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 on-line trackers that use deep learning. Such trackers train a deep network to pick a specified object out from the background in an initial frame (initialization) and then keep training the model as tracking proceeds (updates). Our core contribution is a meta-learning-based method to adjust deep networks for tracking using off-line training. First, we learn initial parameters and per-parameter coefficients for fast online adaptation. Second, we use training signal from future frames for robustness to target appearance variations and environment changes. The resulting networks train significantly faster during the initialization, while improving robustness and accuracy. We demonstrate this approach on top of the current highest accuracy tracking approach, tracking-by-detection based MDNet[39] and close competitor, the correlation-based CREST[46]. Experimental results on both standard benchmarks, OTB[53] and VOT2016[29], show improvements in speed, accuracy, and robustness on both trackers.

1. Introduction

Visual object tracking is a task that locates target objects precisely over a sequence of image frames given a target bounding box in 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. This challenge invites applying powerful deep-learning-based approaches, but these in turn pose difficulties in the tracking setting where speed is of the essence and very few training examples are provided for a target. This paper will present a meta-approach to alleviating these issues in state-of-the-art trackers.

Going back to the current state of the art in tracking, in order to carefully distinguish the target from background and distractors, the main stream of object tracking methods have been based on online target appearance modeling [12, 46, 39, 9, 21, 34, 26, 7]. The target model, e.g. DCF(discriminative correlation filter) or binary classifiers (the object vs backgrounds), is obtained at the first frame, and the model is continuously updated to be well-adapted to target appearance variations over time. With the recent emergence of powerful generic deep representations, recent top performing trackers are now leveraging the best of both worlds: deep learned features and online adaptation methods. Offline-only trained trackers with deep learning have also been suggested, with promising results and high speed, but accuracy has fallen short compared to state-of-the-art online adaptive trackers [6, 20, 48]. We believe this may be due to their inability to finely discriminate specific instances in videos.

A common practice to combine deep learning features and online adaptation is to train a target model on top of deep 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 these type of approaches have shown the best results so far, there remains several important issues to be resolved.

First, most state-of-the-art trackers spend a significant amount of time on the initial training stage [39, 9, 46]. Although many works have proposed fast training methods [21, 9], 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 a robust target model very quickly at the initial frame.
Another challenge lies in the fact that, in the tracking setting, we have very few training examples. In the initial frame, we are given a single sample. In subsequent frames, trackers collect additional images, but many are redundant since they are essentially the same target and background. Using pre-trained generic deep learned features, which are known to be very robust and highly discriminative, may alleviate this issue. However, a target model trained on top of them sometimes suffers because it overfits to the background clutter, small parts or features of the target, or noise. Furthermore, recent trends toward building deep models for target appearance [39, 46] make the problem more challenging since deep models are known to be vulnerable to overfitting on small datasets. 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 [39], factorized convolution [9], spatio_residual modules [46], or incorporating contextual information [38].

In this work, we propose a generic and principled way of tackling these challenges. Inspired by recent meta-learning (learning to learn) studies [13, 40, 2, 43, 32], we seek to find a generic initial representation for trackers that enables quickly fitting the target model at the first frame. In addition, we also propose to automatically learn fast and robust gradient directions in place of using hand-designed optimization algorithms and regularizers [40, 2, 32]. Thanks to recent efforts to collect a large-scale video detection dataset [42], we can mimic an actual tracking scenario during the meta-training process.

Our proposed technique can be applied to any learning based tracker with minor modifications. To show the effectiveness of the proposed technique, we select two state-of-the-art trackers, MDNet [39], from the classifier based tracker (tracking-by-detection) category, and CREST [46], a correlation filter based tracker. Experimental results show that our meta-learned version of these trackers can adapt very quickly—just one iteration—for the first frame without losing accuracy (in contrast to 10 to 30 in the original papers). 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. This improved performance is due to our proposed meta-training process. 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 its computational efficiency and discriminative power. From the early success of the MOSSE tracker [7], a large number of variations have been suggested. [21] makes it more efficient by taking advantage of circulant matrices, further improved by resolving artificial boundary issues [11, 15]. Many hard cases have been tackled by using context information [55, 38], short and long-term memory [35, 23], and scale-estimation [10], just to name a few. Recently, deep learning features have begun to play an important role in correlation filters [46, 12, 9, 34, 50, 39, 30]. 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 [26], many learning techniques have been suggested, e.g. multiple instance learning [3], structured output SVMs [19], online boosting [18], and model ensembles [4]. More recently, MDNet [39], with pre-trained deep features and a deep classifier model, achieved significantly higher accuracy.

Offline trackers: Several recent studies have shown that we can build accurate trackers without online adaptation [6, 48, 20] 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 [20] or generate a response map [6] via a correlation layer [14]. In order to consider temporal information, recurrent networks have also been explored in [25, 16, 17, 54].

Learning to optimize: Meta-learning, also known as learning-to-learn, is an emerging field in machine learning and its applications. Although it is not a new concept [44, 45, 22, 49], many recent works have shown very promising results along with deep learning success. [2, 8, 52, 31] attempted to replace hand-crafted optimization algorithms with meta-learned deep networks. [40] 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, [13] focuses on learning initialization that are most suitable for existing algorithms. [32] 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 [5, 51, 54].

3. Meta-Learning for Visual Object Trackers

In this section, we explain the proposed generic meta-training framework for visual object trackers (Figure 1). The details for each tracker are in Section 4.

3.1. Motivation

A typical tracking episode is the following. Trackers are updated based on the initial frame and bounding box cen-
Figure 1. Overview of meta-training for visual object tracking: A computational graph in meta-training object trackers at the initial frame. It gets the gradients of the loss on the first frame, and a meta-updater updates parameters of the tracker for every iterations with those gradients. Final loss is computed from a future frame, and red arrows are used to compute the gradients w.r.t parameters of meta-initializer and meta-updater. More details in section 3.

Algorithm 1 Meta-training object trackers algorithm

Input: Randomly initialized \( \theta_0 \) and \( \alpha \), training dataset \( D \)

Output: \( \theta_0^* \) and \( \alpha^* \)

1: while not converged do
2: \( \operatorname{grad}_{\theta_0}, \operatorname{grad}_{\alpha} = 0 \) \( \triangleright \) Initialize to zero vector
3: for all \( k \in \{0, \ldots, N_{\text{mini}} - 1\} \) do
4: \( S, j, \delta \sim p(D) \) \( \triangleright \) Sample a training example
5: \( \theta_0^k = \theta_0 \)
6: for all \( t \in \{0, \ldots, T - 1\} \) do
7: \( \hat{y}_j = F(x_j; \theta_0^k) \)
8: \( \theta_0^{k+1} = \theta_0^k - \alpha \odot \nabla_{\theta_0} L(y_j, \hat{y}_j; \theta_0^k) \)
9: end for
10: \( \theta_1 = \theta_0^k \)
11: \( \hat{y}_{j+\delta} = F(x_{j+\delta}; \theta_1) \) \( \triangleright \) Apply to a future frame
12: \( \operatorname{grad}_{\theta_0} = \operatorname{grad}_{\theta_0} + \nabla_{\theta_0} L(y_{j+\delta}, \hat{y}_{j+\delta}) \)
13: \( \operatorname{grad}_{\alpha} = \operatorname{grad}_{\alpha} + \nabla_{\alpha} L(y_{j+\delta}, \hat{y}_{j+\delta}) \)
14: end for
15: \( \theta_0 = \text{Optimizer}(\theta_0, \operatorname{grad}_{\theta_0}) \) \( \triangleright \) Update \( \theta_0 \)
16: \( \alpha = \text{Optimizer}(\alpha, \operatorname{grad}_{\alpha}) \) \( \triangleright \) Update \( \alpha \)
17: end while

3.2. A general on-line 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 information \( x \) from a frame and the tracker parameters \( \theta \) and produces an estimate \( \hat{y} \) of the label, e.g. a response map or a location in the frame. For initialization, on a frame \( x_0 \) with specified \( y_0 \), we (approximately) solve for \( \theta_1(x_0, y_0) \), or \( \theta_1 \) for brevity, with respect to a loss, \( L(F(x_0, \theta_1), y_0) \). For updates during tracking, we take the parameters \( \theta_j \) from frame \( j - 1 \) and find \( \hat{y}_j = F(x_j, \theta_j) \), then find \( \theta_{j+1} \) with respect to a loss that may incorporate transforming \( \hat{y}_j \) into a specific estimate of the target location as well as temporal smoothing, etc. Given a video sequence \( S = \{(x_j, y_j)\} \), combining initialization that finds \( \theta_1 \) for \( x_0 \) with specified \( y_0 \) and tracking for \( n \) subsequent frames with updates, we denote the estimate after frame \( x_n \) as \( (y_n, \theta_{n+1}) = \text{Track}(\theta_1(x_0, y_0), x_1, \ldots, x_n) \) and may drop the second output for brevity.

3.3. Meta-training algorithm

Our meta-training approach has two goals. One is that initialization for a tracker can be performed by starting with \( \theta_0 \) and applying 1 or a very small number of iterations of a learning step \( M \) parameterized by \( \alpha \). Another goal is that the resulting tracker be accurate and robust on later frames.

The gradient-descent style update rule \( M \) is parameterized by \( \alpha \):

\[
M(\theta, \nabla_{\theta} L; \alpha) = \theta - \alpha \odot \nabla_{\theta} L
\] (1)
where \(\alpha\) is the same size as the tracker parameters and \(\odot\) is element-wise product. This form of meta-update function was also suggested in [32], and we found it simple and effective in our setting.

Our meta-training algorithm to find a good \(\theta_0\) and \(\alpha\) by repeatedly sampling a video, performing initialization, then applying the learned initial model to a frame slightly ahead in the sequence, and back-propagating to update \(\theta_0\) and \(\alpha\). 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 to \(\theta\) during tracking fast as well if not much needs to be fixed.

After sampling a random starting frame with ground truth, \((x_j, y_j)\), from a random video. We perform optimization for initialization starting with \(\theta_0^0 = \theta_0\). A step of optimization proceeds as

\[
\theta_0^{j+1} = M(\theta_0^j, \nabla_{\theta_0} \mathcal{L}(y_j, F(x_j; \theta_0^j))).
\]

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 \(j + \delta\) and evaluate the model trained on the initial frame on that future frame to produce: \(\hat{y}_{j+\delta} = F(\theta_1, x_{j+\delta})\).

The larger \(\delta\), 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 tracked tracker parameters. The objective function is defined as

\[
\theta_0^*, \alpha^* = \arg\min_{\theta_0, \alpha} \mathbb{E}_{S,j,\delta} [\mathcal{L}(y_{j+\delta}, \hat{y}_{j+\delta})] \tag{3}
\]

We used the ADAM [27] gradient descent algorithm to optimize. Note that \(\theta_0\) and \(\alpha\) are fixed across different episodes in a mini-batch, but \(\theta_0^T\) is changed over every episode. To compute gradients of the objective function w.r.t \(\theta_0\) and \(\alpha\), it is required to compute higher-order gradients (the gradients of function of gradients). This type of computation has been exploited in recent studies [36, 37, 13]. We can easily compute this thanks to recent automatic differentiation software libraries [1]. More details are explained in Algorithm 1.

In this work, we mainly focus on obtaining robust \(\theta_1\) very quickly by using meta-learned \(\theta_0, \alpha\). For the rest of the tracker parameters, \(\theta_{2:N}\), we used ADAM optimization algorithms with pre-defined learning rates. In current experiments, applying \(\alpha\) to obtain \(\theta_{2:N}\) did not give any further improvement. We could also learn update rules for subsequent frames with similar meta-training algorithms, e.g.

\[\alpha_{2:N}.\] We leave it as future work.

4. MetaTrackers

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 tracking-by-detection based trackers MDNet [39], and one from correlation filter based trackers, CREST [46].

4.1. Meta-training of tracking-by-detection tracker

4.1.1 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 2). 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 these devised techniques, we aim to obtain robust and quickly adaptive classifier solely resting on our proposed meta-learning process.

4.1.2 Meta-training

In our proposed meta-training framework for MDNet, an input \(x_j \in \mathbb{R}^{N \times D}\) is a set of image patches and \(y_j \in \{0, 1\}^N\) is the corresponding labels, where \(D\) is size of the patches and \(N\) is the number of patches. Then, \(L(x_j; \theta)\) becomes the CNN classifier, and the loss function \(L\) is a simple cross entropy loss.

\[- \frac{1}{N} \sum_{k=1}^{N} y_j^k \log(F_k(x_j; \theta))\]  

Figure 2. MDNet vs MetaSDNet
they varied depending on the target shape. We could resize all objects to the same size and same aspect ratio. However, it will introduce distortion of the target and has been known to degrade recognition performance [33, 41]. In order to fully make use of the power of the correlation filter, we propose canonical size initialization $\theta_0$ and learning parameter of meta-update function $\alpha$. As depicted in Figure 4.2, we share a global initialization for every tracking episode and its size and aspect ratio is calculated as a mean of the objects in the training dataset. At the initial frame, we warp the canonical size initialization to the specific target object, $\theta_0^0 = \text{Warp}(\theta_0)$. We used differentiable bilinear sampling method [24].

**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 trained at the initial frame and used for the rest of the sequence. This gives rise to similar issue as aspect ratio problem. We seek to find a global initialization of the correlation filter for every object, but PCA changes the basis for each different sequence. Similar to the warping trick, we meta-learn how to reduce the dimensionality of the input $x_j$. 

And, $\text{Track}(\theta, x_{j+1}, \ldots, x_{j+\delta})$ is $F(x_{j+\delta}; \theta)$ since MDNet is a stateless tracker. Both can be easily plugged into the suggested meta-training framework to complete the training algorithm.

**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 [43]. 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.

**4.2. Meta-training of CF based tracker**

**4.2.1 CREST**

A typical correlation filter objective is defined as follows.

$$\argmin_f ||y - x * f||^2 + \lambda||f||^2$$

(5)

where $f$ is the correlation filter, $x$ is a cropped image centered around the target, and $y$ is a gaussian shaped response map. The cropped image is usually larger than the target object so that it can provide enough background information. Once we have the correlation filter, target localization at the next frame is simply finding the location that has the maximum response value.

$$\argmax_{h,w} R(h, w), \quad \text{where} \quad R = x_{\text{new}} * f$$

(6)

They [46] claimed that it is unlikely that a single layer correlation filter can generate accurate response maps. By reformulating the correlation filter as a one-layer convolutional layer, it can be effectively integrated into an CNN framework. This allows us to add new modules easily, since the optimization can be nicely done with standard gradient descent algorithm 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 any of suggested techniques, we aim to learn a robust single layer correlation filter via proposed meta-learning process.

**4.2.2 Meta-training**

Similarly, an input $x_j \in \mathbb{R}^{3 \times H \times W}$ is a cropped image centered around the target. Then, the label $y_j \in \mathbb{R}^{h \times w}$ is the corresponding ground truth response map, where $H, W$ are 4 times larger than $h, w$ after two max-pooling operations. $F(x_j; \theta)$ consists of a CNN feature extractor, a 1x1 convolution for dimensionality reduction (explained in next subsection), and a correlation filter layer. CREST is also a stateless tracker, so $\text{Track}(\theta, x_{j+1}, \ldots, x_{j+\delta})$ is simply $F(x_{j+\delta}; \theta)$. And, the loss function $L$ is defined as $||y_j - F(x_j; \theta)||^2$. Although it can be easily plugged into the meta-training framework, there remains two important issues to be resolved.

**Canonical size initialization** The size of the correlation filter varies depending on the target shape. We could resize all objects to the same size and same aspect ratio. However, it will introduce distortion of the target and has been known to degrade recognition performance [33, 41]. In order to fully make use of the power of the correlation filter, we propose canonical size initialization $\theta_0$ and learning parameter of meta-update function $\alpha$. As depicted in Figure 4.2, we share a global initialization for every tracking episode and its size and aspect ratio is calculated as a mean of the objects in the training dataset. At the initial frame, we warp the canonical size initialization to the specific target object, $\theta_0^0 = \text{Warp}(\theta_0)$. We used differentiable bilinear sampling method [24].
5. Experiments

5.1. Experimental setup

**OTB (Object Tracking Benchmark)** [53]: It consists of 100 fully annotated video sequences. Two evaluation metrics are commonly used, bounding box overlap (IoU-intersection over union) and center location distance. The precision plot represents how many times the distances of center locations between our predictions and ground truths are within pre-defined threshold (from 0 to 50, interval 1). The success plots show how many times the IoUs are within pre-defined threshold (from 0 to 1, interval 0.05). We used the one pass evaluation (OPE) with two standard metrics, and the overall score is defined as area-under-the-curve.

**VOT2016 (Visual Object Tracking Challenge)** [29]: It contains 60 videos (same videos from VOT 2015 [28]). The main difference from OTB is the evaluation metrics. Unlike one-pass evaluation, the trackers are automatically reinitialized once it drifts off the target: zero overlap between predictions 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 (The more details in [28]).

**Dataset for meta-training**: We used a large scale video detection dataset [42] 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 OTB dataset, we also used an additional 58 sequences from object tracking datasets in VOT2013, VOT2014, and VOT2015 [29], excluding the videos included in OTB, following MDNet’s approach [39]. These sequences were selected in the mini-batch selection stage with the probability 0.3. Similarly, we used 80 sequences from OTB, excluding the videos in VOT2016 for the experiments on VOT2016 dataset.

**Baseline implementations** We selected two trackers, MDNet [39] and CREST [46]. The authors of MDNet provide two different public source codes, written in MATLAB and python, respectively. We used the latter one (written in python, called it as pyMDNet) for meta-training. Note that overall accuracy of pyMDNet is lower than MDNet on OTB (.652 vs .678 in success rates with overlap metric). For fair comparison, we compared our MetaSDNet to pyMDNet. For CREST, we re-implemented our own version (written in python) based on their code (written in MATLAB), and we used this version for meta-training.

**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 ($\theta_0$), 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 $\alpha$, 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. In MetaCREST, we randomly initialized $\theta_0$ and also used Adam with learning rate 1e-6. For $\alpha$, 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 OTB and VOT experiments. For other hyper-parameters, we mostly followed the ones in the original trackers. For more details, please refer to the open source code.

5.2. Experimental results

5.2.1 Quantitative evaluation

Quantitative results on OTB are depicted in Figure 4. A comparison (CREST-65 vs MetaCREST, other comparisons will be provided in supplementary materials) of individual sequences are also presented in Figure 5. Both of MetaSDNet and MetaCREST outperformed their baseline counterparts (pySDNet and CREST-Base) by a significant margin in both precision and success plots. Ours also outperformed their advanced models, pyMDNet with specialized multi-domain training and CREST with spatio-temporal residual modules. In terms of tracking efficiency, ours requires only one iteration to get close to maximum performance at the initial frame. We also tried more than one iterations, but the performance gain was not significant. For MDNet, it 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). Surprisingly, only one model
Figure 4. Precision and success plots over 100 sequences in OTB-2015 dataset with one-pass evaluation(OPE). The numbers along with the legends represent the area-under-the-curve. Left two figures are results of MDNet and right two are results of CREST. For MDNet experiments, 01-15 means, 1 training iterations at the initial frame and 15 training iterations for the subsequent model updates. For CREST, only iterations at the initial frame are presented, and all used 2 iterations for the subsequent model updates.

Figure 5. Success rates(with IoU metric) of 100 sequences in OTB-2015 dataset for CREST-65 vs MetaCREST-01 updates in subsequent frames of our model(MetaSDNet-01-01) also outperformed its baseline(pySDNet-30-15).

We also made a great improvement on VOT2016. Table 1 and 2 show quantitative results on VOT2016. In VOT2016, EAO is considered as the main metric since it consider both accuracy and robustness. Our meta-trackers, both MetaSDNet and MetaCREST, consistently improved upon their original trackers. The fact that we have a good initialization and gradient update directions at the initial frame, had a great influence on the entire tracking process, which resulted in performance improvement on both robustness and EAO metrics. Note that our technique do not change any other factors in the original tracking algorithms, e.g. scale estimation, which might affect the performance of precise localization. Therefore, the accuracy performance remains about the same as the original tracker since this metric computes the average overlap by only taking successful tracking periods into account.

5.2.2 Tracking time

We reported the wall clock time speed at the initial frame in Table 3 and 4, on a single TITAN-X GPU. As we mentioned earlier, 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 train-

Table 1. Results of MDNet trackers on VOT-2016. EAO(expected average overlap) - 0 to 1 scale(higher is better), Accuracy - 0 to 1 scale(higher is better), Robustness - 0 to N(lower is better). The numbers in legends represent the number of iterations at the initial frame. We ran each tracker 15 times and reported averaged scores following VOT2016 convention.

| Tracker      | EAO   | Accuracy | Robustness |
|--------------|-------|----------|------------|
| MetaSDNet-01 | .314  | .526     | .934       |
| pyMDNet-30   | .304  | .540     | .943       |
| pyMDNet-15   | .299  | .541     | .977       |
| pyMDNet-10   | .291  | .535     | .989       |
| pyMDNet-05   | .254  | .523     | 1.198      |
| pyMDNet-03   | .184  | .488     | 1.703      |
| pyMDNet-01   | .119  | .431     | 2.733      |

Table 2. Results of CREST trackers on VOT-2016. The numbers in legends represent the number of iterations at the initial frame.

| Tracker      | EAO   | Accuracy | Robustness |
|--------------|-------|----------|------------|
| MetaCREST-01 | .317  | .519     | .932       |
| CREST        | .283  | .514     | 1.083      |
| CREST-Base   | .249  | .502     | 1.383      |
| CREST-10     | .252  | .509     | 1.380      |
| CREST-05     | .262  | .510     | 1.298      |
| CREST-03     | .262  | .514     | 1.283      |
| CREST-01     | .259  | .505     | 1.277      |

\(^1\)We reported the result from the original paper

We reported the result from the original paper

We reported the result from the original paper

We reported the result from the original paper

We reported the result from the original paper

We reported the result from the original paper

We reported the result from the original paper

We reported the result from the original paper

We reported the result from the original paper
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, and they are stored in a database, and used for the model update for every 10 frames. Therefore, the actual computation is well distributed across every frames. MetaSDNet also achieved better and on-par performance with only one iteration in subsequent frames. We believe it could be improved by meta-learning update rules for subsequent frames. We will leave it as future work. In CREST, in addition to feature extraction, there are two more computational bottlenecks. The first is the convolutions 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 dimensionality reduction processed at the initial frame. It also depends on the size of the objects. Larger objects lead to larger center cropped images, features, and more computation in PCA.

5.2.3 Performance at the initial frame

We also showed the performance of loss functions at the initial frames to see the effectiveness of our approach (in Table 3 and 4). 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 chance (the chance 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 the CREST tracker. Meta-learned initialization without any updates gave already close performance to CREST-05. In original CREST, they trained the model until it reaches a loss of 0.02, which corresponds to an average 65 iterations. However, its generalizability at future frames is limited compared to ours (.053 vs .298). 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 (Figure 4).

5.2.4 Visualization of response maps

We visualized response maps in MetaCREST at the initial frame (Figure 6). A meta-learned initialization 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 6 shows response maps by applying correlation filters to the cropped image (first column) with meta-learned initialization. With-

![Figure 6. 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 $\theta_0$, response map after updating 1 iteration with learned $\alpha$, respectively. The rest of seven columns on the right shows response maps after updating the model up to 15 iterations.](image-url)
out any training, it already generates high response values on some locations where there are objects. But, more importantly, there is no clear maximum. Right after one iteration, the maximum became apparent located at the center of the response map. In contrast to MetaCREST, CREST consumes more iterations to produce high response values on the target.

6. Conclusion

In this paper, we improved upon two state-of-the-art trackers by learning fast and robust online adaption via meta-learning approach. Experimental results have shown the importance of initialization and learned update rules on the standard object tracking benchmark. We also achieved significant tracking speed-up at the initial frame, which will be very useful in many practical applications. The proposed technique is also very general so that many other trackers can benefit from it. We showed this is indeed the case by applying to two different types of trackers.

We focus on learning good initialization and gradient direction for every arbitrary object. Although we showed it adapts very well to specific instances after one model update iteration, we believe initialization and learning rules also should be conditioned on the target (e.g. $\theta_0(x)$, $\alpha(x)$). There have been a few studies in this direction [40, 5, 54] and we hope this will further improve on current meta-trackers.

Other than target appearance modeling that is the focus of this paper, there are many other important factors in object tracking algorithms. For example, when or how often to update the model [47], how to manage the database [9], how to define the search space, and so on. These hyperparameters are sometimes more important than target appearance modeling. We believe that the principled ways of handling these tracking strategies can be effectively performed by learning-based approaches.

References

[1] Pytorch. http://www.pytorch.org

[2] M. Andrychowicz, M. Denil, S. G. Colmenarejo, M. W. Hoffman, D. Pfau, T. Schaul, B. Shillingford, and N. d. Freitas. Learning to learn by gradient descent by gradient descent. In ICML, 2016. 2

[3] B. Babenko, M.-H. Yang, and S. Belongie. Robust object tracking with online multiple instance learning. TPAMI, 2010. 2

[4] Q. Bai, Z. Wu, S. Sclaroff, M. Betke, and C. Monnier. Randomized ensemble tracking. In ICCV, 2013. 2

[5] L. Bertinetto, J. F. Henriques, J. Valmadre, P. H. S. Torr, and A. Vedaldi. Learning feed-forward one-shot learners. In NIPS, 2016. 2, 9

[6] L. Bertinetto, J. Valmadre, J. F. Henriques, A. Vedaldi, and P. H. Torr. Fully-convolutional siamese networks for object tracking. arXiv:1606.09549, 2016. 1, 2

[7] D. S. Bolme, J. R. Beveridge, B. A. Draper, and Y. M. Lui. Visual object tracking using adaptive correlation filters. In CVPR, 2010. 1, 2

[8] Y. Chen, M. W. Hoffman, S. G. Colmenarejo, M. Denil, T. P. Lillicrap, M. Botvinick, and N. d Freitas. Learning to learn without gradient descent by gradient descent. In ICML, 2017. 2

[9] M. Danelljan, G. Bhat, F. Shahbaz Khan, and M. Felsberg. Eco: Efficient convolution operators for tracking. In CVPR, 2017. 1, 2, 9

[10] M. Danelljan, G. Hager, F. S. Khan, and M. Felsberg. Accurate scale estimation for robust visual tracking. In BMVC, 2014. 2

[11] M. Danelljan, G. Hager, F. S. Khan, and M. Felsberg. Learning spatially regularized correlation filters for visual tracking. In ICCV, 2015. 2

[12] M. Danelljan, A. Robinson, F. Shahbaz Khan, and M. Felsberg. Beyond correlation filters: Learning continuous convolution operators for visual tracking. In ECCV, 2016. 1, 2

[13] C. Finn, P. Abbeel, and S. Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In ICML, 2017. 2, 4

[14] P. Fischer, A. Dosovitskiy, E. Ilg, P. Hausser, C. Hazrbas, and V. Golkov. Flownet: Learning optical flow with convolutional networks. In CVPR, 2015. 2

[15] H. K. Galoogahi, T. Simon, and S. Lucey. Correlation filters with limited boundaries. In CVPR, 2015. 2

[16] Q. Gan, Q. Guo, Z. Zhang, and K. Cho. First step toward model-free, anonymous object tracking with recurrent neural networks. arXiv:1511.06425, 2015. 2

[17] D. Gordon, A. Farhadi, and D. Fox. R3: Real-time recurrent regression networks for object tracking. arXiv:1705.06368, 2017. 2

[18] H. Grabner, C. Leistner, and H. Bischof. Semi-supervised on-line boosting for robust tracking. In ECCV, 2008. 2

[19] S. Hare, S. Golodetz, A. Saffari, V. Vineet, M.-M. Cheng, S. L. Hicks, and P. H. S. Torr. Struck: Structured output tracking with kernels. TPAMI, 2015. 2

[20] D. Held, S. Thrun, and S. Savarese. Learning to track at 100 fps with deep regression networks. In ECCV, 2016. 1, 2

[21] J. F. Henriques, R. Caseiro, P. Martins, and J. Batista. High-speed tracking with kernelized correlation filters. TPAMI, 2015. 1, 2

[22] S. Hochreiter, A. S. Younger, and P. R. Conwell. Learning to learn using gradient descents. ICANN, 2001. 2

[23] Z. Hong, Z. Chen, C. Wang, X. Mei, D. Prokhorov, and D. Tao. Multi-store tracker (muster): a cognitive psychological approach to object tracking. In CVPR, 2015. 2

[24] M. Jaderberg, K. Simonyan, A. Zisserman, and K. Kavukcuoglu. Spatial transformer networks. In CVPR, 2015. 2

[25] S. E. Kahou, V. Michalski, and R. Memisevic. Ratm: Recurrent attentive tracking model. arXiv:1510.08660, 2015. 2
[26] Z. Kalal, K. Mikolajczyk, and J. Matas. Tracking-learning-detection. *TPAMI*, 2010. 1, 2
[27] D. P. Kingma and J. L. Ba. Adam: A method for stochastic optimization. 2015. 4
[28] M. Kristan, A. Leonardis, J. Matas, M. Felsberg, and et al. The visual object tracking vot2015 challenge results. 2015. 6
[29] M. Kristan, A. Leonardis, J. Matas, M. Felsberg, and et al. The visual object tracking vot2016 challenge results. 2016. 1, 6
[30] H. Li, Y. Li, and F. Porikli. Deeptrack: Learning discriminative feature representations by convolutional neural networks for visual tracking. In *BMVC*, 2014. 2
[31] K. Li and J. Malik. Learning to optimize. In *ICLR*, 2017. 2
[32] Z. Li, F. Zhou, F. Chen, and H. Li. Meta-sgd: Learning to learn quickly for few shot learning. *arXiv:1707.09835*, 2017. 2, 4
[33] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg. Ssd: Single shot multibox detector. In *ECCV*, 2016. 5
[34] C. Ma, J.-B. Huang, X. Yang, and M.-H. Yang. Hierarchical convolutional features for visual tracking. In *ICCV*, 2015. 1, 2
[35] C. Ma, X. Yang, C. Zhang, and M.-H. Yang. Long-term correlation tracking. In *CVPR*, 2015. 2
[36] D. Maclaurin, D. Duvenaud, and R. P. Adams. Gradient-based hyperparameter optimization through reversible learning. In *ICML*, 2015. 4
[37] L. Metz, B. Poole, D. Pfau, and J. Sohl-Dickstein. Unrolled generative adversarial networks. In *ICLR*, 2017. 4
[38] M. Mueller, N. Smith, and B. Ghanem. Context-aware correlation filter tracking. In *CVPR*, 2017. 2
[39] H. Nam and B. Han. Learning multi-domain convolutional neural networks for visual tracking. In *CVPR*, 2016. 1, 2, 4, 6
[40] S. Ravi and H. Larochelle. Optimization as a model for few-shot learning. In *ICLR*, 2017. 2, 9
[41] S. Ren, K. He, R. Girshick, and J. Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *NIPS*, 2016. 5
[42] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *IJCV*, 2015. 2, 6
[43] A. Santoro, S. Bartunov, M. Botvinick, D. Wierstra, and T. Lillicrap. Meta-learning with memory-augmented neural networks. In *ICML*, 2016. 2, 5
[44] J. Schmidhuber. Evolutionary principles in self-referential learning. *Diploma thesis, Institut f. Informatik, Tech. Univ. Munich*, 1987. 2
[45] J. Schmidhuber. Learning to control fast-weight memories: an alternative to dynamic recurrent networks. *Neural Computation*, 1992. 2
[46] Y. Song, C. Ma, L. Gong, J. Zhang, R. Lau, and M.-H. Yang. Crest: Convolutional residual learning for visual tracking. In *ICCV*, 2017. 1, 2, 4, 5, 6
[47] J. Supancic and D. Ramanan. Tracking as online decision-making: Learning a policy from streaming videos with reinforcement learning. In *ICCV*, 2017. 9
[48] R. Tao, E. Gavves, and A. W. M. Smeulders. Siamese instance search for tracking. In *CVPR*, 2016. 1, 2
[49] S. Thrun and L. Pratt. Learning to learn: introduction and overview. *Springer*, 1998. 2
[50] J. Valmadre, L. Bertinetto, J. F. Henriques, A. Vedaldi, and P. H. S. Torr. End-to-end learning for correlation filter based tracking. In *CVPR*, 2017. 2
[51] Y.-X. Wang and M. Hebert. Learning to learn: Model regression networks for easy small sample learning. In *ECCV*, 2016. 2
[52] O. Wichrowska, N. Maheswaranathan, M. W. Hoffman, S. G. Colmenarejo, M. Denil, N. de Freitas, and J. Sohl-Dickstein. Learned optimizers that scale and generalize. In *ICML*, 2017. 2
[53] Y. Wu, J. Lim, and M.-H. Yang. Object tracking benchmark. *TPAMI*, 2015. 1, 6
[54] T. Yang and A. B. Chan. Recurrent filter learning for visual tracking. In *ICCV*, 2017. 2, 9
[55] K. Zhang, L. Zhang, Q. Liu, and D. Z. andMing Hsuan Yang. Fast visual tracking via dense spatio-temporal context learning. In *ECCV*, 2014. 2
Appendix

A. Comparisons of individual sequences on OTB-2015

Figure 7. Success rates (with IoU metric) of 100 sequences in OTB-2015 dataset for pyMDNet-01 vs MetaSDNet-01

Figure 8. Success rates (with IoU metric) of 100 sequences in OTB-2015 dataset for pyMDNet-05 vs MetaSDNet-01

Figure 9. Success rates (with IoU metric) of 100 sequences in OTB-2015 dataset for pyMDNet-15 vs MetaSDNet-01

Figure 10. Success rates (with IoU metric) of 100 sequences in OTB-2015 dataset for pyMDNet-30 vs MetaSDNet-01
B. More visualizations of response maps in CREST trackers

Figure 11. More visualizations of response maps in CREST: Left three columns represents the image patch at initial frame, response map with meta-learned initial correlation filters $\theta_0$, response map after updating 1 iteration with learned $\alpha$, respectively. The rest of seven columns on the right shows response maps after updating the model up to 15 iterations.