Joint Representation and Truncated Inference Learning for Correlation Filter based Tracking

Yingjie Yao[1][0000−0002−3533−1569], Xiaohe Wu[1][0000−0001−6884−9121], Lei Zhang[2][0000−0002−4424−4942], Shiguang Shan[3][0000−0002−8348−392X], and Wangmeng Zuo[1(✉)][0000−0002−3330−783X]

[1] Harbin Institute of Technology, Harbin 150001, China
[2] University of Pittsburgh, 3362 Fifth Avenue, Pittsburgh, PA 15213
[3] Institute of Computing Technology, CAS, Beijing, 100049, China
{yaoyoyogurt,xhwu.cpsl.hit,cszhanglei}@gmail.com,
sgshan@ict.ac.cn, wmzuo@hit.edu.cn

Abstract Correlation filter (CF) based trackers generally include two modules, i.e., feature representation and on-line model adaptation. In existing off-line deep learning models for CF trackers, the model adaptation usually is either abandoned or has closed-form solution to make it feasible to learn deep representation in an end-to-end manner. However, such solutions fail to exploit the advances in CF models, and cannot achieve competitive accuracy in comparison with the state-of-the-art CF trackers. In this paper, we investigate the joint learning of deep representation and model adaptation, where an updater network is introduced for better tracking on future frame by taking current frame representation, tracking result, and last CF tracker as input. By modeling the representor as convolutional neural network (CNN), we truncate the alternating direction method of multipliers (ADMM) and interpret it as a deep network of updater, resulting in our model for learning representation and truncated inference (RTINet). Experiments demonstrate that our RTINet tracker achieves favorable tracking accuracy against the state-of-the-art trackers and its rapid version can run at a real-time speed of 24 fps. The code and pre-trained models will be publicly available at `https://github.com/tourmaline612/RTINet`.

Keywords: Visual tracking · correlation filters · convolutional neural networks · unrolled optimization

1 Introduction

In recent years, correlation filters (CFs) have achieved noteworthy advances as well as state-of-the-art performance in visual tracking. Generally, the CF-based approaches learn CFs on feature representation for model adaptation along with an image sequence. Therefore, the advancement of CF-based tracking performance is mainly driven by the improvement on both feature representation and CF learning model. The development of feature representation has witnessed the

evolution from handcrafted HOG [16] and ColorNames (CN) [11] to deep convolutional neural network (CNN) features [7,22,26]. And their combination has also been adopted [6,10]. Meanwhile, the learning models have also been continuously improved with the introduction of spatial regularization [7–9], continuous convolution [10], target response adaptation [2], context regularization [23], temporal regularization [20], and other sophisticated learning models [6, 17, 34].

Motivated by the unprecedented success of CNNs [14, 19, 27, 28] in computer vision, it is encouraging to study the off-line training of deep CNNs for feature representation and model adaptation in CF trackers. Unfortunately, model adaptation in CF tracking usually requires to solve a complex optimization problem, and is not trivial to be off-line trained together with deep representation. To enable off-line training of deep representation specified for visual tracking, the Siamese network solutions [1, 4, 29] are suggested to bypass the model adaptation by learning a matcher to discriminate whether a patch is matched with the exemplar image annotated in the first frame. In [1, 4, 29], the tracker is fixed since the first frame, and cannot adapt to the appearance temporal variation of target. For joint off-training of deep representation and model adaptation, Valmadre et al. [30] adopt the original CF form due to its model adaptation has the closed-form solution and can be interpreted as a differentiable CNN layer. Instead of directly taking model adaptation into account, Guo et al. [13] suggest a dynamic Siamese network for modeling temporal variation, while Choi et al. [5] exploit the forward-pass of meta-learner network to provide new appearance information to Siamese network. These approaches, however, fail to exploit the continuous improvement on CF models [7, 8, 10, 17], and even may not achieve comparable tracking accuracy with the deployment of advanced CF models on deep features pre-trained for classification and detection tasks.

In response to the aforementioned issues, this paper presents a bi-level optimization formulation as well as a RTINet architecture for joint off-line learning of deep representation and model adaptation in CF-based tracking. To exploit the advances in CF tracking, the lower-level task adopts a more sophisticated CF model [17] by incorporating background-aware modeling, which can learn CFs with limited boundary effect from large spatial supports. And we define the upper-level objective on future frame for task-driven learning and improving the tracking accuracy. With unrolled optimization, we truncate the alternating direction method of multipliers (ADMM) for solving the lower-level task to form our RTINet, which can be interpreted as an updater network based on the deep representation provided by another representor network. Therefore, our RTINet model enables the end-to-end off-line training of both deep representation and truncated inference. Furthermore, task-driven learning of truncated inference is also helpful in improving the effectiveness of the baseline CF tracker [30]. Experiments show that combining CNN with advanced CF tracker can benefit tracking performance, and the joint learning of deep representation and truncated inference also improves tracking accuracy. In comparison with state-of-the-art trackers, our RTINet tracker achieves favorable tracking accuracy, and its rapid version can achieve a real time speed of 24 fps.

To sum up, the contribution of this work is three-fold:

1. We present a framework, i.e., RTINet, for off-line training of deep representation and model adaptation. Instead of combining CNN with the standard CF tracker [30], we show that the combination with the advanced CF tracker (i.e., BACF [17]) can improve the tracking performance with a large margin.
2. The model adaptation of the advanced CFs generally requires to solve a complex optimization problem, making it difficult to jointly train the representor and updater networks. To tackle this issue, we design the updater network by unrolling the ADMM algorithm, and define the loss on future frame to guide the model learning.
3. Experiments show that our RTINet achieves favorable accuracy against state-of-the-art trackers, while its rapid version can perform at real time speed.

2 Related Work

Deep CNNs have demonstrated excellent performance in many challenging vision tasks [12, 27], and inspire numerous works to adopt deep features in CF based trackers [6, 7, 22]. These methods simply use the feature representation generated by CNNs pre-trained for image classification, which, however, are not tailored to visual tracking. Several Siamese networks, e.g., SINT [29], GOTURN [15], and SiameseFC [1], have been exploited for the off-line learning of CNN feature extractor for tracking, but both the feature extractor and tracker are fixed for the first frame, making them generally perform inferior to state-of-the-arts.

As a remedy, Guo et al. [13] and Choi et al. [5] learn to on-line update the feature extractor for adapting to appearance variation during tracking. Instead of learning to update the feature extractor, Valmadre et al. [30] adopt the simple CF model to off-line learn deep representation. Due to that the original CF has the closed-form solution, it can be interpreted as a differentiable CNN layer and enables the joint learning of deep representation and model adaptation. These aforementioned approaches fail to exploit the continuous improvement on CF models [7, 8, 10, 17], and cannot compete with the advanced CF models based on deep features.

Another related work is the meta-tracker by Park et al. [25] which automatically learns fast gradient directions for online model adaptation of an existing tracker (e.g., MDNet [24]). In contrast, our RTINet focuses on the joint off-line learning of deep representation and model adaptation in CF-based tracking. Moreover, most advanced CF trackers are formulated as constrained optimization, which cannot be readily solved by gradient descent as meta-tracker [25] does. Therefore, we truncate the ADMM algorithm for solving BACF [10, 17] to design the updater network, and then present our RTINet that enables the end-to-end off-line training of both deep representation and truncated inference. Furthermore, off-line learning of truncated inference also benefits the improvement on effectiveness of the baseline optimization algorithm [32, 33].

3 Proposed Method

In this section, we present our RTINet approach for joint off-line training of deep representation and model adaptation in CF trackers. To this end, we first briefly revisit a recent CF tracker, i.e., BACF [17], to deliver some insights, and then introduce the formulation, network architecture, and learning of our RTINet.

3.1 Revisiting BACF

Let $\mathbf{z}_t \in \mathbb{R}^{m\times n\times L}$ and $\mathbf{f}_t$ denote the feature representation of the current frame $\mathbf{x}_t$, and the CFs adopted at frame t, respectively. In CF based trackers, tracking can be performed by first computing the response map $\sum_{l=1}^{L} \mathbf{z}_{t,l} \star \mathbf{f}_{t,l}$ as the cross-correlation between $\mathbf{z}_t$ and $\mathbf{f}_t$, and then locating the target based on the maximum of the response map. Here, $\star$ denotes the convolution operator, and the cross-correlation can be efficiently performed with the Fast Fourier Transform (FFT), making CFs very encouraging and intensively studied in visual tracking. The original CF model updates the CFs by solving the following problem,

$$\min_{\mathbf{f}} \frac{1}{2}\left\|\mathbf{y}_t - \sum_{l=1}^{L} \mathbf{z}_{t,l} \star \mathbf{f}_l\right\|^2 + \frac{\lambda}{2}\sum_{l=1}^{L}\|\mathbf{f}_l\|^2, \tag{1}$$

where $\mathbf{y}_t$ is a Gaussian shaped function based on the tracking result at frame t, and λ is the regularization parameter.

Recently, many advanced CF models have been suggested to improve the original CF, resulting in continuous performance improvement on visual tracking. Here we take BACF [17] as an example, which learns CFs by better exploiting real negative samples via background-aware modeling. The BACF model can be equivalently formulated as,

$$\min_{\mathbf{f},\mathbf{h}} \frac{1}{2}\left\|\mathbf{y}_t - \sum_{l=1}^{L} \mathbf{z}_{t,l} \star \mathbf{f}_l\right\|^2 + \frac{\lambda}{2}\|\mathbf{h}\|^2, \text{ s.t. } \mathbf{f}_l = \mathbf{M}^\top \mathbf{h}_l, \tag{2}$$

where $\mathbf{M}$ is a binary selection matrix to crop the center patch of an image. The BACF model can be efficiently solved using the Alternating Direction Method of Multipliers (ADMM). Accordingly, the augmented Lagrangian function of Eqn. (2) can be expressed as,

$$L(\mathbf{f},\mathbf{h},\boldsymbol{\mu}) = \frac{1}{2}\left\|\mathbf{y}_t - \sum_{l=1}^{L} \mathbf{z}_{t,l} \star \mathbf{f}_l\right\|^2 + \frac{\lambda}{2}\|\mathbf{h}\|^2 + \sum_{l=1}^{L} \boldsymbol{\mu}_l^\top (\mathbf{f}_l - \mathbf{M}^\top \mathbf{h}_l) + \frac{\rho}{2}\sum_{l=1}^{L}\|\mathbf{f}_l - \mathbf{M}^\top \mathbf{h}_l\|^2, \tag{3}$$

where $\boldsymbol{\mu}$ denotes the Lagrange multiplier, and ρ is the penalty parameter. By introducing $\mathbf{g} = \frac{1}{\rho}\boldsymbol{\mu}$, the optimization on $\{\mathbf{f},\mathbf{h}\}$ of Eqn. (3) can be equivalently formed as,

$$L(\mathbf{f},\mathbf{h},\mathbf{g}) = \frac{1}{2}\left\|\mathbf{y}_t - \sum_{l=1}^{L} \mathbf{z}_{t,l} \star \mathbf{f}_l\right\|^2 + \frac{\lambda}{2}\|\mathbf{h}\|^2 + \frac{\rho}{2}\sum_{l=1}^{L}\|\mathbf{f}_l - \mathbf{M}^\top \mathbf{h}_l + \mathbf{g}_l\|^2. \tag{4}$$

The ADMM algorithm can then be applied to alternatingly update $\mathbf{h}$, $\mathbf{g}$ and $\mathbf{f}$,

$$\begin{cases} \mathbf{h}^{(k+1)} = \arg\min_{\mathbf{h}} \frac{\lambda}{2}\|\mathbf{h}\|^2 + \frac{\rho}{2}\sum_{l=1}^{L}\|\mathbf{f}_l^{(k)} - \mathbf{M}^\top\mathbf{h}_l + \mathbf{g}_l^{(k)}\|^2 \\ \mathbf{g}_l^{(k+1)} = \mathbf{g}_l^{(k)} + \mathbf{f}_l^{(k)} - \mathbf{M}^\top\mathbf{h}_l^{(k+1)} \\ \mathbf{f}^{(k+1)} = \arg\min_{\mathbf{f}} \frac{1}{2}\left\|\mathbf{y}_t - \sum_{l=1}^{L}\mathbf{z}_{t,l} \star \mathbf{f}_l\right\|^2 + \frac{\rho}{2}\sum_{l=1}^{L}\|\mathbf{f}_l - \mathbf{M}^\top\mathbf{h}_l^{(k+1)} + \mathbf{g}_l^{(k+1)}\|^2 \end{cases} \tag{5}$$

We note that the subproblems on $\mathbf{f}^{(k+1)}$ and $\mathbf{h}^{(k+1)}$ have closed-form solutions. Once the solution $\mathbf{f}^*$ to Eqn. (2) is obtained, the CFs adopted at frame $t+1$ can then be attained with the linear interpolation updating rule defined as,

$$\mathbf{f}_{t+1} = (1-\eta)\mathbf{f}_t + \eta\mathbf{f}^* \tag{6}$$

where η denotes the on-line adaptation rate.

Based on the formulation and optimization of BACF [17], we further explain its motivations to the extension of CFNet [30] and the joint off-line learning of deep representation and model adaptation:

1. In CFNet, the deep representation is integrated with the simplest CF tracker [16] for offline training. Note that many advanced CF models, e.g., BACF [17], can significantly outperform the simple CF in terms of tracking accuracy. Thus, it is natural to conjecture that the combination of deep representation and BACF can result in improved tracking performance.
2. One reason that CFNet only considers the conventional CF is that it has closed-form solution and can be interpreted as a differentiable CNN layer. As for BACF, the solution to Eqn. (2) defines an implicit function of the feature representation $\mathbf{z}_t$ and model parameter λ, restricting its integration with CNN representation. Fortunately, when the number of iterations is fixed (i.e., truncated inference [32,33]), the $\mathbf{f}_{t+1}$ from Eqns. (5) and (6) can then be represented as an explicit function of the feature representation and model parameter. Therefore, by unrolling the ADMM optimization of BACF, it is feasible to facilitate the end-to-end off-line learning of truncated inference for visual tracking.
3. Moreover, BACF is performed on the handcrafted features in [17]. Denote by $\psi(\cdot\,;\mathbf{W}_F)$, a fully convolutional network with parameters $\mathbf{W}_F$. Thus, by letting $\mathbf{z}_t = \psi(\mathbf{x}_t;\mathbf{W}_F)$, both deep representation and truncated inference can be jointly off-line learned from annotated sequences.

Motivated by the above discussions, we in the following first introduce a bi-level optimization framework for joint learning of deep representation and truncated inference, and then present the architecture and learning of our RTINet.

3.2 Model Formulation

Suppose $\mathbf{z}_t = \psi(\mathbf{x}_t;\mathbf{W}_F)$ is the deep representation of $\mathbf{x}_t$, where $\mathbf{W}_F$ denotes the parameters of the representor network $\psi(\cdot\,;\mathbf{W}_F)$. Naturally, we require that

the learned CFs $\mathbf{f}_{t+1} = \eta\mathbf{f}^* + (1-\eta)\mathbf{f}_t$ should be effective in tracking the target of the future frame. Thus, the integration of BACF and deep representation can be formulated as a bi-level optimization problem,

$$\begin{aligned} &\min_{\lambda,\rho,\mathbf{M},\eta} \left\| \mathbf{y}_{t+1} - \sum_{l=1}^{L} \mathbf{z}_{t+1,l} \star (\eta\mathbf{f}_l^* + (1-\eta)\mathbf{f}_{t,l}) \right\|^2, \\ &\text{s.t. } \mathbf{f}^* = \arg\min_{\mathbf{f}} \left\| \mathbf{y}_t - \sum_{l=1}^{L} \mathbf{z}_{t,l} \star \mathbf{f}_l \right\|^2 + \lambda\|\mathbf{h}\|^2, \\ &\text{s.t. } \mathbf{f}_l = \mathbf{M}^\top \mathbf{h}_l \end{aligned} \tag{7}$$

However, $\mathbf{f}^*$ defines an implicit function of $\mathbf{z}_t$, and $\mathbf{f}_{t+1}$, making it difficult to compute the gradient.

With the unrolled ADMM optimization, when the number of iterations K is fixed, all the $\mathbf{f}^{(1)}$, ..., $\mathbf{f}^{(K)}$, and $\mathbf{f}_{t+1}$ can be represented as the functions of $\mathbf{z}_t$, $\mathbf{y}_t$, and $\mathbf{f}_t$. For joint learning of deep representation and truncated inference, we also slightly modify the BACF model and ADMM algorithm to make that the model parameters λ and $\mathbf{M}$, algorithm parameters ρ and η are both iteration-wise and learnable, i.e., $\Theta = \{\Theta^{(1)}, ..., \Theta^{(K)}\}$ with $\Theta^{(k)} = \{\lambda^{(k)}, \mathbf{M}^{(k)}, \rho^{(k)}, \eta^{(k)}\}$. To improve the robustness of the learned tracker, we require that $\mathbf{f}_{t+1}$ can also be applied to the $(t+1)$-th frame. To ease the training, we further introduce $\mathbf{f}_{t+1}^{(k)} = \eta^{(k)}\mathbf{f}^{(k)} + (1-\eta^{(k)})\mathbf{f}_t$, and require that $\mathbf{f}_{t+1}^{(k)}$ also performs well. Taking all the aforementioned factors into account, we present the whole RTINet model for joint learning of representation and truncated inference as

$$\min \mathcal{L}(\mathbf{W}_F, \Theta) = \sum_{k=1}^{K} \left\| \mathbf{y}_{t+1} - \sum_{l=1}^{L} \psi_l(\mathbf{x}_{t+1}; \mathbf{W}_F) \star \mathbf{f}_{t+1,l}^{(k)} \right\|^2 \tag{8}$$

where

$$\mathbf{f}_{t+1}^{(k)} = F_{Int}(\mathbf{f}^{(k)}, \mathbf{f}_t; \eta^{(k)}) = \eta^{(k)}\mathbf{f}^{(k)} + (1-\eta^{(k)})\mathbf{f}_t\,, \tag{9}$$

$$\begin{cases} \mathbf{h}^{(k)} = F_{\mathbf{h}}(\mathbf{f}^{(k-1)}, \mathbf{g}^{(k-1)}; \lambda^{(k)}, \rho^{(k)}, \mathbf{M}^{(k)}) & \text{(10a)} \\ \quad = \left(\lambda^{(k)}\mathbf{I} + \rho^{(k)}\left(\mathbf{M}^{(k)}\mathbf{M}^{(k)\top} \otimes \mathbf{I}_L\right)\right)^{-1} \rho^{(k)}\left(\mathbf{M}^{(k)} \otimes \mathbf{I}_L\right)\left(\mathbf{f}^{(k-1)} + \mathbf{g}^{(k-1)}\right) & \\ \mathbf{g}_l^{(k)} = F_{\mathbf{g}}(\mathbf{g}^{(k-1)}, \mathbf{f}^{(k-1)}, \mathbf{h}^{(k)}; \mathbf{M}^{(k)}) & \text{(10b)} \\ \quad = \mathbf{g}_l^{(k-1)} + \mathbf{f}_l^{(k-1)} - \mathbf{M}^{(k)\top}\mathbf{h}_l^{(k)} & \\ \hat{\mathbf{f}}_l^{(k)} = F_{\mathbf{f}}(\mathbf{z}_t, \mathbf{y}_t, \mathbf{g}^{(k)}, \mathbf{h}^{(k)}; \rho^{(k)}, \mathbf{M}^{(k)}) & \text{(10c)} \\ \quad = \dfrac{\hat{\mathbf{z}}_{t,l}^* \circ \hat{\mathbf{q}}}{\rho^{(k)} + \sum_{l=1}^{L} \hat{\mathbf{z}}_{t,l}^* \circ \hat{\mathbf{z}}_{t,l}}, \ \hat{\mathbf{q}} = \rho^{(k)}\hat{\mathbf{h}}_l^{(k)} - \rho^{(k)}\hat{\mathbf{g}}_l^{(k)} + \hat{\mathbf{z}}_{t,l} \circ \hat{\mathbf{y}}_t & \end{cases}$$

where $\hat{\cdot} = \mathcal{F}(\cdot)$ denotes the FFT of a signal, $\otimes$ indicates the Kronecker product, $\mathbf{I}_L$ is an identity matrix of size $L \times L$ and $\hat{\mathbf{h}}_l^{(k)} = \mathcal{F}(\mathbf{M}^{(k)\top}\mathbf{h}_l^{(k)})$. $\mathbf{f}^{(k)}$ can be

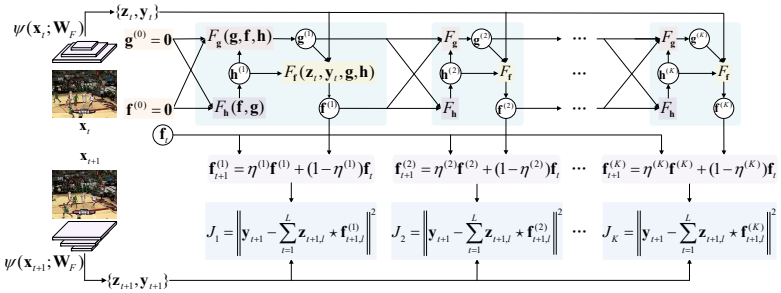


Figure 1. Overview of the RTINet architecture, which includes a representor network and an updater network. In the inference learning, we compute $\mathbf{h}$, $\mathbf{g}$ and $\mathbf{f}$ recursively following Eqns. (9)∼(10c) in each stage.

further obtained by the inverse FFT of $\hat{\mathbf{f}}^{(k)}$. In the first iteration, $\mathbf{f}^{(0)}$ and $\mathbf{g}^{(0)}$ are initialized as zeros.

To sum up, our RTINet consists of two subnetworks: (i) a representor network to generate deep representation $\mathbf{z}_t = \psi(\mathbf{x}_t; \mathbf{W}_F)$, and (ii) an updater network to update the CF model $\mathbf{f}_{t+1} = \mathbf{f}_{t+1}^{(K)} = \phi(\mathbf{z}_t, \mathbf{y}_t, \mathbf{f}_t; \Theta)$. While the representor network adopts the architecture of fully convolutional network, the updater network is recursively defined based on Eqns. (9)∼(10c). More detailed explanation on the representor and updater architecture will be given in the next subsection.

3.3 Architecture of RTINet

Fig. 1 provides an overview of the RTINet architecture, which includes a representor network and a updater network. For the representor network $\psi(\cdot; \mathbf{W}_F)$, we adopt the first three convolution (conv) layers of the VGG-M [3]. ReLU nonlinearity and local response normalization are employed after each convolution operation, and the pooling operation is deployed for the first two conv layers. To handle different sizes of targets, we resize the patches to 224×224 as inputs and produce the feature map with the size of $13 \times 13 \times 512$.

As for the updater network $\phi(\mathbf{z}_t, \mathbf{y}_t, \mathbf{f}_t; \Theta)$, we follow the unrolled ADMM optimization to design the network architecture. As shown in Fig. 1, given $\{\mathbf{z}_t, \mathbf{y}_t\}$, we initialize $\mathbf{f}^{(0)} = \mathbf{0}$ and $\mathbf{g}^{(0)} = \mathbf{0}$. In the first stage of the updater network, (i) the node $F_{\mathbf{h}}(\mathbf{f}, \mathbf{g})$ takes $\mathbf{f}^{(0)}$ and $\mathbf{g}^{(0)}$ as input to generate $\mathbf{h}^{(1)}$, (ii) the node $F_{\mathbf{g}}(\mathbf{g}, \mathbf{f}, \mathbf{h})$ takes $\mathbf{g}^{(0)}$, $\mathbf{f}^{(0)}$, and $\mathbf{h}^{(1)}$ as input to generate $\mathbf{g}^{(1)}$, and finally (iii) the node $F_{\mathbf{f}}(\mathbf{z}, \mathbf{y}, \mathbf{g}, \mathbf{h})$ takes $\mathbf{z}_t$, $\mathbf{y}_t$, $\mathbf{g}^{(1)}$ and $\mathbf{h}^{(1)}$ as input to generate $\mathbf{f}^{(1)}$. By repeating K stages, we can obtain $\mathbf{f}^{(K)}$, and then the node $F_{Int}(\mathbf{f}, \mathbf{f}_t)$ takes $\mathbf{f}^{(K)}$ and $\mathbf{f}_t$ as input to generate $\mathbf{f}_{t+1}$. Note that all the nodes $F_{\mathbf{g}}$, $F_{\mathbf{h}}$, $F_{\mathbf{f}}$, and F_{Int} are differentiable. Thus, with the annotated video sequences, both the updater network and the representor network can be end-to-end trained by minimizing the model objective in Eqn. (8).

3.4 Model Learning

In this subsection, we present a stage-wise learning scheme to learn the model parameters $\mathbf{W}_F$ and $\Theta = \{\Theta^{(k)}\}_{k=1,2,\cdots,K}$. After the first $(k'-1)$ stages of learning, we can obtain the current model parameters $\mathbf{W}_F$ and $\{\Theta^{(k)}\}_{k=1,2,\cdots,(k'-1)}$. Denote by $\Theta^{(k')} = \{\lambda^{(k')}, \mathbf{M}^{(k')}, \rho^{(k')}, \eta^{(k')}\}$. To guide the model learning, we define the stage-wise loss function as,

$$J_{k'} = \left\| \mathbf{y}_{t+1} - \sum_{l=1}^{L} \mathbf{z}_{t+1,l} \star \mathbf{f}_{t+1,l}^{(k')} \right\|^2 . \tag{11}$$

Then we introduce the gradient computation which is used to update model parameters with the stochastic gradient descent (SGD) algorithm.

According to Eqns. (9)$\sim$(10c), we have the following observations:

(a) $\mathbf{f}_{t+1}^{(k')}$ is a function of $\mathbf{f}^{(k')}$, $\mathbf{f}_t$ and $\eta^{(k')}$;
(b) $\mathbf{h}^{(k')}$ is a function of $\mathbf{f}^{(k'-1)}$, $\mathbf{g}^{(k'-1)}$, $\lambda^{(k')}$, $\rho^{(k')}$ and $\mathbf{M}^{(k')}$;
(c) $\mathbf{g}^{(k')}$ is a function of $\mathbf{g}^{(k'-1)}$, $\mathbf{f}^{(k'-1)}$, $\mathbf{h}^{(k')}$ and $\mathbf{M}^{(k')}$;
(d) $\mathbf{f}^{(k')}$ is a function of $\mathbf{z}_t$, $\mathbf{y}_t$, $\mathbf{h}^{(k')}$, $\mathbf{g}^{(k')}$, $\rho^{(k')}$ and $\mathbf{M}^{(k')}$.

Combined these observations with Eqn. (11), we can obtain the gradient of $J_{k'}$ w.r.t. $\Theta^{(k')}$ in the k'-th stage, i.e., $\nabla_{\Theta^{(k')}} J_{k'} = \left(\nabla_{\eta^{(k')}} J_{k'}, \nabla_{\rho^{(k')}} J_{k'}, \nabla_{\mathbf{M}^{(k')}} J_{k'}, \nabla_{\lambda^{(k')}} J_{k'}\right)$.

Specifically, for each parameter in $\Theta^{(k')}$, we have,

$$\begin{cases} \nabla_{\eta^{(k')}} J_{k'} = \nabla_{\mathbf{f}_{t+1}^{(k')}} J_{k'} \nabla_{\eta^{(k')}} \mathbf{f}_{t+1}^{(k')} \\ \nabla_{\rho^{(k')}} J_{k'} = \nabla_{\mathbf{f}^{(k')}} J_{k'} \nabla_{\rho^{(k')}} \mathbf{f}^{(k')} + \nabla_{\mathbf{h}^{(k')}} J_{k'} \nabla_{\rho^{(k')}} \mathbf{h}^{(k')} \\ \nabla_{\mathbf{M}^{(k')}} J_{k'} = \nabla_{\mathbf{f}^{(k')}} J_{k'} \nabla_{\mathbf{M}^{(k')}} \mathbf{f}^{(k')} + \nabla_{\mathbf{g}^{(k')}} J_{k'} \nabla_{\mathbf{M}^{(k')}} \mathbf{g}^{(k')} + \nabla_{\mathbf{h}^{(k')}} J_{k'} \nabla_{\mathbf{M}^{(k')}} \mathbf{h}^{(k')} \\ \nabla_{\lambda^{(k')}} J_{k'} = \nabla_{\mathbf{h}_{t+1}^{(k')}} J_{k'} \cdot \nabla_{\lambda^{(k')}} \mathbf{h}_{t+1}^{(k')} \end{cases} \tag{12}$$

The derivations of $\nabla_{\mathbf{f}^{(k')}} J_{k'}$, $\nabla_{\mathbf{g}^{(k')}} J_{k'}$ and $\nabla_{\mathbf{h}^{(k')}} J_{k'}$ are presented in the supplementary materials.

Furthermore, $J_{k'}$ should also be used to update the model parameters $\mathbf{W}_F$ and $\{\Theta^{(k)}\}_{k=1,2,\cdots,(k'-1)}$ for the sake of joint representation and truncated inference learning. Thus, we also give the gradient of $J_{k'}$ w.r.t. $\mathbf{h}^{(k'-1)}$, $\mathbf{g}^{(k'-1)}$, and $\mathbf{f}^{(k'-1)}$ as follows,

$$\begin{cases} \nabla_{\mathbf{h}^{(k'-1)}} J_{k'} = \nabla_{\mathbf{g}^{(k'-1)}} J_{k'} \nabla_{\mathbf{h}^{(k'-1)}} \mathbf{g}^{(k'-1)} + \nabla_{\mathbf{f}^{(k'-1)}} J_{k'} \nabla_{\mathbf{g}^{(k'-1)}} \mathbf{f}^{(k'-1)} \\ \nabla_{\mathbf{g}^{(k'-1)}} J_{k'} = \nabla_{\mathbf{g}^{(k')}} J_{k'} \nabla_{\mathbf{g}^{(k'-1)}} \mathbf{g}^{(k')} + \nabla_{\mathbf{h}^{(k')}} J_{k'} \nabla_{\mathbf{g}^{(k'-1)}} \mathbf{h}^{(k')} \\ \nabla_{\mathbf{f}^{(k'-1)}} J_{k'} = \nabla_{\mathbf{g}^{(k')}} J_{k'} \nabla_{\mathbf{f}^{(k'-1)}} \mathbf{g}^{(k')} + \nabla_{\mathbf{h}^{(k')}} J_{k'} \nabla_{\mathbf{f}^{(k'-1)}} \mathbf{h}^{(k')} \end{cases} \tag{13}$$

Please refer to the supplementary material for the detail of the derivation. Therefore, we can back-propagate the gradient to the $(k'-1), ..., 1$ layers and the representor network $\psi(\cdot; \mathbf{W}_F)$. After the learning of the k'-th stage, we can further

conduct the $(k'+1)$-th stage-wise training by learning $\Theta^{(k'+1)}$ and fine-tuning $\mathbf{W}_F$ and $\{\Theta^{(k)}\}_{k=1,2,\cdots,k'}$ until the ending of the K-th stage-wise training. Finally, all the model parameters $\mathbf{W}_F$ and Θ are adopted for target localization and model adaptation during the on-line tracking process.

4 Experiments

In this section, we first describe the implementation details, then compare with the baseline trackers highly relevant to our approach. For comprehensive analysis, ablation studies are conducted to investigate the effect of the joint feature representation learning and stage-wise training scheme. Finally, we compare the proposed RTINet with state-of-the-art trackers on the OTB-2015 [31], TB-50 [31] (i.e., the 50 more challenging sequences from OTB-2015), TempleColor-128 [21] and VOT2016 [18] datasets. Our approach is implemented in MATLAB 2017a using MatConvNet library, and all the experiments are run on a PC equipped with an Intel i7 CPU 4.0GHz, 32GB and a single NVIDIA GTX 1080 GPU.

4.1 Implementation Details

Training Set. To train the RTINet, we employ the 2015 edition of ImageNet Large Scale Visual Recognition Challenge (ILSVRC2015) dataset, which consists of more than 4,500 videos from 30 different object categories. For each video, we pick up 20 successive frames in which the target sizes are not larger than 50% of the image size. Then, 2,000 sequences are randomly chosen for training and the rest are used as the validation set. To avoid the influence of target distortion, we crop the square region centered at the target with the size of $5\sqrt{WH} \times 5\sqrt{WH}$, where W and H represent the width and height of the target, respectively. And the cropped regions are further resized to 224×224 as the input of the RTINet.

Training Details. Since it is not trivial to train the RTINet with all the parameters directly, we decouple the training of the representor network and updater network into two steps: (1) We firstly keep the representor network fixed and train the updater network in a greedily stage-wise manner. As for the stage k, we initialize the hyper-parameters of the updater network (i.e., $\lambda^{(k)}$, $\rho^{(k)}$, $\eta^{(k)}$ and $\mathbf{M}^{(k)}$) with the trained parameters in the previous stage $k-1$. Then the updater network is trained with 50 epochs with all the parameters in the previous stages fixed. (2) After the stage-wise training of the updater network, we apply another 50 epochs to jointly train the representor network and updater network.

During training, we initialize the convolution layers of the representor network with the pre-trained VGG-M model [3]. As for the model parameters, we set $\lambda^{(0)}$, $\rho^{(0)}$, $\eta^{(0)}$ and $\mathbf{M}^{(0)}$ in the first stage of the updater network as 1, 1, 0.013 and the binary selection matrix, respectively. We use the stochastic gradient descent (SGD) as the optimizer with the mini-batch size of 16, and the learning rate is exponentially decayed from 10^{-2} to 10^{-5}.

Table 1. Comparison with the baseline CFNet variants on OTB-2015.

Trackers	CFNet-conv1	CFNet	CFNet-conv1-Rep	CFNet-Rep	RTINet-conv1	RTINet
AUC	53.6	56.8	54.8	58.0	64.3	68.2
FPS	84	75	82.7	68	23.3	9.0

Table 2. Comparison with the baseline BACF variants on OTB-2015.

Trackers	BACF	BACF-VGGM	BACF-Rep	RTINet-VGGM	stdBACF-Rep	RTINet
AUC	61.5	63.1	64.0	66.5	64.2	68.2
FPS	35.3	6.1	6.5	8.9	7.0	9.0

4.2 Comparison with CFNet

The most relevant methods to our RTINet is CFNet [30], which is also proposed for the joint learning of deep representation and CF tracker. In comparison, the updater network of our RTINet is designed based on the unrolled optimization of BACF [17]. Here, we evaluate two variants of the proposed method: RTINet with three convolution layers and its rapid version, i.e., RTINet-conv1 with one convolution layer, and compare them with CFNet, CFNet-conv1, and their two variants with features extracted by RTINet representor, i.e., CFNet-conv1-Rep and CFNet-Rep on OTB-2015. Following the protocols in [31], we report the results in terms of area under curve (AUC) and tracking speed in Table 1. And we have two observations. (1) The CFNet variants with RTINet features perform better than CFNet-conv1 and CFNet with an AUC gain of 1.2% and 1.2%, respectively, thereby showing the effectiveness and generalization of the deep features learned by RTINet. (2) In terms of AUC, both RTINet variants perform favorably against their counterparts, indicating that RTINet is effective in learning feature representation and truncated inference. In particular, RTINet brings an AUC gain of 11.4% over CFNet on the OTB-2015 dataset. As for the rapid version, RTINet-conv1 also outperforms its baseline CFNet-conv1 by a gain of 10.7%. RTINet even achieves an AUC of 68.2% on OTB-2015, outperforming other trackers with a large margin. We owe the improvements to both the introduction of the advanced BACF tracker and truncated inference into the RTINet framework.

We also report the average FPS of different trackers. While the best speed belongs to the CFNet-conv1 (84 fps) and CFNet-conv1-Rep (82.7 fps), RTINet runs at 9 fps and achieves the state-of-the-art tracking accuracy. Actually, a large part of computational cost in RTINet comes from the deeper CNN feature extraction. When conv1 feature is adopted, and RTINet-conv1 achieves a real time speed of 24 fps while still performing favorably against CFNet.

Table 3. The AUC scores of RTINet by training with different number of stages.

Number of Stages	1	2	3	4	5	6	7	8	9	10
Basketball	62.0	75.9	69.1	64.3	69.4	69.1	68.9	68.9	68.8	68.8
BlurCar1	77.1	83.0	81.2	81.1	80.6	80.7	80.5	80.4	80.3	80.3
CarDark	76.2	85.7	83.3	82.9	82.2	82.1	81.6	81.7	82.2	82.3
Human4	44.1	57.0	55.6	57.7	61.5	51.0	52.2	51.5	52.0	52.3
Toy	60.1	61.1	63.1	62.8	62.1	61.9	62.8	62.8	62.7	63.0
OTB-2015	59.6	68.2	67.2	67.2	66.3	66.0	65.6	66.3	66.0	66.2

4.3 Ablation studies

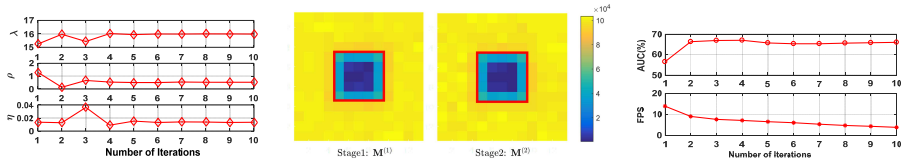
In this section, we analyze in depth the effect of joint feature representation and truncated inference learning as well as stage-wise training.

Joint Learning. To investigate the effect of joint learning, we decouple the feature representation and truncated inference learning, which results in four variants of RTINet: BACF-VGGM (BACF with the fixed convolutional feature from pre-trained VGG-M), BACF-Rep (BACF with the learned RTINet representation), RTINet-VGGM (RTINet with the fixed convolution feature from pre-trained VGG-M) and the full RTINet model. Besides, we also apply the learned RTINet representation and model parameters λ, η and $\mathbf{M}$ to the standard BACF, resulting in stdBACF-Rep. Table 2 shows the AUC scores of the default BACF with HOG features, and the BACF variants on OTB-2015.

From Table 2, it can be seen that RTINet and RTINet-VGGM improve the AUC scores significantly in comparison with the corresponding BACF variants. This can be attributed to that the truncated inference learning in updater network does benefit the tracking performance. Moreover, RTINet also improves the performance of RTINet-VGGM by an AUC gain of 1.7%, and BACF-Rep obtains a gain of 0.9% over BACF-VGGM, validating the effectiveness of representation learning. It is worth noting that, in our RTINet the inference learning improves the performance more than the feature learning, implying that pre-trained VGG-M does have good representation and generalization ability. To sum up, both the learned feature representation and truncated inference are helpful in improving tracking accuracy, which together explain the favorable performance of our RTINet.

Stage-wise Learning. In Section 3, we present a stage-wise training scheme to learn model parameters. In particular, we solve the BACF [17] formulation using the truncated ADMM optimization. Thus, we analyse the effect of stage number on tracking performance. Table 3 gives the average AUC score of RTINet on all sequences as well as several representative ones by setting different number of stages on the OTB-2015 dataset. RTINet with one stage performs poorly with the AUC of 59.6%, even lower than the BACF (61.5%). This is reasonable due to that RTINet only with one stage is similar to the simple CF rather than the advanced BACF model. Benefited from the advanced BACF, RTINet achieves

significantly better performance within 2∼5 iterations for most sequences. The best AUC score of 68.2% of RTINet is attained with two stages on OTB-2015, indicating that efficient solver can be learned. It can also be found that increasing number of stages causes moderate decrease on AUC. One possible reason is that for smaller number of stages, RTINet focuses on minimizing upper loss in Eqn. (7) and benefits accuracy. For larger number of stages, RTINet may begin to minimize lower loss in Eqn. (7) instead of accuracy.

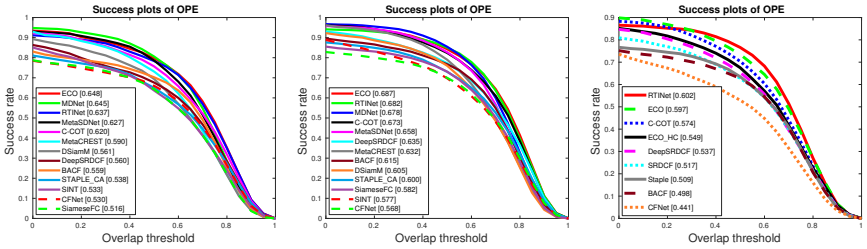


(a) (b) (c)

Figure 2. (a) The learned λ, ρ, η for each stage. (b) Visualization of $\mathbf{M}$ for the first two stages. (c) Evaluation on the number of stages used for testing with an off-line trained 10-stage RTINet.

Visualization of Learned Parameters. Parameters at all stages are off-line trained and then keep fixed during tracking. Fig. 2(a) shows the plots of the learned stage-wise λ, ρ, η used in Table 3. It can be noted that the values of λ, ρ, η become stable from the fourth stage. From Table 3, the best tracking accuracy is attained when the stage number is two. Thus, we present the visualization of the learned $\mathbf{M}$s for the first two stages in Fig. 2(b). From Fig. 2(a)(b), we have two observations: (1) each stage has its specific parameter values, (2) the learned $\mathbf{M}$s relax the binary cropping operation which is slightly different with the $\mathbf{M}$ adopted in BACF. We also note that both the $\mathbf{M}$ in BACF and our learned $\mathbf{M}$s are resized to the feature map size in tracking.

Effects of Convergence on Tracking. Generally, the ADMM algorithms are adopted to resolve the constrained convex optimization problem with a guarantee of convergence. Thus, it is interesting to discuss the effect of iteration numbers after training RTINet with a fixed number of stages. To this end, we train a 10-stage RTINet and test it on the OTB-2015 by using different number of iterations in tracking. From Fig. 2(c), the best tracking accuracy is obtained after 4 iterations. Then RTINet may focus on minimizing the lower loss and more iterations does not bring any increase on accuracy. Fig. 2(c) also shows the plot of tracking speed. Comparing Table 3 and Fig. 2(c), it can be seen that direct training RTINet with small K is better than first training a 10-stage RTINet and then testing it with small iterations.



(a) TB-50 (b) OTB-2015 (c) TempleColor-128

Figure 3. Overlap success plots of different trackers on the TB-50, OTB-2015 and TempleColor-128 datasets.

4.4 Comparison with the state-of-the-art methods

We compare RTINet with several state-of-the-art trackers, including CF-based trackers (i.e., ECO [6], C-COT [10], DeepSRDCF [7], BACF [17], STAPLE-CA [23]) and learning-based CNN trackers (i.e., MDNet [24], MetaSDNet [25], MetaCREST [25], SiameseFC [1], DSiamM [13] and SINT [29]). Note that all the results are obtained by using either the publicly available codes or the results provided by the authors for fair comparison. Experiments are conducted on TB-50 [31], OTB-2015 [31], TempleColor-128 [21] and VOT-2016 [18]. On the first three datasets, we follow the OPE protocol provided in [31] and present the success plots ranked by the AUC scores. On VOT-2016, we evaluate the trackers in terms of accuracy, robustness and expected average overlap (EAO).

OTB-2015 and TB-50. Fig. 3(a)(b) shows the success plots of the competing trackers the OTB-2015 and TB-50 benchmarks. And the proposed RTINet is ranked in top-3 on the two datasets, achieves comparable performance with the top trackers such as ECO and MDNet [24]. Moreover, RTINet obtains an AUC score of 68.2% on OTB-2015, outperforming its counterparts CFNet and BACF by a margin of 11.4% and 6.7%, respectively. In Fig. 3, we also compare RTINet with the recently proposed Meta-Trackers [25] (i.e., MetaSDNet and MetaCREST). Again our RTINet performs better than both MetaSDNet and MetaCREST by the AUC score. And even the rapid version RTINet-conv1 outperforms MetaCREST, and is comparable to MetaSDNet. On the more challenging sequences in TB-50, our RTINet is still on par with the state-of-the-art ECO and ranks the second among the competing trackers. Specifically, RTINet performs better than the other learning-based trackers, including SiameseFC [1], DSiamM [13] and SINT [29], and surpasses its baseline CFNet [30] by 10.7%. In comparison to CFNet and BACF, the superiority of RTINet can be ascribed to the incorporation of the advanced BACF model, and the joint learning of deep representation and truncated inference. Finally, we analyze the performance with respect to attributes. RTINet performs in top-3 on 6 of the 11 attributes and is on par with the state-of-the-arts on the other attributes. Detailed results are given

Table 4. Comparison with the state-of-the-art trackers in terms of EAO, Robustness, and Accuracy on VOT-2016 dataset.

Trackers	ECO	C-COT	DeepSRDCF	SRDCF	HCFT	Staple	BACF	RTINet
EAO	0.374	0.331	0.276	0.247	0.220	0.295	0.233	0.298
Accuracy	0.54	0.52	0.51	0.52	0.47	0.54	0.56	0.57
Robustness	0.72	0.85	1.17	1.50	1.38	1.35	1.88	1.07

in the supplementary materials. The results further validates the effectiveness of our proposed RTINet.

TempleColor-128. Fig. 3(c) shows the success plots on TempleColor-128. RTINet performs favorably against ECO with an AUC score of 60.2%, and achieves significant improvements over BACF and C-COT, by a gain of 10.4% and 2.8%, respectively. In particular, compared with its counterpart CFNet, RTINet improves the performance with a large margin of 16.1%. The results further demonstrate the effectiveness of joint representation and truncated inference learning.

VOT2016. Quantitative results on VOT2016 are also be presented in terms of accuracy, robustness and EAO in Table 4. RTINet achieves promising performance and performs much better than the BACF, SRDCF and DeepSRDCF both in terms of accuracy and robustness. In particular, it obtains the best result on accuracy with a value of 0.57, and performs the third-best on robustness and EAO. It is worth noting that, RTINet performs favorably to ECO by accuracy but is inferior by robustness, which may be ascribed to that only the accuracy is considered in the training loss in Eqn. (8) of RTINet.

5 Conclusion

This paper presents a RTINet framework for joint learning of deep representation and model adaptation in visual tracking. We adopt the deep convolutional network for feature representation and integrate the CNN with advanced BACF tracker. To solve the BACF in the CNN architecture, we design the model adaptation network as truncated inference by unrolling the ADMM optimization of the BACF model. Moreover, a greedily stage-wise learning scheme is introduced for the joint learning of deep representation and truncated inference from the annotated video sequences. Experimental results on three tracking benchmarks show that our RTINet tracker achieves favorable performance in comparison with the state-of-the-art trackers. Besides, our rapid version of RTINet can run in real-time (24 fps) at a moderate sacrifice of accuracy. By taken BACF as an example, our RTINet sheds some light on incorporating the advances in CF modeling for improving the performance of learning-based trackers, and thus deserves in-depth investigation in future work.

Acknowledgement. This work was supported in part by the National Natural Science Foundation of China under Grant No.s 61671182 and 61471146.

References

1. Bertinetto, L., Valmadre, J., Henriques, J.F., Vedaldi, A., Torr, P.H.: Fully-convolutional siamese networks for object tracking. In: ECCV. pp. 850–865 (2016)
2. Bibi, A., Mueller, M., Ghanem, B.: Target response adaptation for correlation filter tracking. In: ECCV. pp. 419–433 (2016)
3. Chatfield, K., Simonyan, K., Vedaldi, A., Zisserman, A.: Return of the devil in the details: Delving deep into convolutional nets. In: BMVC (2014)
4. Chen, K., Tao, W.: Once for all: a two-flow convolutional neural network for visual tracking. TCSVT **PP**, 1–1 (2017)
5. Choi, J., Kwon, J., Lee, K.M.: Deep meta learning for real-time visual tracking based on target-specific feature space. arXiv:1712.09153 (2017)
6. Danelljan, M., Bhat, G., Khan, F.S., Felsberg, M.: ECO: efficient convolution operators for tracking. In: CVPR. pp. 21–26 (2017)
7. Danelljan, M., Hager, G., Shahbaz Khan, F., Felsberg, M.: Convolutional features for correlation filter based visual tracking. In: ICCV Workshop. pp. 58–66 (2015)
8. Danelljan, M., Hager, G., Shahbaz Khan, F., Felsberg, M.: Learning spatially regularized correlation filters for visual tracking. In: ICCV. pp. 4310–4318 (2015)
9. Danelljan, M., Hager, G., Shahbaz Khan, F., Felsberg, M.: Adaptive decontamination of the training set: A unified formulation for discriminative visual tracking. In: CVPR. pp. 1430–1438 (2016)
10. Danelljan, M., Robinson, A., Khan, F.S., Felsberg, M.: Beyond correlation filters: Learning continuous convolution operators for visual tracking. In: ECCV. pp. 472–488 (2016)
11. Danelljan, M., Shahbaz Khan, F., Felsberg, M., Van de Weijer, J.: Adaptive color attributes for real-time visual tracking. In: CVPR. pp. 1090–1097 (2014)
12. Dong, C., Loy, C.C., He, K., Tang, X.: Learning a deep convolutional network for image super-resolution. In: ECCV. pp. 184–199 (2014)
13. Guo, Q., Feng, W., Zhou, C., Huang, R., Wan, L., Wang, S.: Learning dynamic siamese network for visual object tracking. In: ICCV. pp. 1–9 (2017)
14. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: CVPR. pp. 770–778 (2016)
15. Held, D., Thrun, S., Savarese, S.: Learning to track at 100 fps with deep regression networks. In: ECCV. pp. 749–765 (2016)
16. Henriques, J.F., Caseiro, R., Martins, P., Batista, J.: High-speed tracking with kernelized correlation filters. TPAMI **37**(3), 583–596 (2015)
17. Kiani Galoogahi, H., Fagg, A., Lucey, S.: Learning background-aware correlation filters for visual tracking. In: CVPR. pp. 1135–1143 (2017)
18. Kristan, M., Leonardis, A., Matas, J., Felsberg, M., Pflugfelder, R., Čehovin, L., Vojir, T., Häger, G., Lukežič, A., Fernandez, G.: The Visual Object Tracking VOT2016 Challenge Results (Oct 2016), `http://www.springer.com/gp/book/9783319488806`
19. Krizhevsky, A., Sutskever, I., Hinton, G.E.: ImageNet classification with deep convolutional neural networks. In: NIPS. pp. 1097–1105 (2012)
20. Li, F., Tian, C., Zuo, W., Zhang, L., Yang, M.H.: Learning spatial-temporal regularized correlation filters for visual tracking. In: CVPR (2018)
21. Liang, P., Blasch, E., Ling, H.: Encoding color information for visual tracking: Algorithms and benchmark. TIP **24**(12), 5630–5644 (2015)

22. Ma, C., Huang, J.B., Yang, X., Yang, M.H.: Hierarchical convolutional features for visual tracking. In: ICCV. pp. 3074–3082 (2015)
23. Mueller, M., Smith, N., Ghanem, B.: Context-aware correlation filter tracking. In: CVPR. pp. 1396–1404 (2017)
24. Nam, H., Han, B.: Learning multi-domain convolutional neural networks for visual tracking. In: CVPR. pp. 4293–4302 (2015)
25. Park, E., Berg, A.C.: Meta-Tracker: Fast and robust online adaptation for visual object trackers. arXiv:1801.03049 (2018)
26. Qi, Y., Zhang, S., Qin, L., Yao, H., Huang, Q., Lim, J., Yang, M.H.: Hedged deep tracking. In: CVPR. pp. 4303–4311 (2016)
27. Ren, S., He, K., Girshick, R., Sun, J.: Faster R-CNN: Towards real-time object detection with region proposal networks. In: NIPS. pp. 91–99 (2015)
28. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. arXiv:1409.1556 (2014)
29. Tao, R., Gavves, E., Smeulders, A.W.: Siamese instance search for tracking. In: CVPR. pp. 1420–1429 (2016)
30. Valmadre, J., Bertinetto, L., Henriques, J., Vedaldi, A., Torr, P.H.: End-to-End representation learning for correlation filter based tracking. In: CVPR. pp. 5000–5008 (2017)
31. Wu, Y., Lim, J., Yang, M.H.: Object Tracking Benchmark. TPAMI **37**(9), 1834–1848 (2015)
32. Yang, Y., Sun, J., Li, H., Xu, Z.: Deep ADMM-Net for compressive sensing MRI. In: NIPS. pp. 10–18 (2016)
33. Zuo, W., Ren, D., Gu, S., Lin, L., Zhang, L., et al.: Discriminative learning of iteration-wise priors for blind deconvolution. In: CVPR. pp. 3232–3240 (2015)
34. Zuo, W., Wu, X., Lin, L., Zhang, L., Yang, M.H.: Learning support correlation filters for visual tracking. TPAMI (2018)