Meta-Tracker: Fast and Robust Online Adaptation for Visual Object Trackers

Eunbyung Park Alexander C. Berg

Department of Computer Science,
University of North Carolina at Chapel Hill
{eunbyung,aberg}@cs.unc.edu

Abstract. This paper improves state-of-the-art visual object trackers that use online adaptation. Our core contribution is an offline meta-learning-based method to adjust the initial deep networks used in online adaptation-based tracking. The meta learning is driven by the goal of deep networks that can quickly be adapted to robustly model a particular target in future frames. Ideally the resulting models focus on features that are useful for future frames, and avoid overfitting to background clutter, small parts of the target, or noise. By enforcing a small number of update iterations during meta-learning, the resulting networks train significantly faster. We demonstrate this approach on top of the high performance tracking approaches: tracking-by-detection based MDNet [1] and the correlation based CREST [2]. Experimental results on standard benchmarks, OTB2015 [3] and VOT2016 [4], show that our meta-learned versions of both trackers improve speed, accuracy, and robustness.

1 Introduction

Visual object tracking is a task that locates target objects precisely over a sequence of image frames given a target bounding box at the initial frame. In contrast to other object recognition tasks, such as object category classification and detection, in visual object tracking, instance-level discrimination is an important factor. For example, a target of interest could be one particular person in a crowd, or a specific product (e.g. coke can) in a broader category (e.g. soda cans). Therefore, an accurate object tracker should be capable of not only recognizing generic objects from background clutter and other categories of objects, but also discriminating a particular target among similar distractors that may be of the same category. Furthermore, the model learned during tracking should be flexible to account for appearance variations of the target due to viewpoint change, occlusion, and deformation.

One approach to these challenges is applying online adaptation. The model of the target during tracking, e.g. DCF (discriminative correlation filter) or binary classifier (the object vs backgrounds), is initialized at the first frame of a sequence, and then updated to be adapted to target appearance in subsequent

The code is available at https://github.com/silverbottlep/meta_trackers.

frames [1,2,5,6,7,8,9,10]. With the emergence of powerful generic deep-learning representations, recent top performing trackers now leverage the best of both worlds: deep learned features and online adaptation methods. Offline-only trackers trained with deep methods have also been suggested, with promising results and high speed, but with a decrease in accuracy compared to state-of-the-art online adaptive trackers [11,12,13], perhaps due to difficulty finely discriminating specific instances in videos.

A common practice to combine deep learning features and online adaptation is to train a target model on top of deeply learned features, pre-trained over a large-scale dataset. These pre-trained features have proven to be a powerful and broad representation that can recognize many generic objects, enabling effective training of target models to focus on the specified target instance. Although this type of approach has shown the best results so far, there remain several important issues to be resolved.

First, very few training examples are available. We are given a single bounding box for the target in the initial frame. In subsequent frames, trackers collect additional images, but many are redundant since they are essentially the same target and background. Furthermore, recent trends towards building deep models for target appearance [1,2] make the problem more challenging since deep models are known to be vulnerable to overfitting on small datasets. As a consequence, a target model trained on top of deeply learned features sometimes suffers because it overfits to background clutter, small parts or features of the target, or noise. Many recent studies have proposed various methods to resolve these issues. Some include using a large number of positive and negative samples with aggressive regularizers [1], factorized convolution [6], spatio-residual modules [2], or incorporating contextual information [14].

Second, most state-of-the-art trackers spend a significant amount of time on the initial training stage [1,2,6]. Although many works have proposed fast training methods [6,7], this still remains a bottleneck. In many practical applications of object tracking, such as surveillance, real-time processing is required. Depending on the application, falling behind on the initial frame could mean failure on the whole task. On the other hand, an incompletely trained initial target model could affect performance on future frames, or in the worst case, result in failures on all subsequent frames. Therefore, it is highly desirable to obtain the robust target model very quickly at the initial frame.

In this work, we propose a generic and principled way of tackling these challenges. Inspired by recent meta-learning (learning to learn) studies [15,16,17,18,19,20], we seek to learn how to obtain the target model. The key idea is to train the target model in a way that generalizes well over future frames. In all previous works [1,2,5,6,7,8,9,10], the target model is trained to minimize a loss function on the current frame. Even if the model reaches an optimal solution, it does not necessarily mean it would work well for future frames. Instead, we suggest to use error signals from future frames. During the meta-training phase, we aim to find a generic initial representation and gradient directions that enable the target model to focus on features that are useful for future frames. Also, this

meta-training phase helps to avoid overfitting to distractors in the current frame. In addition, by enforcing the number of update iterations during meta-training, the resulting networks train significantly faster during the initialization.

Our proposed approach can be applied to any learning based tracker with minor modifications. We select two *state-of-the-art trackers*, MDNet [1], from the classifier based tracker (tracking-by-detection) category, and CREST [2], a correlation based tracker. Experimental results show that our meta-learned version of these trackers can adapt very quickly—just one iteration—for the first frame while improving accuracy and robustness. Note that this is done even without employing some of the hand engineered training techniques, sophisticated architectural design, and hyperparameter choices of the original trackers. In short, we present an easy way to make very good trackers even better without too much effort, and demonstrate its success on two different tracking architectures, indicating potentially general applicability.

2 Related Work

Online trackers: Many online trackers use correlation filters as the back-bone of the algorithms due to computational efficiency and discriminative power. From the early success of the MOSSE tracker [10], a large number of variations have been suggested. [7] makes it more efficient by taking advantage of circulant matrices, further improved by resolving artificial boundary issues [21,22]. Many hard cases have been tackled by using context information [14,23], short and long-term memory [24,25], and scale-estimation [26], just to name a few. Recently, deep learning features have begun to play an important role in correlation filters [1,2,5,6,8,27,28]. On the other hand, tracking-by-detection approaches typically learn a classifier to pick up the positive image patches wrapping around the target object. Pioneered by [9], many learning techniques have been suggested, e.g. multiple instance learning [29], structured output SVMs [30], online boosting [31], and model ensembles [32]. More recently, MDNet [1], with deep features and a deep classifier, achieved significantly higher accuracy.

Offline trackers: Several recent studies have shown that we can build accurate trackers without online adaptation [11,12,13] due to powerful deep learning features. Siamese-style networks take a small target image patch and a large search image patch, and directly regress the target location [12] or generate a response map [11] via a correlation layer [33]. In order to consider temporal information, recurrent networks have also been explored in [34,35,36,37].

Meta-learning: This is an emerging field in machine learning and its applications. Although it is not a new concept [38,39,40,41], many recent works have shown very promising results along with deep learning success. [17,42,43,44] attempted to replace hand-crafted optimization algorithms with meta-learned deep networks. [16] took this idea into few shot or one shot learning problem. It aimed to learn optimal update strategies based on how accurate a learner can classify test images with few training examples when the learner follows the strategies from the meta-learner. Instead of removing existing optimization algorithms, [15] focuses on learning initialization that are most suitable for existing algo-

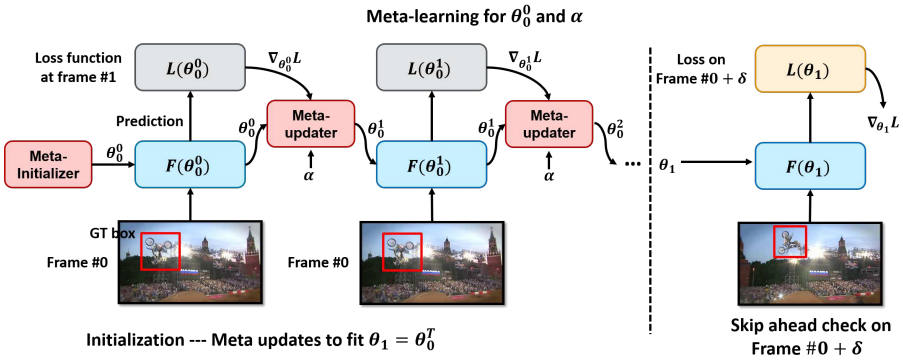


Fig. 1: Our meta-training approach for visual object tracking: A computational graph for meta-training object trackers. For each iteration, it gets the gradient with respect to the loss after the first frame, and a meta-updater updates parameters of the tracker using those gradients. For added stability and robustness a final loss is computed using a future frame to compute the gradients w.r.t parameters of meta-initializer and meta-updater. More details in Section 3.

rithms. [19] further learns parameters of existing optimization algorithms along with the initialization. Unlike approaches introduced above, there also have been several studies to directly predict the model parameters without going through the optimization process [37,45,46].

3 Meta-Learning for Visual Object Trackers

In this section, we explain the proposed generalizable meta-training framework for visual object trackers. The details for applying this to each tracker are found in Section 4.

3.1 Motivation

A typical tracking episode is as follows: The tracking model is adapted to a specified bounding box around the target in the initial frame of a sequence. Aggressive regularizers and fast optimization techniques are adopted to allow this adaptation/training to be done quickly so that the resulting model is robust to target variations and environment changes. Then, the tracking model is used to predict the target location in subsequent frames. Predicted target locations and images are then stored in the database, and the models are regularly updated with collected data according to their own strategies.

A key motivation is to incorporate these actual tracking scenarios into the meta-learning process. The eventual goal of trackers is to predict the target locations in future frames. Thus, it would be desirable to learn trackers with this eventual goal. For example, if we could look at variations in future frames, then we could build more robust target models and prevent them from overfitting to the current target appearance or background clutter. We can take a step back and observe trackers running on videos, see if the trackers generalize well, and

Algorithm 1 Meta-training object trackers algorithm

Input : Randomly initialized θ_0 and α, training dataset D
Output : θ_0^* and α^*

while not converged **do**
  $\text{grad}_{\theta_0}, \text{grad}_{\alpha} = \mathbf{0}$ ▷ Initialize to zero vector
  for all $k \in \{0, \ldots, N_{\text{mini}} - 1\}$ **do**
    $S, j, \delta \sim p(D)$ ▷ Sample a training example
    $\theta_0^0 = \theta_0$
    for all $t \in \{0, \ldots, T-1\}$ **do**
      $\hat{y}_j = F(x_j, \theta_0^t)$
      $\theta_0^{t+1} = \theta_0^t - \alpha \odot \nabla_{\theta_0^t} L(y_j, \hat{y}_j; \theta_0^t)$
    end for
    $\theta_1 = \theta_0^T$
    $\hat{y}_{j+\delta} = F(x_{j+\delta}, \theta_1)$ ▷ Apply to a future frame
    $\text{grad}_{\theta_0} = \text{grad}_{\theta_0} + \nabla_{\theta_0} L(y_{j+\delta}, \hat{y}_{j+\delta})$ ▷ Accumulate the gradients
    $\text{grad}_{\alpha} = \text{grad}_{\alpha} + \nabla_{\alpha} L(y_{j+\delta}, \hat{y}_{j+\delta})$
  end for
  $\theta_0 = \text{Optimizer}(\theta_0, \text{grad}_{\theta_0})$ ▷ Update θ_0
  $\alpha = \text{Optimizer}(\alpha, \text{grad}_{\alpha})$ ▷ Update α
end while

find a reason why they become distracted and adjust the adaptation procedure accordingly.

3.2 A general online tracker

This formulation of online tracking is made general in order to apply to a variety of trackers. Consider the key operation in a tracker, $\hat{y} = F(x, \theta)$, that takes an input x, e.g. image patches around the target or a cropped image centered on putative target from an image I, and the tracker parameters θ and produces an estimate $\hat{y}$ of the label, e.g. a response map or a location in the frame that indicates the target position. For *initialization*, x_0 from the initial frame I_0 with specified y_0, we (approximately) solve for $\theta_1(x_0, y_0)$, or θ_1 for brevity, with respect to a loss, $L\left(F\left(x_0, \theta_1\right), y_0\right)$, measuring how well the model predicts the specified label. For *updates* during tracking, we take the parameters θ_j from frame $j-1$ and find $\hat{y}_j = F(x_j, \theta_j)$, then find θ_{j+1} with respect to a loss. Then, we may incorporate transforming $\hat{y}_j$ into a specific estimate of the target location as well as temporal smoothing, etc. We can write the tracking process initialized with x_0 and y_0 in an initial frame and then proceeding to track and update for frames $I_1 \ldots I_n$ as $\text{Track}\left(\theta_1(x_0, y_0), I_1, \ldots, I_n\right)$ and its output as $\hat{y}_n$, an estimate of the label in the nth frame (indicating target position) and θ_{n+1}, the model parameters after the nth frame.

3.3 Meta-training algorithm

Our meta-training approach has two goals. One is that initialization for a tracker on a sequence can be performed by starting with θ_0 and applying one or a very small number of iterations of a update function M parameterized by α. Another goal is that the resulting tracker be accurate and robust on later frames.

The gradient-descent style update function M is parameterized by α:

$$M(\theta, \nabla_\theta L; \alpha) = \theta - \alpha \odot \nabla_\theta L\,, \tag{1}$$

where α is the same size as the tracker parameters θ [19], L is a loss function, and $\odot$ is element-wise product. α could be a scalar value, which might be either learnable [20] or manually fixed [15]. We empirically found that having per parameter coefficients was the most effective in our settings.

Our meta-training algorithm is to find a good θ_0 and α by repeatedly sampling a video, performing initialization, applying the learned initial model to a frame slightly ahead in the sequence, and then back-propagating to update θ_0 and α. Applying the initial model to a frame slightly ahead in the sequence has two goals, the model should be robust enough to handle more than frame-to-frame variation, and if so, this should make updates during tracking fast as well if not much needs to be fixed.

After sampling a random starting frame from a random video, we perform optimization for initialization starting with $\theta_0^0 = \theta_0$ given the transformed input and output pair, (x_j, y_j). A step of optimization proceeds as

$$\theta_0^{i+1} = M(\theta_0^i, \nabla_{\theta_0^i} L(y_j, F(x_j, \theta_0^i)))\,. \tag{2}$$

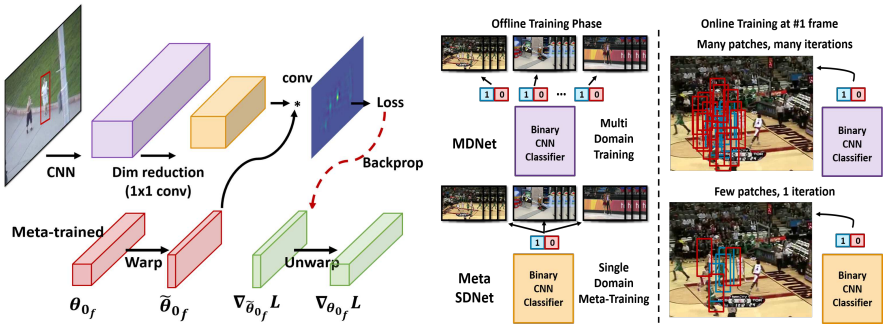
This step can be repeated up to a predefined number of times T to find, $\theta_1(x_j, y_j) = \theta_0^T$. Then, we randomly sample a future frame $I_{j+\delta}$ and evaluate the model trained on the initial frame on that future frame to produce: $\hat{y}_{j+\delta} = F(x_{j+\delta}, \theta_1)$.

The larger δ, the larger target object variations and environment changes are incorporated into training process. Now, we can compute the loss based on the future frame and trained tracker parameters. The objective function is defined as

$$\theta_0^*, \alpha^* = \underset{\theta_0, \alpha}{\operatorname{argmin}}\, \mathbb{E}_{S,j,\delta}[L(y_{j+\delta}, \hat{y}_{j+\delta})]\,. \tag{3}$$

We used the ADAM [47] gradient descent algorithm to optimize. Note that θ_0 and α are fixed across different episodes in a mini-batch, but $\theta_0^1, \ldots, \theta_0^T$ are changed over every episode. To compute gradients of the objective function w.r.t θ_0 and α, it is required to compute higher-order gradients (the gradients of function of gradients). This type of computation has been exploited in recent studies [15,48,49]. We can easily compute this thanks to automatic differentiation software libraries [50]. More details are explained in Algorithm 1.

Update rules for subsequent frames. Most online trackers, including the two trackers we meta-train (Section 4), update the target model regularly to adjust to new examples collected by itself during tracking. We could simply use meta-trained α to update the model, $\theta_j = \theta_{j-1} - \alpha \odot \nabla_{\theta_{j-1}} L$ (only one iteration presented for brevity). However, it often diverges on longer sequences or the sequences that have very small frame-to-frame variations. We believe this is mainly because we train α for fast adaptation at the initial frame, so the values of α are relatively large, which causes unstable convergence behavior (A similar phenomenon was reported in [20] albeit in a different context). Since α is stable when it teams up with θ_0, we could define the update rules for subsequent



(a) MetaCREST (b) MDNet vs MetaSDNet

frames as $\theta_j = \theta_0 - \alpha \odot \nabla_{\theta_0} L$, as suggested in [20]. We could also combine two strategies, $\theta_j = \beta(\theta_{j-1} - \alpha \odot \nabla_{\theta_{j-1}} L) + (1-\beta)(\theta_0 - \alpha \odot \nabla_{\theta_0} L)$. Although we could resolve unstable convergence behavior with these strategies, none of these performed better than simply searching for a single learning rate. Therefore, we find a learning rate for subsequent frames and then use existing optimization algorithms to update the models as was done in the original versions of the trackers.

4 Meta-Trackers

In this section, we show how our proposed meta-learning technique can be realized in state-of-the-art trackers. We selected two different types of trackers, one from correlation based trackers, CREST [2], and one from tracking-by-detection based trackers MDNet [1].

4.1 Meta-training of correlation based tracker

CREST. A typical correlation filter objective is defined as follows.

$$\underset{f}{\operatorname{argmin}} ||y - \Phi(x) * f||^2 + \lambda ||f||^2 \,, \tag{4}$$

where f is the correlation filter, $*$ is the convolution operation, and Φ is a feature extractor, e.g. CNN. x is a cropped image centered on the target, and $y \in \mathbb{R}^{H \times W}$, is a gaussian shaped response map, where H and W are height and width, respectively. The cropped image is usually larger than the target object so that it can provide enough background information. Once we have the correlation filter trained, target localization at a new future frame is simply finding the coordinates (h, w) that has the maximum response value.

$$\underset{(h,w)}{\operatorname{argmax}} \hat{y}(h, w) \,, \tag{5}$$

where $\hat{y} = \Phi(x_{\text{new}}) * f$, and $\hat{y}(h, w)$ represents the element of $\hat{y}$ in (h, w) coordinates. CREST used a variation of the correlation filter objective, defined as

$$\sum_{(h,w) \in P} \frac{1}{|P|} (e^{y(h,w)} |y(h, w) - \hat{y}(h, w)|)^2 + \lambda ||f||^2 \,, \tag{6}$$

where $P = \{(h, w) \mid |y(h, w) - \hat{y}(h, w)| > 0.1\}$. This would encourage the model to focus on parts that are far from the ground truth values.

By reformulating the correlation filter as a convolutional layer, it can be effectively integrated into an CNN framework [2]. This allows us to add new modules easily, since the optimization can be nicely done with standard gradient descent in end-to-end fashion. They inserted spatio-temporal residual modules to avoid target model degradation by large appearance changes. They also devised sophisticated initialization, learning rates, and weight decay regularizers, e.g. 1000 times larger weight decay parameter on spatio-temporal residual modules. Without those bells and whistles, we aim to learn a robust single layer correlation filter via proposed meta-learning process. There are two important issues for plugging CREST tracker into proposed meta-training framework, and we present our solutions in following sections.

Meta-learning dimensionality reduction. CREST used PCA to reduce the number of channels of extracted CNN features, from 512 to 64. This not only reduces computational cost, but also it helps to increase robustness of the correlation filter. PCA is performed at the initial frame and learned projection matrix are used for the rest of the sequence. This becomes an issue when meta-training the correlation filter. We seek to find a global initialization of the correlation filter for the all targets from different episodes. However, PCA would change the basis for every sequences, which makes impossible to obtain a global initialization in projected feature spaces that are changing every time. We propose to learn to reduce dimensions of the features. In CREST, we can insert 1x1 convolution layer right after the feature extraction, the weights of this layer are also meta-learnable and jointly trained during the meta-learning process along with the correlation filter. θ_0 in proposed meta-training framework, therefore, consists of θ_{0_d} and θ_{0_f}, the parameters of dimensionality reduction and the correlation filter, respectively.

Canonical size initialization. The size of the correlation filter varies depending on the target shape and size. In order to meta-train a fixed size initialization of the correlation filter θ_{0_f}, we should resize all objects to the same size and same aspect ratio. However, it introduces distortion of the target and has been known to degrade recognition performance [51,52]. In order to fully make use of the power of the correlation filter, we propose to use canonical size initialization and its size and aspect ratio are calculated as a mean of the objects in the training dataset. Based on canonical size initialization, we warp it to the specific size taylored to the target object for each tracking episodes, $\tilde{\theta}_{0_f} = \text{Warp}(\theta_{0_f})$. We used differentiable bilinear sampling method [53] to pass through gradients all the way down to θ_{0_f}.

Putting it all together, $F(x_j, \theta)$ in our proposed meta-training framework for CREST, now takes an input a cropped image x_j from an input frame I_j, pass it through a CNN feature extractor followed by dimensionality reduction (1x1 convolution with the weight θ_{0_d}). Then, it warps the correlation filter θ_{0_f}, and finally apply warped correlation filter $\tilde{\theta}_{0_f}$ to produce a response map $\hat{y}_j$ (Figure 2a).

4.2 Meta-training of tracking-by-detection tracker

MDNet. MDNet is based on a binary CNN classifier consisting of a few of convolutional layers and fully connected layers. In the offline phase, it uses a multi-domain training technique to pre-train the classifier. At the initial frame, it randomly initializes the last fully connected layer, and trains around 30 iterations with a large number of positive and negative patches (Figure 2b). Target locations in the subsequent frames are determined by average of bounding box regression outputs of top scoring positive patches. It collects positive and negative samples during the tracking process, and regularly updates the classifier. Multi-domain pre-training was a key factor to achieve robustness, and they used an aggressive dropout regularizer and different learning rates at different layers to further avoid overfitting to current target appearance. Without those techniques (the multi-domain training and regularizers), we aim to obtain robust and quickly adaptive classifier solely resting on the proposed meta-learning process.
Meta-training. It can be also easily pluged into the proposed meta-leraning framework. $F(x_j;\theta)$ takes as input image patches $x_j \in \mathbb{R}^{N\times D}$ from an input frame I_j (and $y_j \in \{0,1\}^N$ is the corresponding labels), where D is the size of the patches and N is the number of patches. Then, the patches go through a CNN classifier, and the loss function L is a simple cross entropy loss $-\sum_{k=1}^{N} y_j^k \log(F^k(x_j;\theta))$.
Label shuffling. Although a large-scale video detection dataset contains rich variation of objects in videos, the number of objects and categories are limited compared to other still image datasets. This might lead a deep CNN classifier to memorize all object instances in the dataset and classify newly seen objects as backgrounds. In order to avoid this issue, we adopted the label shuffling trick, suggested in [18]. Every time we run a tracking episode, we shuffle the labels, meaning sometimes labels of positive patches become 0 instead of 1, negative patches become 1 instead of 0. This trick encourages the classifier to learn how to distinguish the target objects from background by looking at current training examples, rather than memorizing specific targets appearance.

5 Experiments

5.1 Experimental setup

VOT2016. It contains 60 videos (same videos from VOT 2015 [54]). Trackers are automatically reinitialized once it drifts off the target: zero overlap between predicted bounding box and the ground truth. In this reset-based experiments, three primary measures have been used, (i) *accuracy*, (ii) *robustness* and (iii) *expected average overlap (EAO)*. The accuracy is defined as average overlap during successful tracking periods. The robustness is defined as how many times the trackers fail during tracking. The expected average overlap is an estimator of the average overlap a tracker is expected to attain on a large collection of short-term sequences.
OTB2015. It consists of 100 fully annotated video sequences. Unlike VOT2016, the one pass evaluation (OPE) is commonly used in OTB dataset (no restart at failures). The precision plots (based on the center location error) and the

success plots (based on the bounding box overlap) are used to access the tracker performance.

Dataset for meta-training. We used a large scale video detection dataset [55] for meta-training both trackers. It consists of 30 object categories, which is a subset of 200 categories in the object detection dataset. Since characteristics of the dataset are slightly different from the object tracking dataset, we sub-sampled the dataset. First, we picked a video frame that contains a target object whose size is not larger than 60% of the image size. Then, a training video sequence is constructed by sampling all subsequent frames from that frame until the size of the target object reaches 60%. We ended up having 718 video sequences. In addition, for the experiments on OTB2015 dataset, we also used an additional 58 sequences from object tracking datasets in VOT2013,VOT2014,and VOT2015 [4], excluding the videos included in OTB2015, following MDNet's approach [1]. These sequences were selected in the mini-batch selection stage with the probability 0.3. Similarly, we used 80 sequences from OTB2015, excluding the videos in VOT2016 for the experiments on VOT2016 dataset.

Baseline implementations. We selected two trackers, MDNet [1] and CREST [2]. For CREST, we re-implemented our own version in python based on publicly released code written in MATLAB. We meta-trained our version. For MDNet, the authors of MDNet provide two different source codes, written in MATLAB and python, respectively. We used the latter one and called it as pyMDNet or pySDNet, depending on pre-training methods. We meta-trained pySDNet, and call it as MetaSDNet. Note that overall accuracy of pyMDNet is lower than MDNet on OTB2015 (.652 vs .678 in success rates with overlap metric). For fair comparison, we compared our MetaSDNet to pyMDNet.

Meta-training details. In MetaSDNet, we used the first three conv layers from pre-trained vgg16 as feature extractors. During meta-training, we randomly initialized the last three fc layers, and used Adam as the optimizer with learning rate 1e-4. We only updated the last three fc layers for the first 5,000 iterations and trained all layers for the rest of iterations. The learning rate was reduced to 1e-5 after 10,000 iterations, and we trained up to 15,000 iterations. For α, we initialized to 1e-4, and also used Adam with learning rate 1e-5, then was decayed to 1e-6 after 10,000 iterations. We used mini-batch size $N_{\text{mini}} = 8$. For the meta-update iteration T, larger T gave us only small improvement, so we set to 1. For each training episode, we sample one random future frame uniformly from 1 to 10 ahead. In MetaCREST, we randomly initialized θ_0 and also used Adam with learning rate 1e-6. For α, we initialized to 1e-6, and learning rate of Adam was also set to 1e-6. $N_{\text{mini}} = 8$ and meta-training iterations was 10,000 (at 50,000 iterations, the learning rate was reduced to 1e-7). We used same hyper-parameters for both OTB2015 and VOT2016 experiments. For other hyper-parameters, we mostly followed the ones in the original trackers. For more details, the code and raw results will be released.

5.2 Experimental results

Quantitative evaluation. Table 1 shows quantitative results on VOT2016. In VOT2016, EAO is considered as the main metric since it consider both accuracy

Table 1: Quantitative results on VOT2016 dataset. The numbers in legends represent the number of iterations at the initial frame. EAO (expected average overlap) - 0 to 1 scale, higher is better. A (Accuracy) - 0 to 1 scale, higher is better. R (Robustness) - 0 to N, lower is better. We ran each tracker 15 times and reported averaged scores following VOT2016 convention.

	EAO	Acc	R
MetaCREST-01	**0.317**	**0.519**	**0.932**
CREST	0.283	0.514	1.083
CREST-Base	0.249	0.502	1.383
CREST-10	0.252	0.509	1.380
CREST-05	0.262	0.510	1.298
CREST-03	0.262	0.514	1.283
CREST-01	0.259	0.505	1.277

(a) The results of MetaCREST

	EAO	Acc	R
MetaSDNet-01	**0.314**	0.526	**0.934**
pyMDNet-30	0.304	**0.540**	0.943
pyMDNet-15	0.299	0.541	0.977
pyMDNet-10	0.291	0.535	0.989
pyMDNet-05	0.254	0.523	1.198
pyMDNet-03	0.184	0.488	1.703
pyMDNet-01	0.119	0.431	2.733

(b) The results of MetaSDNet

and robustness. Our meta-trackers, both MetaCREST and MetaSDNet, consistently improved upon their counterparts by significant margins. Note that this is the improvement without their advanced techniques, e.g. pyMDNet with specialized multi-domain training and CREST with spatio-temporal residual modules. The performances of the accuracy metric are not very different than the original trackers. Because it computes the average overlap by only taking successful tracking periods into account, we did not change other factors that might affect the accuracy in the original trackers, e.g. scale estimation. Quantitative results on OTB2015 are depicted in Figure 3. Both of MetaSDNet and MetaCREST also improved upon their counterparts in both precision and success plots with only one iteration for initialization. Detailed results of individual sequences in both of VOT2016 and OTB2015 are presented in Appendix.

We require only one iteration at the initial frame to outperform the original trackers. We also performed the experiments with more than one iteration, but the performance gain was not significant. On the other hand, MDNet takes 30 iterations to converge at the initial frame as reported in their paper, and fewer iterations caused serious performance degradation. This confirms that getting a robust target model at the initial frame is very important for subsequent frames. For CREST, performance drop was not significant as MDNet, but it was still more than 10 iterations to reach to its maximum performance. MDNet updates the model 15 iterations for subsequent frames at every 10 frames regularly (or when it failed, meaning its score is below a predefined threshold).

Speed and performance of the initialization. We reported the wall clock time speed at the initial frame in Table 2, on a single TITAN-X GPU. In CREST, in addition to feature extraction, there are two more computational bottlenecks. The first is the convolution with correlation filters. Larger objects means larger filters and more computations. We reported average time across all 100 sequences. Another heavy computation comes from PCA at the initial

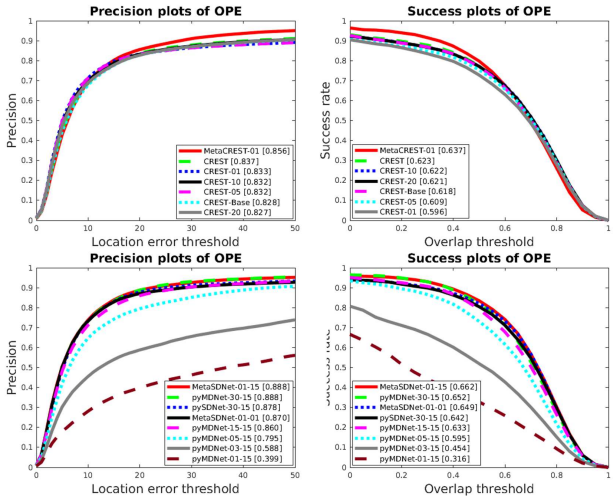


Fig. 3: Precision and success plots over 100 sequences in OTB2015 dataset with one-pass evaluation (OPE). For CREST (top row),The numbers in legends represent the number of iterations at the initial frame, and all used 2 iterations for the subsequent model updates. For MDNet experiments (bottom row), 01-15 means, 1 training iterations at the initial frame and 15 training iterations for the subsequent model updates.

frame. It also depends on the size of the objects. Larger objects lead to larger center cropped images, features, and more computation in PCA.

MDNet requires many positive and negative patches, and also many model update iterations to converge. A large part of the computation comes from extracting CNN features for every patch. MetaSDNet needs only a few training patches and can achieve 30x speedup (0.124 vs 3.508), while improving accuracy. If we used more compact CNNs for feature extractions, the speed could have been in the range of real-time processing. For subsequent frames in MDNet, model update time is of less concern because MDNet only updates the last 3 fully connected layers, which are relatively faster than feature extractors. The features are extracted at every frame, stored in a database, and update the model every 10 frames. Therefore, the actual computation is well distributed across every frames.

We also showed the performance of the initialization to see the effectiveness of our approach (in Table 2. We measured the performance with learned initialization. After initial training, we measure the performance on the first frame and 5 future frames to see generalizability of trackers. MetaSDNet achieved very high accuracy after only one iteration, but accuracy of pyMDNet after one iteration was barely above guessing (guessing is 50% and all negative prediction is 75% accuracy since sampling ratio was 1:3 between positive and negative samples). The effectiveness is more apparent in MetaCREST. MetaCREST-01 without any updates gave already close performance to CREST-05 after training (0.48 vs 0.45). In original CREST tracker, they train the model until it reaches a loss

Table 2: Speed and performance of the initialization: The right table shows the losses of estimated response map in MetaCREST. The left table shows the accuracy of image patches in MetaSDNet. B (Before) - the performance of the initial frame before training, A (After) - the performance of the initial frame after training, LH (Lookahead) - the performance of next 5 frames after training, Time - wall clock time to train in seconds

	B	A	LH	Time(s)
MetaCREST-01	0.48	0.04	**0.05**	**0.090**
CREST-01	0.95	0.82	0.87	0.395
CREST-03	0.95	0.62	0.75	0.424
CREST-05	0.95	0.45	0.63	0.550
CREST-10	0.95	0.24	0.40	0.668
CREST-20	0.95	0.18	0.31	1.048
CREST-65	0.95	0.01	0.30	1.529

	B	A	LH	Time(s)
MetaSDNet-01	0.50	0.98	**0.97**	**0.124**
pyMDNet-01	0.51	0.56	0.56	0.123
pyMDNet-03	0.51	0.79	0.78	0.373
pyMDNet-05	0.51	0.84	0.84	0.656
pyMDNet-10	0.51	0.95	0.93	1.171
pyMDNet-15	0.51	0.97	**0.97**	1.819
pyMDNet-30	0.51	0.99	**0.98**	3.508

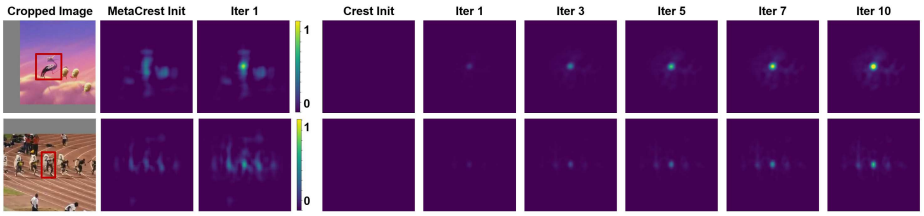


Fig. 4: Visualizations of response maps in CREST: Left three columns represents the image patch at the initial frame, response map with meta-learned initial correlation filters θ_{0_f}, response map after updating 1 iteration with learned α, respectively. The rest of seven columns on the right shows response maps after updating the model up to 10 iterations.

of 0.02, which corresponds to an average 65 iterations. However, its generalizability at future frames is limited compared to ours (.05 vs .30). Although this is not directly proportional to eventual tracking performance, we believe this is clear evidence that our meta-training algorithm based on future frames is indeed effective, as also supported by overall tracking performance.

Visualization of response maps. We visualized response maps in MetaCREST at the initial frame (Figure 4). A meta-learned initialization, θ_0 should be capable of learning generic objectness or visual saliency. At the same time, it should not be instance specific. It turns out that is the case. The second column in Figure 4 shows response maps by applying correlation filters to the cropped image (first column) with θ_0. Without any training, it already generates high response values on some locations where there are objects. But, more importantly, there is no clear maximum. After one iteration, the maximum is clearly located at the center of the response map. In contrast to MetaCREST, CREST consumes more iterations to produce high response values on the target.

Qualitative examples of robust initialization. In Figure 5, we present some examples where MetaCrest overcomes some of the issues in the original CREST.

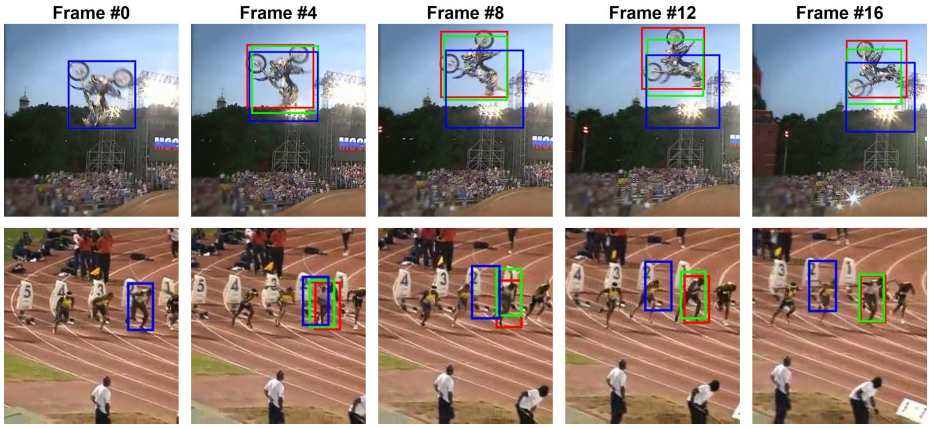


Fig. 5: Qualitative examples: tracking results at early stage of MotorRolling (top) and Bolt2 (bottom) sequences in OTB2015 dataset. Color coded boxes: ground Truth (Red), MetaCREST-01 (Green) and CREST (Blue).

In MotorRolling sequence (top row), CREST was distracted by a horizontal line from the forest in the background. CREST easily reached to 0.0000 loss defined in Equation 6 at the initial frame, as opposed to 0.1255 in MetaCREST. This is a strong evidence that an optimal solution does not necessarily mean good generalizability on future frames. In contrast, MetaCREST, generalizes well to future frames, despite not finding an optimal solution at the current frame. In Bolt2 sequence (bottom row), CREST also reached to 0.0000 loss (vs 0.0534 in MetaCREST). In a similar way, a top left part in the bounding box was the distractor. MetaCREST could easily ignore the background clutter and focused on the object in the center of the bounding box.

6 Conclusion and future work

In this paper, we present an approach to use meta-learning to improve online trackers based on deep networks. We demonstrate this by improving two state-of-the-art trackers (CREST and MDNet). We learn to obtain a robust initial target model based on the error signals from future frames during meta-training phase. Experimental results show improvements in speed, accuracy, and robustness for both trackers. The proposed technique is general so that other trackers may benefit from it as well.

Other than target appearance modeling, which is the focus of this paper, there are many additional important factors in object tracking algorithms. For example, when or how often to update the model [56], how to manage the database [6], and how to define the search space. These considerations are sometimes more important than target appearance modeling. In future work we propose including handling of these as part of learning and meta-learning.

Acknowledgements We thank the reviewers for their valuable feedback and acknowledge support from NSF 1452851, 1526367, 1446631.

References

1. Nam, H., Han, B.: Learning multi-domain convolutional neural networks for visual tracking. In: CVPR (2016)
2. Song, Y., Ma, C., Gong, L., Zhang, J., Lau, R., Yang, M.H.: CREST: Convolutional residual learning for visual tracking. In: ICCV (2017)
3. Wu, Y., Lim, J., Yang, M.H.: Object tracking benchmark. TPAMI (2015)
4. Kristan, M., Leonardis, A., Matas, J., Felsberg, M., et al: The visual object tracking vot2016 challenge results. In: ECCV Workshop (2016)
5. Danelljan, M., Robinson, A., Shahbaz Khan, F., Felsberg, M.: Beyond Correlation Filters: Learning continuous convolution operators for visual tracking. In: ECCV (2016)
6. Danelljan, M., Bhat, G., Shahbaz Khan, F., Felsberg, M.: ECO: Efficient convolution operators for tracking. In: CVPR (2017)
7. Henriques, J.F., Caseiro, R., Martins, P., Batista, J.: High-speed tracking with kernelized correlation filters. TPAMI (2015). https://doi.org/10.1109/TPAMI.2014.2345390
8. Ma, C., Huang, J.B., Yang, X., Yang, M.H.: Hierarchical convolutional features for visual tracking. In: ICCV (2015)
9. Kalal, Z., Mikolajczyk, K., Matas, J.: Tracking-learning-detection. TPAMI (2010)
10. Bolme, D.S., Beveridge, J.R., Draper, B.A., Lui, Y.M.: Visual object tracking using adaptive correlation filters. In: CVPR (2010)
11. Bertinetto, L., Valmadre, J., Henriques, J.F., Vedaldi, A., Torr, P.H.: Fully-convolutional siamese networks for object tracking. ECCV Workshop (2016)
12. Held, D., Thrun, S., Savarese, S.: Learning to track at 100 fps with deep regression networks. In: ECCV (2016)
13. Tao, R., Gavves, E., Smeulders, A.W.M.: Siamese instance search for trackin. In: CVPR (2016)
14. Mueller, M., Smith, N., Ghanem, B.: Context-aware correlation filter tracking. In: CVPR (2017)
15. Finn, C., Abbeel, P., Levine, S.: Model-agnostic meta-learning for fast adaptation of deep networks. In: ICML (2017)
16. Ravi, S., Larochelle, H.: Optimization as a model for few-shot learning. In: ICLR (2017)
17. Andrychowicz, M., Denil, M., Colmenarejo, S.G., Hoffman, M.W., Pfau, D., Schaul, T., Shillingford, B., Freitas, N.d.: Learning to learn by gradient descent by gradient descent. In: NIPS (2016)
18. Santoro, A., Bartunov, S., Botvinick, M., Wierstra, D., Lillicrap, T.: Meta-learning with memory-augmented neural networks. In: ICML (2016)
19. Li, Z., Zhou, F., Chen, F., Li, H.: Meta-SGD: Learning to learn quickly for few shot learning. arXiv:1707.09835 (2017)
20. Al-Shedivat, M., Bansal, T., Burda, Y., Sutskever, I., Mordatch, I., Abbeel, P.: Continuous adaptation via meta-learning in nonstationary and competitive environments. In: ICLR (2018)
21. Danelljan, M., Hager, G., Khan, F.S., Felsberg, M.: Learning spatially regularized correlation filters for visual tracking. In: ICCV (2015)
22. Galoogahi, H.K., Sim, T., Lucey, S.: Correlation filters with limited boundaries. In: CVPR (2015)
23. Zhang, K., Zhang, L., Liu, Q., andMing Hsuan Yang, D.Z.: Fast visual tracking via dense spatio-temporal context learning. In: ECCV (2014)

24. Ma, C., Yang, X., Zhang, C., Yang, M.H.: Long-term correlation tracking. In: CVPR (2015)
25. Hong, Z., Chen, Z., Wang, C., Mei, X., Prokhorov, D., Tao, D.: Multi-store tracker (muster): a cognitive psychology inspired approach to object tracking. In: CVPR (2015)
26. Danelljan, M., Hager, G., Khan, F.S., Felsberg, M.: Accurate scale estimation for robust visual tracking. In: BMVC (2014)
27. Valmadre, J., Bertinetto, L., Henriques, J.F., Vedaldi, A., Torr, P.H.S.: End-to-end representation learning for correlation filter based tracking. In: CVPR (2017)
28. Li, H., Li, Y., Porikli, F.: Deeptrack: Learning discriminative feature representations by convolutional neural networks for visual tracking. In: BMVC (2014)
29. Babenko, B., Yang, M.H., Belongie, S.: Robust object tracking with online multiple instance learning. TPAMI (2010)
30. Hare, S., Golodetz, S., Saffari, A., Vineet, V., Cheng, M.M., Hicks, S.L., Torr, P.H.S.: Struck: Structured output tracking with kernels. TPAMI (2015)
31. Grabner, H., Leistner, C., Bischof, H.: Semi-supervised on-line boosting for robust tracking. In: ECCV (2008)
32. Bai, Q., Wu, Z., Sclaroff, S., Betke, M., Monnier, C.: Randomized ensemble tracking. In: ICCV (2013)
33. Fischer, P., Dosovitskiy, A., Ilg, E., Hausser, P., Hazrbas, C., Golkov, V.: FlowNet: Learning optical flow with convolutional networks. In: CVPR (2015)
34. Kahou, S.E., Michalski, V., Memisevic, R.: RATM: Recurrent attentive tracking model. CVPR Workshop (2017)
35. Gan, Q., Guo, Q., Zhang, Z., Cho, K.: First step toward model-free, anonymous object tracking with recurrent neural networks. arXiv:1511.06425 (2015)
36. Gordon, D., Farhadi, A., Fox, D.: Re3: Real-time recurrent regression networks for object tracking. arXiv:1705.06368 (2017)
37. Yang, T., Chan, A.B.: Recurrent filter learning for visual tracking. In: ICCV (2017)
38. Schmidhuber, J.: Evolutionary principles in self-referential learning. Diploma thesis, Institut f. Informatik, Tech. Univ. Munich (1987)
39. Schmidhuber, J.: Learning to control fast-weight memories: an alternative to dynamic recurrent networks. Neural Computation (1992)
40. Hochreiter, S., Younger, A.S., Conwell, P.R.: Learning to learn using gradient descents. ICANN (2001)
41. Thrun, S., Pratt, L.: Learning to learn: introduction and overview. Springer (1998)
42. Chen, Y., Hoffman, M.W., Colmenarejo, S.G., Denil, M., Lillicrap, T.P., Botvinick, M., de Freitas, N.: Learning to learn without gradient descent by gradient descent. In: ICML (2017)
43. Wichrowska, O., Maheswaranathan, N., Hoffman, M.W., Colmenarejo, S.G., Denil, M., de Freitas, N., Sohl-Dickstein, J.: Learned optimizers that scale and generalize. In: ICML (2017)
44. Li, K., Malik, J.: Learning to optimize. In: ICLR (2017)
45. Bertinetto, L., Henriques, J.F., Valmadre, J., Torr, P.H.S., Vedaldi, A.: Learning feed-forward one-shot learners. In: NIPS (2016)
46. Wang, Y.X., Hebert, M.: Learning to Learn: Model regression networks for easy small sample learning. In: ECCV (2016)
47. Kingma, D.P., Ba, J.L.: Adam: A method for stochastic optimization. In: ICLR (2015)
48. Maclaurin, D., Duvenaud, D., Adams, R.P.: Gradient-based hyperparameter optimization through reversible learning. In: ICML (2015)

49. Metz, L., Poole, B., Pfau, D., Sohl-Dickstein, J.: Unrolled generative adversarial networks. In: ICLR (2017)
50. Pytorch, http://www.pytorch.org.
51. Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.Y., Berg, A.C.: Ssd: Single shot multibox detector. In: ECCV (2016)
52. Ren, S., He, K., Girshick, R., Sun, J.: Faster R-CNN: Towards real-time object detection with region proposal networks. In: NIPS (2016)
53. Jaderberg, M., Simonyan, K., Zisserman, A., Kavukcuoglu, K.: Spatial transformer networks. In: NIPS (2016)
54. Kristan, M., Leonardis, A., Matas, J., Felsberg, M., et al: The visual object tracking vot2015 challenge results (2015)
55. Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A.C., Fei-Fei, L.: ImageNet Large Scale Visual Recognition Challenge. IJCV (2015)
56. Supancic, J., Ramanan, D.: Tracking as online decision-making: Learning a policy from streaming videos with reinforcement learning. In: ICCV (2017)