	0.529	0.506	-	0.491	0.572	0.396	-	0.566	0.339	0.494
Norm. Precision	0.638 → 0.653	0.660 → **0.681**	0.599	0.576	0.650	0.569	-	0.494	-	-	0.420	0.574
FPS	57 → 50	42 → 37	6	30	43	35	13	25.7	42	25	58	45.6

Table 4.3: Comparison on the **OxUvA** test set.

(%)	TACT-50 → **ConTACT-50**	GlobalTrack [48]	SPLT [121]	MBMD [130]	LTMU [18]	EBT [136]	SiamFC$_{+R}$ [7]	SINT [103]	LCT [71]	TLD [52]	MDNet [82]	ECO-HC [19]
MaxGM	70.9 → **76.3**	60.3	62.2	54.4	75.1	28.3	45.4	32.6	39.6	43.1	34.3	31.4
TPR	**80.9** → 79.1	57.4	49.8	60.9	74.9	32.1	42.7	42.6	29.2	20.8	47.2	39.5
TNR	62.2 → 73.6	63.3	77.6	48.5	75.4	0.0	48.1	0.0	53.7	**89.5**	0.0	0.0

Table 4.4: Comparison on the **TLP** dataset.

(%)	TACT-50 → **ConTACT-50**	LTMU [18]	GlobalTrack [48]	SPLT [121]	MDNet [82]	SiamFC [7]	ECO [19]	GOTURN [41]	TLD [52]
AUC	0.508 → 0.522	**0.571**	0.520	0.416	0.372	0.237	0.205	0.200	0.122

Table 4.5: Comparison on the **TrackingNet** test set.

(%)	TACT-18 → **ConTACT-18**	TACT-50 → **ConTACT-50**	GlobalTrack [48]	ATOM [20]	DiMP-50 [8]	SiamRPN++ [59]	DASiam [137]	UPDT [9]	FCOS-MAML [109]	SiamFC [7]	ROAM++ [122]	ECO [19]
Precision	70.1 → 71.6	70.8 → **72.7**	65.6	64.8	68.7	69.4	59.1	55.7	-	53.3	62.3	49.2
Norm. Precision	78.4 → 79.9	78.8 → 80.7	75.4	77.1	80.1	80.0	73.3	70.2	**82.2**	66.6	75.4	61.8
Success	73.4 → 75.0	74.0 → **75.8**	70.4	70.3	74.0	73.3	63.8	61.1	75.7	57.1	67.0	55.4

Table 4.6: Comparison on the **GOT-10k** test set.

(%)	TACT-18 → **ConTACT-18**	TACT-50 → **ConTACT-50**	ATOM [20]	DiMP-50 [8]	SiamMask [113]	Ocean [133]	CFNet [106]	SiamFC [7]	GOTURN [41]	ROAM++ [122]	ECO [19]	CF2 [70]	MDNet [82]
$\mathbf{SR}_{0.50}$	64.8 → 66.9	66.5 → 68.7	63.4	71.7	58.7	**72.1**	40.4	35.3	37.5	53.2	30.9	29.7	30.3
$\mathbf{SR}_{0.75}$	44.7 → 47.1	47.7 → 48.7	40.2	**49.2**	36.6	-	14.4	9.8	12.4	23.6	11.1	8.8	9.9
AO	55.9 → 57.7	57.8 → 59.4	55.6	**61.1**	51.4	**61.1**	37.4	34.8	34.7	46.5	31.6	31.5	29.9

Table 4.7: Attribute-wise ablation on **LaSOT** test set.

	TACT-18	**ConTACT-18**	TACT-50	**ConTACT-50**
ARC	0.551	0.565	0.570	0.584
BC	0.477	0.494	0.486	0.509
DEF	0.586	0.599	0.609	0.629
FOC	0.456	0.469	0.478	0.491
IV	0.579	0.595	0.613	0.627
ROT	0.558	0.573	0.582	0.596

4.4.3 Analysis

4.4.3.1 Ablation Study

Attribute-wise ablation: To further analyze the effectiveness of proposed meta-learner, we show attribute-wise AUC performance on the LaSOT test set in Table 4.7, with comparison on six different challenge attributes of LaSOT. Displaying performance gains in all attributes, largest improvement comes from BC (background clutter) attribute, which validates the effectiveness of our initial and online adaptation strategy on eliminating the hard negatives while tracking.

Component-wise ablation: To verify the contribution of each component in our meta-learning framework, component-wise ablation results are shown in Table 4.8, where we sequentially remove each adaptive learning component in **(2)**-**(5)**. The results suggest that every component contributes to performance gain, where adaptive instance weighting contributes the most. Results in **(8)** show that our adaptive learning approach is effective even without any online adaptation, where only initial adaptation is adaptively performed. Regarding the online adaptation, results in **(6)**, which are obtained with naïve online finetuning with learning rate of 10^{-3} on TACT, show reduced performance possibly due to erroneous updates and overfitting. Also, results in **(7)** suggest that online adaptation from the initial weights θ_1 instead of

Table 4.8: Component-wise ablation on **LaSOT** test set.

	Variant	**ConTACT-18**	**ConTACT-50**
(1)	Full Model (ConTACT)	0.5688	0.5889
(2)	**(1)** w/o adaptive weighting	0.5649	0.5826
(3)	**(2)** w/o adaptive focal loss	0.5639	0.5808
(4)	**(3)** w/o adaptive KD	0.5619	0.5790
(5)	**(4)** w/o adaptive learning rate	0.5608	0.5773
(6)	**(5)** w/o per-param. learning rate	0.5532	0.5721
(7)	**(1)** w/o online adaptation from θ_1	0.5583	0.5762
(8)	**(1)** w/o online adaptation	0.5620	0.5804
(9)	w/o any adaptation (TACT)	0.5562	0.5755

previous weights θ_{i-1} contributes to a large performance gain, owing to reduced error accumulation.

4.4.3.2 Visualizing the Adaptive Learning

In Figure 4.6, we show five video examples of online adaptation with self-labeled training samples, where erroneous predictions in the future frames are corrected after the adaptation. During the adaptation process, β, γ and δ values predicted by the meta-learner for each training sample dynamically change. The meta-learner assigns relatively lower β and δ values to examples with less confident, uncertain predictions while the negative γ value consistently directs to focus more on maximizing the margin for confident examples, giving less attention to ambiguous examples.

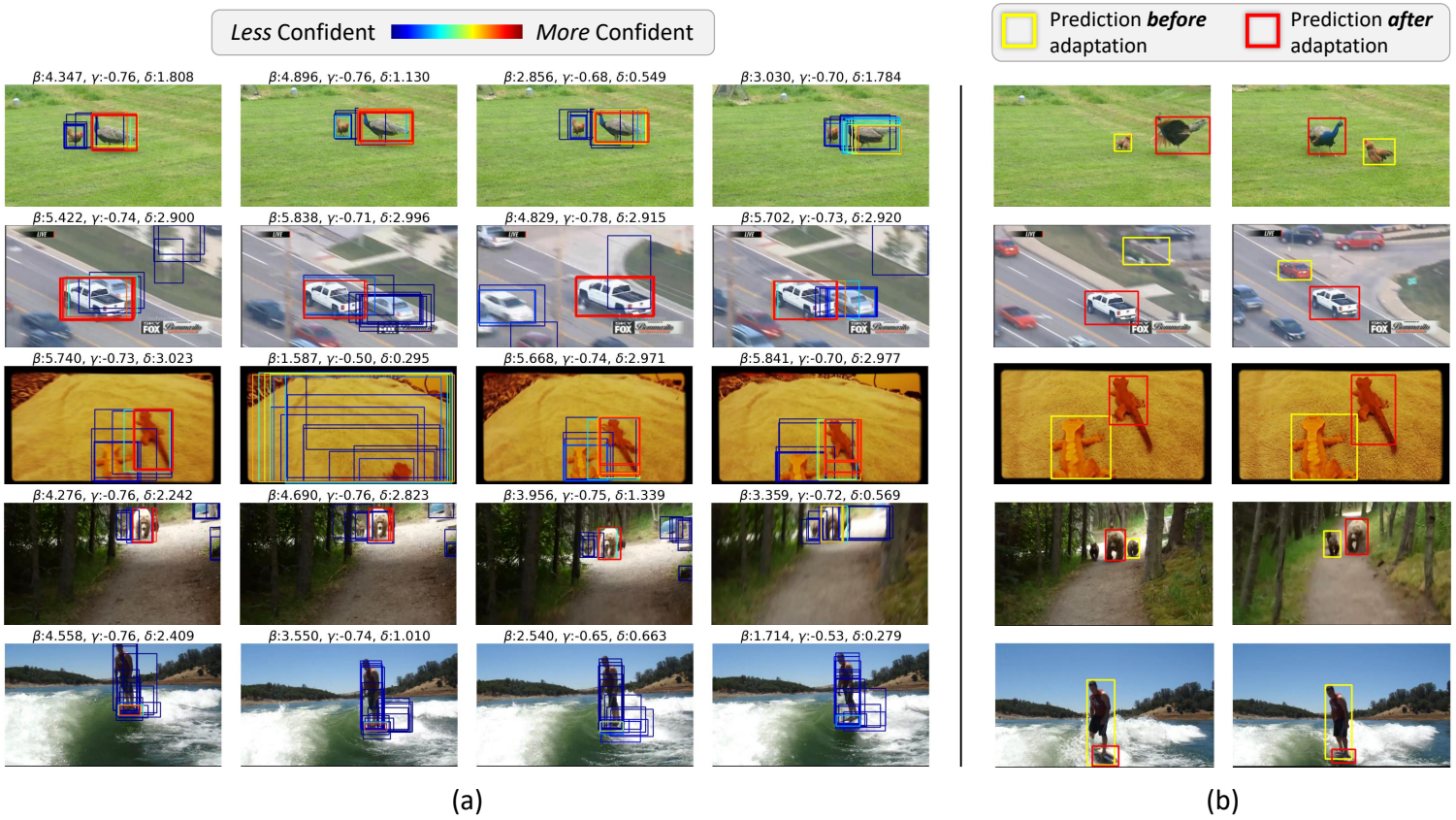


Figure 4.6: **Visualization for the effectiveness of the meta-learner.** (a) Self-labeled frames for online adaptation, with candidate boxes color-coded according to their confidence values, along with β, γ and δ values generated by the meta-learner. (b) Future frames with predicted output boxes before and after the online adaptation with corresponding frames in (a).

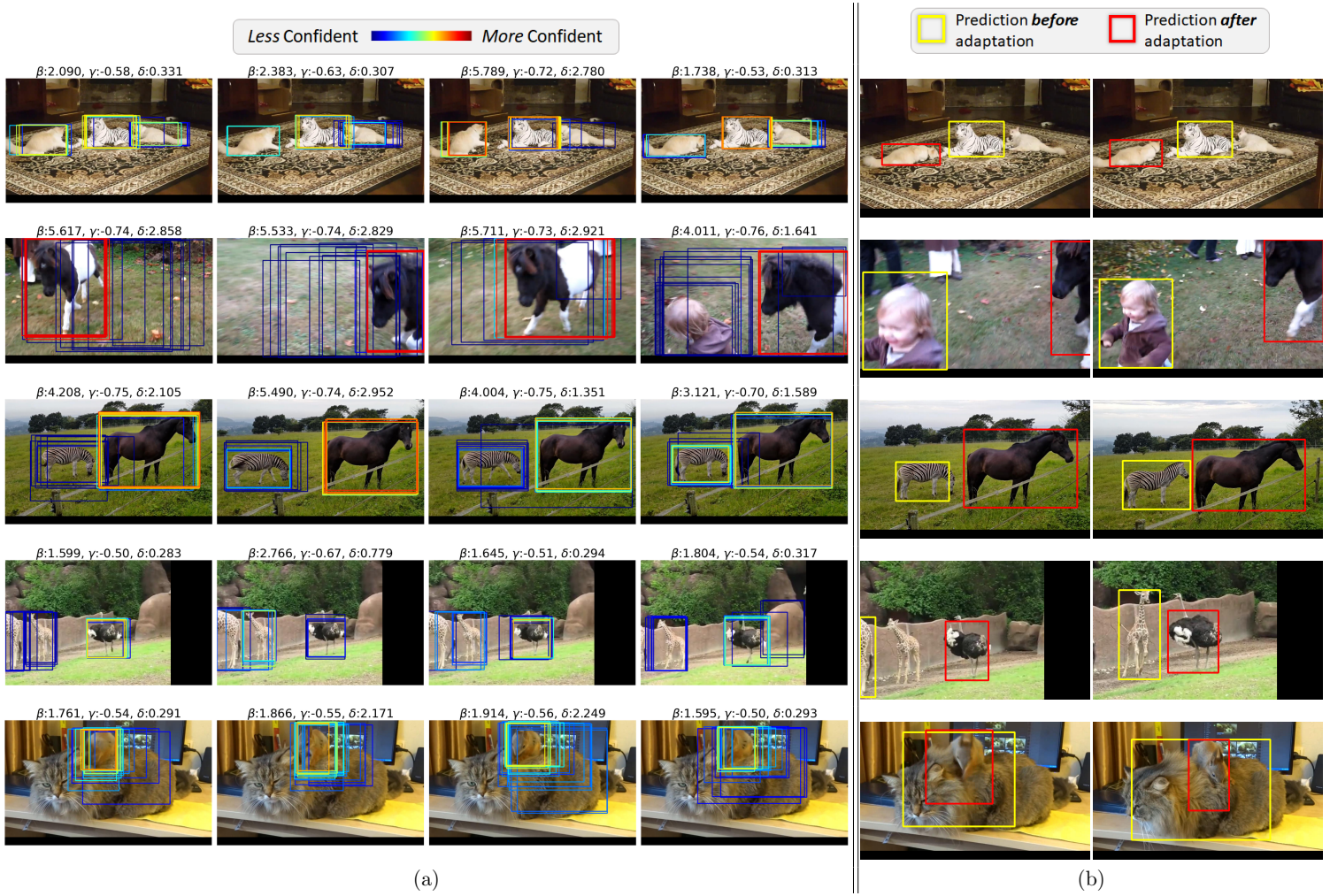


Figure 4.7: **Visualization for the effectiveness of the proposed meta-learner.** (a) Self-labeled frames for online adaptation, with candidate boxes color-coded according to their confidence values, along with β, γ, and δ values generated by the meta-learner. (b) Future frames with predicted output boxes before and after the online adaptation with corresponding frames in (a).

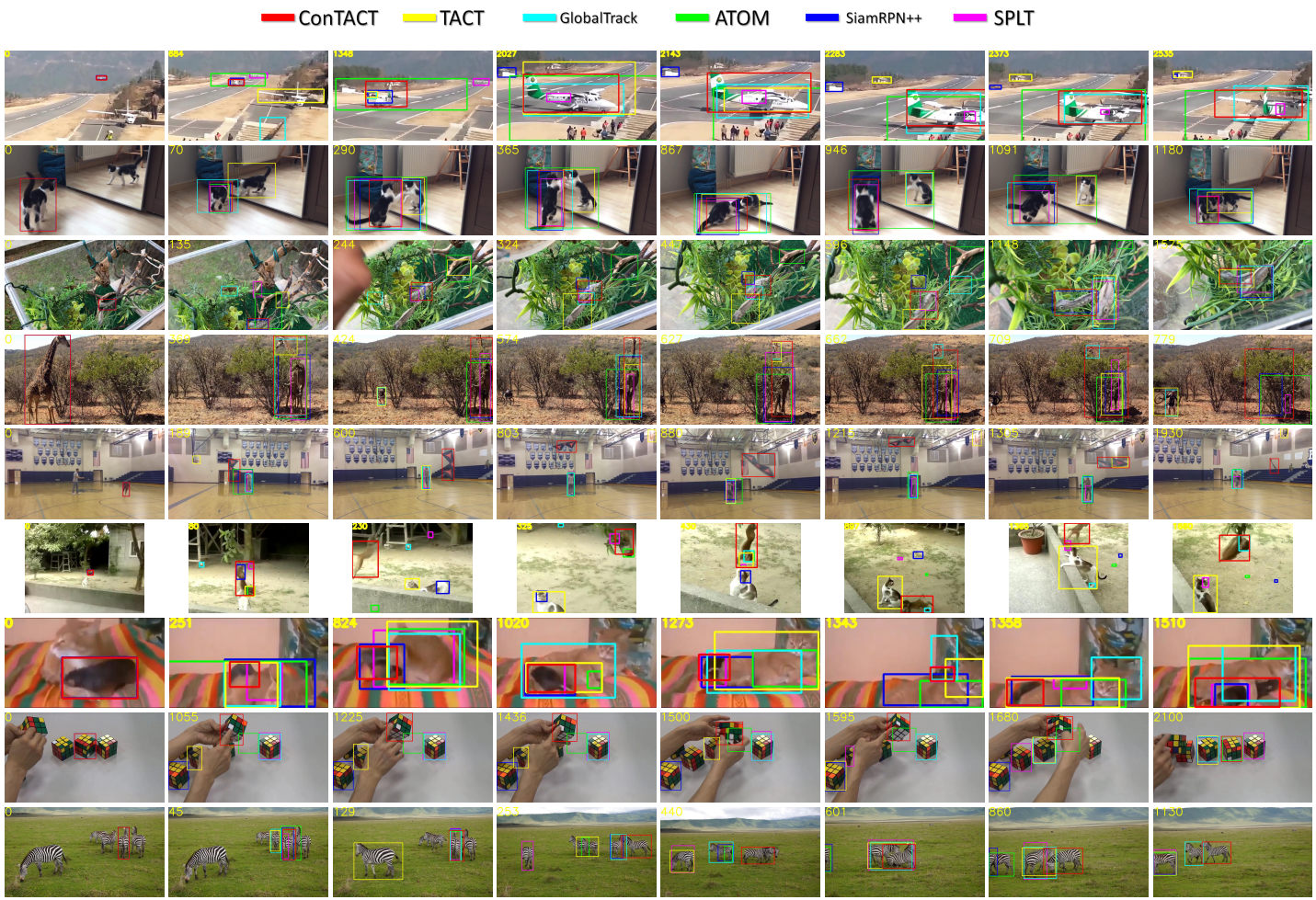


Figure 4.8: **Qualitative comparison with other trackers.** Results are shown for sequences *airplane-1, cat-20, chameleon-20, giraffe-15, kite-6, monkey-9, mouse-17, rubicCube-19, and zebra-10*. Best viewed zoomed in on a high resolution display.

4.5 Summary

In this chapter, we proposed a novel adaptive continual meta-learning framework for visual tracking that dynamically generates the hyperparameters needed for initialization and online update with self-labeled examples. Also, our continual meta-learning approach based on knowledge distillation scheme helps the tracker adapt to new examples while retaining its knowledge on previously seen examples. We apply our proposed framework to two deep learning-based tracking algorithms, where our **ConTACT-18** and **ConTACT-50** achieve noticeable performance gains and competitive results against recent state-of-the-art tracking algorithms on five large-scale visual tracking benchmarks, while running at real-time speeds.

Chapter 5

Conclusion

5.1 Summary and Contributions of the Dissertation

In this dissertation, the model adaptation problem of visual tracking algorithms was addressed. In particular, we introduced novel approaches to solve the problem of conventional model adaptation methods, which are (1) model overfitting due to small number of training samples, (2) error accumulation originating from label noise, and (3) slow tracking speed due to heavy computational load.

Three types of model adaptation approaches were proposed, based on the following: (1) reinforcement learning based exemplar selection, (2) deep meta-learning based feature space update, (3) deep adaptive continual meta-learning based adaptation. The proposed approaches introduced external, deep neural network based meta-learner networks that can handle model adaptations under various circumstances with reduced overfitting and error accumulation, while the meta-learners were designed to be light-weight and achieved real-time speeds for the overall visual tracking framework. Summaries of each proposed method and their contributions

are as follows:

• **Reinforcement learning based exemplar selection**:

A deep reinforcement learning based exemplar selection method was introduced for real-time visual tracking. The proposed tracking algorithm utilizes the model selection strategy to choose the appropriate template for tracking a given frame. The template selection strategy was learned by utilizing a simple policy gradient method on simulated training episodes randomly generated from a tracking benchmark dataset. The experimental results showed that our tracking algorithm runs at a real-time speed of 43 fps and the proposed policy network effectively decides the appropriate template for successful visual tracking, achieving noticeable performance gains over the baseline tracker.

• **Deep meta-learning based feature space update**:

A deep meta-learning based method, where the meta-learner network was used to construct the target-specific feature space using the loss gradient information obtained from training examples acquired online, was proposed. The meta-learner network provides the matching network with new information on the appearance of the target object by constructing a target-aware feature space, where the information was given in forms of convolutional kernels and channel attention weights. By eliminating the necessity of solving complex optimization tasks in the course of tracking, experimental results demonstrated that the proposed tracking algorithm performs at a real-time speed while maintaining competitive performance among other state-of-the-art tracking algorithms.

• **Deep adaptive continual meta-learning based adaptation**:

A deep continual meta-learning based method, with simultaneous modeling of the initial and online adaptations under the adaptive continual meta-learning framework,

was proposed. The proposed adaptive meta-learning strategy dynamically generates the hyperparameters needed for fast initialization and online update to achieve more robustness through adaptively regulating the learning process. In addition, our continual meta-learning approach based on knowledge distillation scheme helped the tracker adapt to new examples while retaining its knowledge on previously seen examples. We applied our proposed framework to two deep learning-based tracking algorithms to obtain noticeable performance gains and competitive results against recent state-of-the-art tracking algorithms while performing at real-time speeds.

5.2 Future Work

For future research directions, three possible suggestions can be considered to overcome the limitations of the proposed approaches.

- **Elaborate context modeling with temporal and spatial cues**:

Conventional single-target visual tracking methods lack the adequate context modeling to deal with all potential distractor objects appearing in a given scene. Although an appearance based context modeling method was proposed in [17], limitations still persist in distractor objects with identical appearances to the target, thus lacking the necessary cues to discriminate these objects. To fully discriminate and rule out all distractor objects, temporal and spatial cues that are readily available in a video sequence can be utilized for a tracking algorithm.

- **Learning to choose online training examples**:

Although instance weighting by meta-learner network was proposed and its effectiveness was shown in the experimental results of this dissertation, the proposed tracking framework still relies on a hand-crafted metric with predefined threshold

value for choosing the online training examples. To fully achieve end-to-end, learning based adaptation for visual tracking, the meta-learner network is required to make decisions on what training examples to use for online adaptation. Reinforcement learning or probabilistic sampling from Gumbel-softmax distribution are possible approaches to designing and training this meta-learner.

• **Accelerating the overall framework through neural architecture search and network pruning**:

Even though proposed tracking algorithms can run at real-time speeds, their tracking speeds were evaluated on high-end GPUs with large computing capacities. To make real-world applications of proposed tracking algorithms possible, they are required to run at real-time speeds on low-cost hardware and mobile devices. In order to achieve this performance, recent methods such as neural architecture search (NAS) and network pruning methods can be used for acceleration.

Bibliography

[1] Martın Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*, 2016.

[2] Antreas Antoniou, Harrison Edwards, and Amos Storkey. How to train your maml. In *ICLR*, 2019.

[3] Sébastien M. R. Arnold, Praateek Mahajan, Debajyoti Datta, Ian Bunner, and Konstantinos Saitas Zarkias. learn2learn: A library for meta-learning research. *arXiv preprint arXiv:2008.12284*, 2020.

[4] Boris Babenko, Ming-Hsuan Yang, and Serge Belongie. Visual tracking with online multiple instance learning. In *CVPR*, 2009.

[5] Sungyong Baik, Myungsub Choi, Janghoon Choi, Heewon Kim, and Kyoung Mu Lee. Meta-learning with adaptive hyperparameters. In *NeurIPS*, 2020.

[6] Sungyong Baik, Seokil Hong, and Kyoung Mu Lee. Learning to forget for meta-learning. In *CVPR*, 2020.

[7] Luca Bertinetto, Jack Valmadre, João F Henriques, Andrea Vedaldi, and Philip HS Torr. Fully-convolutional siamese networks for object tracking. *arXiv preprint arXiv:1606.09549*, 2016.

[8] Goutam Bhat, Martin Danelljan, Luc Van Gool, and Radu Timofte. Learning discriminative model prediction for tracking. In *ICCV*, 2019.

[9] Goutam Bhat, Joakim Johnander, Martin Danelljan, Fahad Shahbaz Khan, and Michael Felsberg. Unveiling the power of deep tracking. In *ECCV*, 2018.

[10] David S Bolme, J Ross Beveridge, Bruce A Draper, and Yui Man Lui. Visual object tracking using adaptive correlation filters. In *CVPR*, 2010.

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

[12] Dapeng Chen, Zejian Yuan, Gang Hua, Yang Wu, and Nanning Zheng. Description-discrimination collaborative tracking. In *ECCV*, 2014.

[13] Kai Chen and Wenbing Tao. Once for all: a two-flow convolutional neural network for visual tracking. *IEEE Transactions on Circuits and Systems for Video Technology*, 2017.

[14] Zedu Chen, Bineng Zhong, Guorong Li, Shengping Zhang, and Rongrong Ji. Siamese box adaptive network for visual tracking. In *CVPR*, 2020.

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

[16] Janghoon Choi, Junseok Kwon, and Kyoung Mu Lee. Deep meta learning for real-time target-aware visual tracking. In *ICCV*, 2019.

[17] Janghoon Choi, Junseok Kwon, and Kyoung Mu Lee. Visual tracking by tridentalign and context embedding. In *ACCV*, 2020.

[18] Kenan Dai, Yunhua Zhang, Dong Wang, Jianhua Li, Huchuan Lu, and Xiaoyun Yang. High-performance long-term tracking with meta-updater. In *CVPR*, 2020.

[19] Martin Danelljan, Goutam Bhat, Fahad Khan, and Michael Felsberg. Eco: Efficient convolution operators for tracking. In *CVPR*, 2017.

[20] Martin Danelljan, Goutam Bhat, Fahad Shahbaz Khan, and Michael Felsberg. Atom: Accurate tracking by overlap maximization. In *CVPR*, 2019.

[21] Martin Danelljan, Gustav Häger, Fahad Khan, and Michael Felsberg. Accurate scale estimation for robust visual tracking. In *BMVC*, 2014.

[22] Martin Danelljan, Gustav Hager, Fahad Shahbaz Khan, and Michael Felsberg. Convolutional features for correlation filter based visual tracking. In *ICCV Workshop*, 2015.

[23] Martin Danelljan, Gustav Hager, Fahad Shahbaz Khan, and Michael Felsberg. Learning spatially regularized correlation filters for visual tracking. In *CVPR*, 2015.

[24] Martin Danelljan, Andreas Robinson, Fahad Shahbaz Khan, and Michael Felsberg. Beyond correlation filters: Learning continuous convolution operators for visual tracking. In *ECCV*, 2016.

[25] Martin Danelljan, Fahad Shahbaz Khan, Michael Felsberg, and Joost Van de Weijer. Adaptive color attributes for real-time visual tracking. In *CVPR*, 2014.

[26] Xingping Dong and Jianbing Shen. Triplet loss in siamese network for object tracking. In *ECCV*, 2018.

[27] Xingping Dong, Jianbing Shen, Ling Shao, and Fatih Porikli. Clnet: A compact latent network for fast adjusting siamese trackers. In *ECCV*, 2020.

[28] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *JMLR*, 12(Jul):2121–2159, 2011.

[29] 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 *CVPR*, 2019.

[30] Heng Fan and Haibin Ling. Parallel tracking and verifying: A framework for real-time and high accuracy visual tracking. In *ICCV*, 2017.

[31] Heng Fan and Haibin Ling. Siamese cascaded region proposal networks for real-time visual tracking. In *CVPR*, 2019.

[32] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *ICML*, 2017.

[33] Qing Guo, Wei Feng, Ce Zhou, Rui Huang, Liang Wan, and Song Wang. Learning dynamic siamese network for visual object tracking. In *ICCV*, 2017.

[34] Xufeng Han, Thomas Leung, Yangqing Jia, Rahul Sukthankar, and Alexander C Berg. Matchnet: Unifying feature and metric learning for patch-based matching. In *CVPR*, 2015.

[35] Sam Hare, Amir Saffari, and Philip Torr. Struck: Structured output tracking with kernels. In *ICCV*, 2011.

[36] Anfeng He, Chong Luo, Xinmei Tian, and Wenjun Zeng. A twofold siamese network for real-time object tracking. In *CVPR*, 2018.

[37] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *ICCV*, 2017.

[38] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Spatial pyramid pooling in deep convolutional networks for visual recognition. *IEEE TPAMI*, 37(9):1904–1916, 2015.

[39] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016.

[40] David Held, Sebastian Thrun, and Silvio Savarese. Learning to track at 100 fps with deep regression networks. In *ECCV*, 2016.

[41] David Held, Sebastian Thrun, and Silvio Savarese. Learning to track at 100 fps with deep regression networks. In *ECCV*, 2016.

[42] João F Henriques, Rui Caseiro, Pedro Martins, and Jorge Batista. High-speed tracking with kernelized correlation filters. *IEEE TPAMI*, 37(3):583–596, 2015.

[43] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *NIPS Workshop*, 2014.

[44] Seunghoon Hong, Tackgeun You, Suha Kwak, and Bohyung Han. Online tracking by learning discriminative saliency map with convolutional neural network. In *ICML*, 2015.

[45] Zhibin Hong, Zhe Chen, Chaohui Wang, Xue Mei, Danil Prokhorov, and Dacheng Tao. Multi-store tracker (muster): A cognitive psychology inspired approach to object tracking. In *CVPR*, 2015.

[46] Saihui Hou, Xinyu Pan, Chen Change Loy, Zilei Wang, and Dahua Lin. Lifelong learning via progressive distillation and retrospection. In *ECCV*, 2018.

[47] Lianghua Huang, Xin Zhao, and Kaiqi Huang. Bridging the gap between detection and tracking: A unified approach. In *ICCV*, 2019.

[48] Lianghua Huang, Xin Zhao, and Kaiqi Huang. Globaltrack: A simple and strong baseline for long-term tracking. In *AAAI*, 2019.

[49] Lianghua Huang, Xin Zhao, and Kaiqi Huang. Got-10k: A large high-diversity benchmark for generic object tracking in the wild. *IEEE TPAMI*, page 1–1, 2019.

[50] Ilchae Jung, Jeany Son, Mooyeol Baek, and Bohyung Han. Real-time mdnet. In *ECCV*, 2018.

[51] Ilchae Jung, Kihyun You, Hyeonwoo Noh, Minsu Cho, and Bohyung Han. Real-time object tracking via meta-learning: Efficient model adaptation and one-shot channel pruning. In *AAAI*, 2020.

[52] Zdenek Kalal, Krystian Mikolajczyk, and Jiri Matas. Tracking-learning-detection. *IEEE TPAMI*, 34(7):1409–1422, 2011.

[53] Hamed Kiani Galoogahi, Ashton Fagg, and Simon Lucey. Learning background-aware correlation filters for visual tracking. In *ICCV*, 2017.

[54] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015.

[55] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. *PNAS*, 114(13):3521–3526, 2017.

[56] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, 2012.

[57] Junseok Kwon and Kyoung Mu Lee. Visual tracking decomposition. In *CVPR*, 2010.

[58] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

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

[60] Bo Li, Junjie Yan, Wei Wu, Zheng Zhu, and Xiaolin Hu. High performance visual tracking with siamese region proposal network. In *CVPR*, 2018.

[61] Peixia Li, Boyu Chen, Wanli Ouyang, Dong Wang, Xiaoyun Yang, and Huchuan Lu. Gradnet: Gradient-guided network for visual object tracking. In *ICCV*, 2019.

[62] Xi Li, Weiming Hu, Chunhua Shen, Zhongfei Zhang, Anthony Dick, and Anton Van Den Hengel. A survey of appearance models in visual object tracking. *ACM TIST*, 4(4):58, 2013.

[63] Zhizhong Li and Derek Hoiem. Learning without forgetting. *IEEE TPAMI*, 40(12):2935–2947, 2017.

[64] Zhenguo Li, Fengwei Zhou, Fei Chen, and Hang Li. Meta-sgd: Learning to learn quickly for few shot learning. *arXiv preprint*, arXiv:1707.09835, 2017.

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

[66] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. In *ICCV*, 2017.

[67] Xialei Liu, Hao Yang, Avinash Ravichandran, Rahul Bhotika, and Stefano Soatto. Multi-task incremental learning for object detection. *arXiv preprint arXiv:2002.05347*, 2020.

[68] Alan Lukezic, Tomas Vojir, Luka Cehovin Zajc, Jiri Matas, and Matej Kristan. Discriminative correlation filter with channel and spatial reliability. In *CVPR*, 2017.

[69] Chao Ma, Jia-Bin Huang, Xiaokang Yang, and Ming-Hsuan Yang. Hierarchical convolutional features for visual tracking. In *CVPR*, 2015.

[70] Chao Ma, Jia-Bin Huang, Xiaokang Yang, and Ming-Hsuan Yang. Hierarchical convolutional features for visual tracking. In *ICCV*, 2015.

[71] Chao Ma, Xiaokang Yang, Chongyang Zhang, and Ming-Hsuan Yang. Long-term correlation tracking. In *CVPR*, 2015.

[72] Iain Matthews, Takahiro Ishikawa, and Simon Baker. The template update problem. *IEEE TPAMI*, 26(6):810–815, 2004.

[73] Xue Mei and Haibin Ling. Robust visual tracking using l1 minimization. In *ICCV*, 2009.

[74] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy P Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. *ICML*, 2016.

[75] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. In *NIPS Workshop*. 2013.

[76] 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):529–533, 2015.

[77] Abhinav Moudgil and Vineet Gandhi. Long-term visual object tracking benchmark. In *ACCV*, 2018.

[78] Matthias Mueller, Neil Smith, and Bernard Ghanem. A benchmark and simulator for uav tracking. In *ECCV*, 2016.

[79] Matthias Mueller, Neil Smith, and Bernard Ghanem. Context-aware correlation filter tracking. In *CVPR*, 2017.

[80] 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 *ECCV*, 2018.

[81] Tsendsuren Munkhdalai and Hong Yu. Meta networks. In *ICML*, 2017.

[82] Hyeonseob Nam and Bohyung Han. Learning multi-domain convolutional neural networks for visual tracking. In *CVPR*, 2015.

[83] Alex Nichol, Joshua Achiam, and John Schulman. On first-order meta-learning algorithms. *arXiv preprint*, arXiv:1803.02999, 2018.

[84] Eunbyung Park and Alexander C Berg. Meta-tracker: Fast and robust online adaptation for visual object trackers. In *ECCV*, 2018.

[85] Eunbyung Park and Alexander C Berg. Meta-tracker: Fast and robust online adaptation for visual object trackers. In *ECCV*, 2018.

[86] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca

Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. In *NeurIPS*, 2019.

[87] Ethan Perez, Florian Strub, Harm De Vries, Vincent Dumoulin, and Aaron Courville. Film: Visual reasoning with a general conditioning layer. In *AAAI*, 2018.

[88] Yuankai Qi, Shengping Zhang, Lei Qin, Hongxun Yao, Qingming Huang, Jongwoo Lim, and Ming-Hsuan Yang. Hedged deep tracking. In *CVPR*, 2016.

[89] Sachin Ravi and Hugo Larochelle. Optimization as a model for few-shot learning. In *ICML*, 2016.

[90] Sylvestre-Alvise Rebuffi, Alexander Kolesnikov, Georg Sperl, and Christoph H Lampert. icarl: Incremental classifier and representation learning. In *CVPR*, 2017.

[91] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *NIPS*, 2015.

[92] David Ross, Jongwoo Lim, Ruei-Sung Lin, and Ming-Hsuan Yang. Incremental learning for robust visual tracking. *IJCV*, pages 125–141, 2008.

[93] 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. *IJCV*, 115(3):211–252, 2015.

[94] Konstantin Shmelkov, Cordelia Schmid, and Karteek Alahari. Incremental learning of object detectors without catastrophic forgetting. In *ICCV*, 2017.

[95] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

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

[97] Yibing Song, Chao Ma, Lijun Gong, Jiawei Zhang, Rynson Lau, and Ming-Hsuan Yang. Crest: Convolutional residual learning for visual tracking. In *ICCV*, 2017.

[98] Nitish Srivastava, Geoffrey E Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *JMLR*, 15(1):1929–1958, 2014.

[99] James Supancic, III and Deva Ramanan. Tracking as online decision-making: Learning a policy from streaming videos with reinforcement learning. In *ICCV*, 2017.

[100] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*, volume 1. 1998.

[101] Richard S Sutton, David A McAllester, Satinder P Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *NIPS*, 2000.

[102] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *CVPR*, 2015.

[103] Ran Tao, Efstratios Gavves, and Arnold WM Smeulders. Siamese instance search for tracking. In *CVPR*, 2016.

[104] Ran Tao, Efstratios Gavves, and Arnold W M Smeulders. Siamese instance search for tracking. In *CVPR*, 2016.

[105] Zhi Tian, Chunhua Shen, Hao Chen, and Tong He. Fcos: Fully convolutional one-stage object detection. In *ICCV*, 2019.

[106] Jack Valmadre, Luca Bertinetto, Joao Henriques, Andrea Vedaldi, and Philip H. S. Torr. End-to-end representation learning for correlation filter based tracking. In *CVPR*, July 2017.

[107] Jack Valmadre, Luca Bertinetto, Joao F Henriques, Ran Tao, Andrea Vedaldi, Arnold WM Smeulders, Philip HS Torr, and Efstratios Gavves. Long-term tracking in the wild: A benchmark. In *ECCV*, 2018.

[108] Oriol Vinyals, Charles Blundell, Tim Lillicrap, Koray Kavukcuoglu, and Daan Wierstra. Matching networks for one shot learning. In *NIPS*, 2016.

[109] Guangting Wang, Chong Luo, Xiaoyan Sun, Zhiwei Xiong, and Wenjun Zeng. Tracking by instance detection: A meta-learning approach. In *CVPR*, 2020.

[110] Lijun Wang, Wanli Ouyang, Xiaogang Wang, and Huchuan Lu. Visual tracking with fully convolutional networks. In *ICCV*, December 2015.

[111] Lijun Wang, Wanli Ouyang, Xiaogang Wang, and Huchuan Lu. Stct: Sequentially training convolutional networks for visual tracking. In *CVPR*, 2016.

[112] Naiyan Wang and Dit-Yan Yeung. Learning a deep compact image representation for visual tracking. In *NIPS*, 2013.

[113] Qiang Wang, Li Zhang, Luca Bertinetto, Weiming Hu, and Philip HS Torr. Fast online object tracking and segmentation: A unifying approach. In *CVPR*, 2019.

[114] Xiaolong Wang, Ross Girshick, Abhinav Gupta, and Kaiming He. Non-local neural networks. In *CVPR*, 2018.

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

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

[117] Sanghyun Woo, Jongchan Park, Joon-Young Lee, and In So Kweon. Cbam: Convolutional block attention module. In *ECCV*, 2018.

[118] Yuxin Wu and Kaiming He. Group normalization. In *ECCV*, 2018.

[119] Yi Wu, Jongwoo Lim, and Ming-Hsuan Yang. Object tracking benchmark. *IEEE TPAMI*, 37(9):1834–1848, 2015.

[120] 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 *ICCV*, 2019.

[121] Bin Yan, Haojie Zhao, Dong Wang, Huchuan Lu, and Xiaoyun Yang. 'skimming-perusal'tracking: A framework for real-time and robust long-term tracking. In *ICCV*, 2019.

[122] Tianyu Yang, Pengfei Xu, Runbo Hu, Hua Chai, and Antoni B Chan. Roam: Recurrently optimizing tracking model. In *CVPR*, 2020.

[123] Sangdoo Yun, Jongwon Choi, Youngjoon Yoo, Kimin Yun, and Jin Young Choi. Action-decision networks for visual tracking with deep reinforcement learning. In *CVPR*, 2017.

[124] Jure Žbontar and Yann LeCun. Stereo matching by training a convolutional neural network to compare image patches. *JMLR*, 17(1):2287–2318, 2016.

[125] Jianming Zhang, Shugao Ma, and Stan Sclaroff. MEEM: robust tracking via multiple experts using entropy minimization. In *ECCV*, 2014.

[126] Lichao Zhang, Abel Gonzalez-Garcia, Joost van de Weijer, Martin Danelljan, and Fahad Shahbaz Khan. Learning the model update for siamese trackers. In *ICCV*, 2019.

[127] Tianzhu Zhang, Adel Bibi, and Bernard Ghanem. In defense of sparse tracking: Circulant sparse tracker. In *CVPR*, 2016.

[128] T. Zhang, B. Ghanem, S. Liu, and N. Ahuja. Robust visual tracking via multi-task sparse learning. In *CVPR*, 2012.

[129] Tianzhu Zhang, Changsheng Xu, and Ming-Hsuan Yang. Multi-task correlation particle filter for robust object tracking. In *CVPR*, 2017.

[130] Yunhua Zhang, Dong Wang, Lijun Wang, Jinqing Qi, and Huchuan Lu. Learning regression and verification networks for long-term visual tracking. *arXiv preprint arXiv:1809.04320*, 2018.

[131] Yunhua Zhang, Lijun Wang, Jinqing Qi, Dong Wang, Mengyang Feng, and Huchuan Lu. Structured siamese network for real-time visual tracking. In *ECCV*, 2018.

[132] Zhipeng Zhang and Houwen Peng. Deeper and wider siamese networks for real-time visual tracking. In *CVPR*, 2019.

[133] Zhipeng Zhang, Houwen Peng, Jianlong Fu, Bing Li, and Weiming Hu. Ocean: Object-aware anchor-free tracking. In *ECCV*, 2020.

[134] Wei Zhong, Huchuan Lu, and Ming-Hsuan Yang. Robust object tracking via sparsity-based collaborative model. In *CVPR*, 2012.

[135] Wang Zhou, Shiyu Chang, Norma Sosa, Hendrik Hamann, and David Cox. Lifelong object detection. *arXiv preprint arXiv:2009.01129*, 2020.

[136] Gao Zhu, Fatih Porikli, and Hongdong Li. Beyond local search: Tracking objects everywhere with instance-specific proposals. In *CVPR*, 2016.

[137] Zheng Zhu, Qiang Wang, Bo Li, Wei Wu, Junjie Yan, and Weiming Hu. Distractor-aware siamese networks for visual object tracking. In *ECCV*, 2018.

국문초록

본 학위 논문에서는 물체 추적 알고리즘들에서 사용되는 모델 갱신 기법의 문제를 해결하기 위한 방법론을 제안한다. 기존 물체 추적 알고리즘들은 물체 추적 문제를 객체 검출을 통한 추적 문제 (tracking-by-detection) 로 간주하여 왔으며, 이들은 특정 물체를 검출할 수 있는 검출기 모델을 주어진 비디오의 첫번째 프레임에서 학습하여 이 모델을 사용하여 비디오의 차후 프레임들에서 목표 물체를 검출하는 방식으로 물체 추적 문제를 해결하여 왔다. 하지만 이러한 모델은 물체의 변형, 크기 변화, 가려짐, 조명 변화, 배경 물체의 등장 등의 다양한 상황변화와 물체의 외양변화에 따라 추적에 어려움이 존재한다. 이러한 문제들을 해결하기 위해 기존 물체 추적 알고리즘들은 추적 도중에 물체의 변화한 외양과 배경 물체들에 대한 새로운 정보를 추적 과정에 반영하기 위해 모델 갱신 기법을 활용하여 왔다. 하지만 이러한 기법들에서 모델 갱신 과정은 소수의 학습 표본을 사용한 최적화 문제의 해결을 통해 주로 이루어지며, 경험적으로 얻어진 정규화 기법을 활용하기 때문에 모델의 과적합 문제와 오류 누적 문제가 물체 추적 과정에서 지속되는 문제점이 있다.

전술한 문제점들을 해결하기 위해 본 논문에서는 물체 추적 문제에서 사용되는 모델 갱신 기법에 대한 새로운 접근법들을 제시한다. 이에 대해 세 가지의 모델 갱신 기법들을 제안하며 각각은: (1) 강화학습 기법에 기반한 표본 선택기법, (2) 메타학습을 기반으로 한 피쳐 공간의 갱신기법, (3) 적응적 컨티뉴얼 메타학습 기반의 갱신기법이다. 제안한 방법론들은 심층 신경망 구조에 기반한 메타러너를 도입하여 다양한 장면변화와 상황변화에 대해 학습 과정에서의 과적합 문제와 오류 누적 문제를 줄이고자 하였으며,

메타러너는 경량화된 구조로 설계되어 전체 물체 추적 프레임워크가 실시간 속도로 동작할 수 있게 하였다.

첫번째로, 정책 네트워크를 메타러너로 활용하는 강화학습 기반의 표본 선택기법을 제안한다. 정책 네트워크는 주어진 장면에서 목표 물체를 검출하기 위해 여러 표본 중 사용하기에 가장 적합한 표본을 선택하는 의사결정을 학습한다. 다음으로, 메타러너 네트워크를 활용한 메타학습 기법을 제안하며, 여기서 메타러너는 손실함수의 그래디언트 정보를 활용해 목표 물체에 특화된 피쳐 공간을 구축한다. 메타러너 네트워크는 물체 추적기에 대해 적응적인 가중치와 채널 어텐션의 형태로 새로운 정보를 제공한다. 마지막으로, 컨티뉴얼 메타학습 기반의 기법에서는 초기 갱신과정과 온라인 갱신과정 두 가지 모두를 적응형 컨티뉴얼 메타학습 프레임워크로 모델한다. 메타러너는 물체 추적기가 새로운 학습 표본을 배울지, 아니면 기존 지식을 유지할지를 선택할 수 있도록 적응적으로 학습과정을 제어하는 역할을 학습한다.

제안하는 기법들을 물체 추적 알고리즘들에 적용해본 결과 유의미한 성능 향상을 얻을 수 있었으며, 면밀한 실험적 분석과 구성요소별 분석을 통해 유효성을 검증하였다. 또한 저명하면서 널리 사용되는 물체 추적 벤치마크들을 활용한 비교실험을 통해 다른 최신 물체추적 알고리즘들과 비교해서도 실시간 속도로 효율적으로 동작하면서 우수한 성능을 보여줌을 확인하였다.

주요어: 물체 추적, 객체 추적, 강화학습, 메타학습, 컨티뉴얼 학습, 준지도학습

학번: 2013-20896