ROAM: Recurrently Optimizing Tracking Model

Tianyu Yang[1,2]   Pengfei Xu[3]   Runbo Hu[3]   Hua Chai[3]   Antoni B. Chan[2]

[1] Tencent AI Lab   [2] City University of Hong Kong   [3] Didi Chuxing

Abstract

In this paper, we design a tracking model consisting of response generation and bounding box regression, where the first component produces a heat map to indicate the presence of the object at different positions and the second part regresses the relative bounding box shifts to anchors mounted on sliding-window locations. Thanks to the resizable convolutional filters used in both components to adapt to the shape changes of objects, our tracking model does not need to enumerate different sized anchors, thus saving model parameters. To effectively adapt the model to appearance variations, we propose to offline train a recurrent neural optimizer to update tracking model in a meta-learning setting, which can converge the model in a few gradient steps. This improves the convergence speed of updating the tracking model while achieving better performance. We extensively evaluate our trackers, ROAM and ROAM++, on the OTB, VOT, LaSOT, GOT-10K and TrackingNet benchmark and our methods perform favorably against state-of-the-art algorithms.

1. Introduction

Generic visual object tracking is the task of estimating the bounding box of a target in a video sequence given only its initial position. Typically, the preliminary model learned from the first frame needs to be updated continuously to adapt to the target's appearance variations caused by rotation, illumination, occlusion, deformation, etc. However, it is challenging to optimize the initial learned model efficiently and effectively as tracking proceeds. Training samples for model updating are usually collected based on estimated bounding boxes, which could be inaccurate. Those small errors will accumulate over time, gradually resulting in model degradation.

To avoid model updating, which may introduce unreliable training samples that ruin the model, several approaches [4, 43] investigate tracking by only comparing the first frame with the subsequent frames, using a similarity function based on a learned discriminant and invariant deep Siamese feature embedding. However, training such a deep representation is difficult due to drastic appearance variations that commonly emerge in long-term tracking. Other

methods either update the model via an exponential moving average of templates [16, 44], which marginally improves the performance, or optimize the model with hand-designed SGD methods [33, 41], which needs numerous iterations to converge thus preventing real-time speed. Limiting the number of SGD iterations can allow near real-time speeds, but at the expense of poor quality model updates due to the loss function not being optimized sufficiently.

In recent years, much effort has been done on localizing the object using robust online learned classifier, while few attention is paid on designing accurate bounding box estimation. Most trackers simply resort to multi-scale search by assuming that the object aspect ratio does not change during tracking, which is often violated in real world. Recently, SiamRPN [23] borrows the idea of region proposal networks [37] in object detection to decompose tracking task into two branches: 1) classifying the target from the background, and 2) regressing the accurate bounding box based with reference to anchor boxes mounted on different positions. As is shown on the VOT benchmarks [19, 20], SiamRPN achieves higher precision on bounding box estimation but suffers lower robustness compared with state-of-the-art methods [8, 9, 26] due to no online model updating. Furthermore, SiamRPN mounts anchors with different aspect ratios on every spatial location of the feature map to handle possible shape changes, which is redundant in both computation and storage.

In this paper, we propose a tracking framework which is composed of two modules: response generation and bounding box regression, where the first component produces a response map to indicate the possibility of covering the object for anchor boxes mounted on sliding-window positions, and the second part predicts bounding box shifts from the anchors to get refined rectangles. Instead of enumerating different aspect ratios of anchors as in SiamRPN, we propose to use only one sized anchor for each position and adapt it to shape changes by resizing its corresponding convolutional filter using bilinear interpolation, which saves model parameters and computing time. To effectively adapt the tracking model to appearance changes during tracking, we propose a recurrent model optimization method to learn a more effective gradient descent that converges the model update in 1-2 steps, and generalizes better to future frames. The key idea is to train a neural optimizer that can con-

verge the tracking model to a good solution in a few gradient steps. During the training phase, the tracking model is first updated using the neural optimizer, and then it is applied on future frames to obtain an error signal for minimization. Under this particular setting, the resulting optimizer converges the tracking classifier significant faster than SGD-based optimizers, especially for learning the initial tracking model. In summary, our contributions are:

- We propose a tracking model consisting of resizable response generator and bounding box regressor, where only one sized anchor is used on each spatial position and its corresponding convolutional filter could be adapted to shape variations by bilinear interpolation.
- We propose a recurrent neural optimizer, which is trained in a meta-learning setting, that recurrently updates the tracking model with faster convergence.
- We conduct comprehensive experiments on large scale datasets including OTB, VOT, LaSOT, GOT10k and TrackingNet, and our trackers achieve favorable performance compared with the state-of-the-art.

2. Related Work

Visual Tracking Predicting a heat map to indicate the position of object is commonly used in visual tracking community [3, 4, 4, 9, 13, 16, 45]. Among them, SiamFC [4] is one of the most popular methods due to its fast speed and good performance. However, most response generation based trackers, including SiamFC, estimate the bounding box via simple multi-scale search mechanism, which is unable to handle aspect ratio changes. To addresses this issue, recent SiamRPN [23] and its extensions [11, 22, 52] propose to train a bounding box regressor as in object detection [37], showing impressive performance. Different from these algorithms which enumerates a set of predefined anchors with different aspect ratios on each spatial position, we adopt a resizable anchor to adapt the shape variation of object, which saves model parameters and computing time.

Online model updating is another important module that SiamFC lacks. Recent works improve SiamFC [4] by introducing various model updating strategies, including recurrent generation of target template filters through a convolutional LSTM [48], a dynamic memory network [49, 50], where object information is written into and read from an addressable external memory, and distractor-aware incremental learning [53], which make use of hard-negative templates around the target to suppress distractors. It should be noted that all these algorithms essentially achieve model updating by linearly interpolating old target templates with the newly generated one, in which the major difference is how to control the weights when combining them. This is far from optimal compared with optimization methods using gradient decent, which minimize the tracking loss directly to adapt to new target appearances. Instead of using a Siamese network to build the convolutional filter, other methods [26, 34, 41] generate the filter by performing gra-

dient decent on the first frame, which could be continuously optimized during subsequent frames. In particular, [34] proposes to train the *initial* tracking model in a meta-learning setting, which shows promising results. However, it still uses traditional SGD to optimize the tracking model during the subsequent frames, which is not effective to adapt to new appearance and slow in updating the model. In contrast to these trackers, our off-line learned recurrent neural optimizer applies meta-learning on both the initial model and model updates, which allows model initialization and updating in only one or two gradient steps, resulting in much faster runtime speed, and better accuracy.

Learning to Learn. Learning to learn or meta-learning has a long history [2, 32, 38]. With the recent successes of applying meta-learning on few-shot classification [31, 36] and reinforcement learning [12, 39], it has regained attention. The pioneering work [1] designs an off-line learned optimizer using gradient decent and shows promising performance compared with traditional optimization methods. However, it does not generalize well for large numbers of descent step. To mitigate this problem, [28] proposes several training techniques, including parameters scaling and combination with convex functions to coordinate the learning process of the optimizer. [46] also addresses this issue by designing a hierarchical RNN architecture with dynamically adapted input and output scaling. In contrast to other works that output an increment for each parameter update, which is prone to overfitting due to different gradient scales, we instead associate an adaptive learning rate produced by a recurrent neural network with the computed gradient for fast convergence of the model update.

3. Proposed Algorithm

Our tracker consists of two main modules: 1) a tracking model that is resizable to adapt to shape changes; and 2) a neural optimizer that is in charge of model updating. The tracking model contains two branches where the response generation branch determines the presence of target by predicting a confidence score map and the bounding box regression branch estimates the precise box of the target by regressing coordinate shifts from the box anchors mounted on the sliding-window positions. The offline learned neural optimizer is trained using a meta-learning framework to online update the tracking model in order to adapt to appearance variations. Note both response generation and bounding box regression are built on the feature map computed from the backbone CNN network. The whole framework is briefly illustrated on Fig. 1

3.1. Resizable Tracking Model

Trackers like correlation filter [16] and MetaTracker [34] initialize a convolutional filter based on the size of object in the first frame and keep its size fixed during subsequent frames. This setting is based on the assumption that the as-
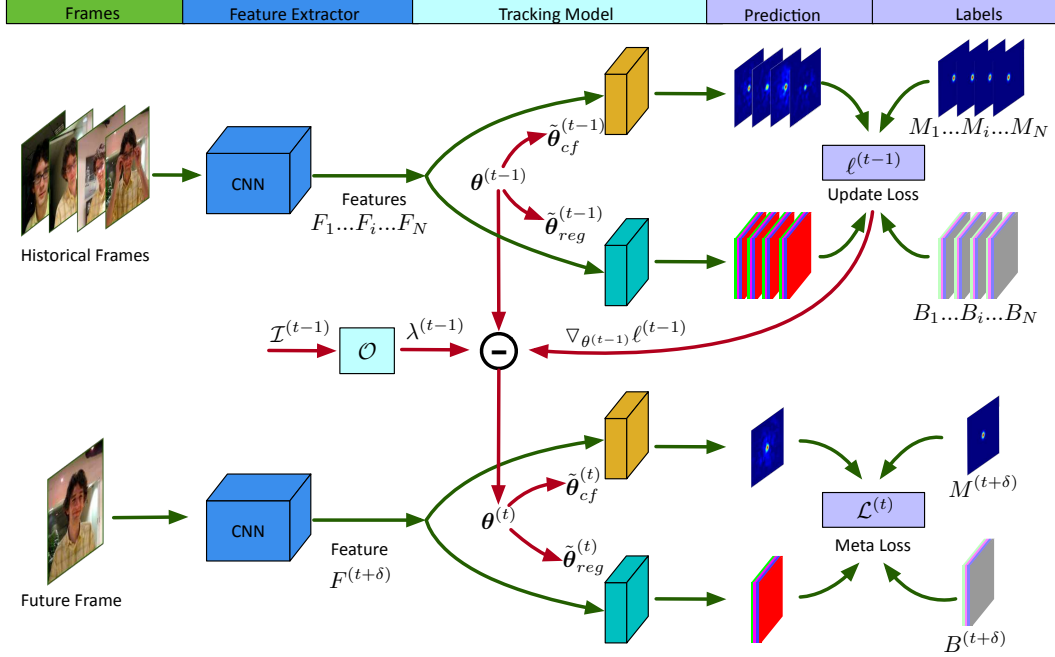
Figure 1: Pipeline of ROAM++. Given a mini-batch of training patches, which are cropped based on the predicted object boxes, deep features are extracted by *Feature Extractor*. The fixed-size *Tracking Model* $\theta^{(t-1)}$ is warped to the current target size yielding the warped tracking model $\tilde{\theta}^{(t-1)}$, as in (2, 3). The response map and bounding boxes are then predicted for each sample using $\tilde{\theta}^{(t-1)}$, from which the update loss $\ell^{(t-1)}$ and its gradient $\nabla_{\theta^{(t-1)}}\ell^{(t-1)}$ are computed using ground truth labels. Next, the element-wise stack $\mathcal{I}^{(t-1)}$ consisting of previous learning rates, current parameters, current update loss and its gradient are input into a coordinate-wise LSTM $\mathcal{O}$ to generate the adaptive learning rate $\lambda^{(t-1)}$ as in (11). The model is then updated using one gradient descent step (denoted by $\ominus$) as in (9). Finally, we apply the updated model $\theta^{(t)}$ on a randomly selected future frame to get a meta loss for minimization as in (13).

pect ratio of object does not change during tracking, which is however often violated. Therefore, dynamically adapting the convolutional filter to the object shape variations is desirable, which means the number of filter parameters may vary between frames in the video and among different sequences. However, this complicates the design of neural optimizer when using separate learning rates for each filter parameter. To simplify the meta-learning framework and to better allow for per-parameter learning rates, we define fixed-shape convolutional filters, which are warped to the desired target size using bilinear interpolation before convolving with the feature map. In subsequent frames, the recurrent optimizer updates the fixed-shape tracking model. Note that MetaTracker [34] also resizes filters to the object size for model initialization, However, instead of dynamically adapting the convolutional filters to the object size for subsequent frames, MetaTracker keep the same shape as the initial filter during the following frames

Specifically, tracking model θ contains two parts, i.e. correlation filter θ_{cf} and bounding box regression filter θ_{reg}. They are both warped to adapt to the shape variation of target,

$$\theta = [\theta_{cf}, \theta_{reg}], \tag{1}$$

$$\tilde{\theta}_{cf} = \mathcal{W}(\theta_{cf}, \phi), \tag{2}$$

$$\tilde{\theta}_{reg} = \mathcal{W}(\theta_{reg}, \phi), \tag{3}$$

where $\mathcal{W}$ resizes the convolutional filter to size $\phi = (f_r, f_c)$ using bilinear interpolation. The filter size is computed from the width and height (w, h) of the object in the previous image patch (and for symmetry the filter size is odd),

$$f_r = \lceil \frac{\rho h}{c} \rceil - \lceil \frac{\rho h}{c} \rceil \bmod 2 + 1, \tag{4}$$

$$f_c = \lceil \frac{\rho w}{c} \rceil - \lceil \frac{\rho w}{c} \rceil \bmod 2 + 1 \tag{5}$$

where ρ is the scale factor to enlarge the filter size to cover some context information, and c is the stride of feature map, and $\lceil \; \rceil$ means ceiling. Because of the resizable filters, there is no need to enumerate different aspect ratios and scales of anchor boxes when performing bounding box regression. We only use one sized anchor on each spatial location whose size is corresponding to the shape of regression filter,

$$(a_w, a_h) = (f_c, f_r)/\rho, \tag{6}$$

This saves regression filter parameters and achieves faster speed. Note that we update the filter size and its corresponding anchor box every τ frames, i.e. just before every model updating, during both offline training and testing/tracking phases. Through this modification, we are able to initialize the tracking model with $\theta^{(0)}$ and recurrently optimize it in subsequent frames without worrying about the shape changes of the tracked object.

3.2. Recurrent Model Optimization

Traditional optimization methods have the problem of slow convergence due to small learning rates and limited training samples, while simply increasing the learning rate has the risk of the training loss wandering wildly. Instead, we design a recurrent neural optimizer, which is trained to converge the model to a good solution in a few gradient steps[1], to update the tracking model. Our key idea is based on the assumption that *the best optimizer should be able to update the model to generalize well on future frames*. During the *offline* training stage, we perform a one step gradient update on the tracking model using our recurrent neural optimizer and then minimize its loss on future frames. Once the *offline* learning stage is finished, we use this learned neural optimizer to recurrently update the tracking model to adapt to appearance variations. The optimizer trained in this way will be able to quickly converge the model update to generalize well for future frame tracking.

We denote the response generation network as $\mathcal{G}(F; \boldsymbol{\theta}_{cf}, \phi)$ and the bounding box regression network as $\mathcal{R}(F; \boldsymbol{\theta}_{reg}, \phi)$, where F is the feature map input and $\boldsymbol{\theta}$ are the parameters. The tracking loss consists of two parts: response loss and regression loss,

$$L(F, M, B; \boldsymbol{\theta}, \phi) = \|\mathcal{G}(F; \boldsymbol{\theta}_{cf}, \phi) - M\|^2 + \\ \|\mathcal{R}(F; \boldsymbol{\theta}_{reg}, \phi) - B\|_s \quad (7)$$

where the first term is the L2 loss and the second term is the smooth L1 loss [37], and B represents the ground truth box. Note we adopt parameterization of bounding box coordinates as in [37]. M is the corresponding label map which is built using a 2D Gaussian function and the ground-truth object location (x_0, y_0) and size (w, h),

$$M(x, y) = \exp\left(-\alpha \left(\frac{(x-x_0)^2}{\sigma_x^2} + \frac{(y-y_0)^2}{\sigma_y^2}\right)\right) \quad (8)$$

where $(\sigma_x, \sigma_y) = (w/c, h/c)$, and α controls the shape of the response map. Note that we use past predictions as pseudo-labels when performing model updating during testing. We only use the ground truth during offline training.

A typical tracking process updates the tracking model using historical training examples and then tests this updated model on the following frames until the next update. We simulate this scenario in a meta-learning paradigm by recurrently optimizing the tracking model, and then testing it on a future frame. Specifically, the tracking network is updated by

$$\boldsymbol{\theta}^{(t)} = \boldsymbol{\theta}^{(t-1)} - \boldsymbol{\lambda}^{(t-1)} \odot \nabla_{\boldsymbol{\theta}^{(t-1)}} \ell^{(t-1)}, \quad (9)$$

where $\boldsymbol{\lambda}^{(t-1)}$ is a fully element-wise learning rate that has the same dimension as the tracking model parameters

$\boldsymbol{\theta}^{(t-1)}$, and $\odot$ is element-wise multiplication. The learning rate is recurrently generated using an LSTM with input consisting of the previous learning rate $\boldsymbol{\lambda}^{(t-2)}$, the current parameters $\boldsymbol{\theta}^{(t-1)}$, the current update loss $\ell^{(t-1)}$ and its gradient $\nabla_{\boldsymbol{\theta}^{(t-1)}} \ell^{(t-1)}$,

$$\mathcal{I}^{(t-1)} = [\boldsymbol{\lambda}^{(t-2)}, \nabla_{\boldsymbol{\theta}^{(t-1)}} \ell^{(t-1)}, \boldsymbol{\theta}^{(t-1)}, \ell^{(t-1)}], \quad (10)$$

$$\boldsymbol{\lambda}^{(t-1)} = \sigma(\mathcal{O}(\mathcal{I}^{(t-1)}; \boldsymbol{\omega})), \quad (11)$$

where $\mathcal{O}(\cdot; \boldsymbol{\omega})$ is a coordinate-wise LSTM [1] parameterized by $\boldsymbol{\omega}$, which shares the parameters across all dimensions of input, and σ is the sigmoid function to bound the predicted learning rate. The LSTM input $\mathcal{I}^{(t-1)}$ is formed by element-wise stacking the 4 sub-inputs along a new axis[2]. The current *update loss* $\ell^{(t-1)}$ is computed from a mini-batch of n updating samples,

$$\ell^{(t-1)} = \frac{1}{n} \sum_{j=1}^{n} L(F_j, M_j, B_j; \boldsymbol{\theta}^{(t-1)}, \phi^{(t-1)}), \quad (12)$$

where the updating samples (F_j, M_j, B_j) are collected from the previous τ frames, where τ is the frame interval between model updates during online tracking. Finally, we test the newly updated model $\boldsymbol{\theta}^{(t)}$ on a randomly selected future frame[3] and obtain the *meta loss*,

$$\mathcal{L}^{(t)} = L(F^{(t+\delta)}, M^{(t+\delta)}, B^{(t+\delta)}; \boldsymbol{\theta}^{(t)}, \phi^{(t-1)}), \quad (13)$$

where δ is randomly selected within $[0, \tau - 1]$.

During offline training stage, we perform the aforementioned procedure on a mini-batch of videos and get an averaged meta loss for optimizing the neural optimizer,

$$\bar{\mathcal{L}} = \frac{1}{NT} \sum_{i=1}^{N} \sum_{t=1}^{T} \mathcal{L}_{\mathcal{V}_i}^{(t)}, \quad (14)$$

where N is the batch size and T is the number of model updates, and $\mathcal{V}_i \sim p(\mathcal{V})$ is a video clip sampled from the training set. It should be noted that the initial tracking model parameter $\boldsymbol{\theta}^{(0)}$ and initial learning rate $\boldsymbol{\lambda}^{(0)}$ are also trainable variables, which are jointly learned with the neural optimizer $\mathcal{O}$. By minimizing the averaged meta loss $\bar{\mathcal{L}}$, we aim to train a neural optimizer that can update the tracking model to generalize well on subsequent frames, as well as to learn a beneficial initialization of the tracking model, which is broadly applicable to different tasks (i.e. videos). The overall training process is detailed in Algorithm 1.

---

[1] We only use one gradient step in our experiment, while considering multiple steps is straightforward.

[2] We therefore get an $|\boldsymbol{\theta}| \times 4$ matrix, where $|\boldsymbol{\theta}|$ is the number of parameters in $\boldsymbol{\theta}$. Note that the current update loss $\ell^{(t-1)}$ is broadcasted to have compatible shape with other vectors. To better understand this process, we can treat the input of LSTM as a mini-batch of vectors where $|\boldsymbol{\theta}|$ is the batch size and 4 is the dimension of the input vector.

[3] We found that using more than one future frame does not improve performance but costs more time during the off-line training phase.

Algorithm 1 Offline training of our framework

Input:
   $p(\mathcal{V})$: distribution over training videos.
Output:
   $\boldsymbol{\theta}^{(0)}, \boldsymbol{\lambda}^{(0)}$: initial tracking model and learning rates.
   $\boldsymbol{\omega}$: recurrent neural optimizer.
1: Initialize all network parameters.
2: **while** not done **do**
3:   Draw a mini-batch of videos: $\mathcal{V}_i \sim p(\mathcal{V})$
4:   **for** all $\mathcal{V}_i$ **do**
5:     Compute $\boldsymbol{\theta}^{(1)} \leftarrow \boldsymbol{\theta}^{(0)} - \boldsymbol{\lambda}^{(0)} \odot \nabla_{\boldsymbol{\theta}^{(0)}} \ell^{(0)}$ in (9).
6:     Compute meta loss $\mathcal{L}^{(1)}$ using (13).
7:     **for** $t = \{1 + \tau, 1 + 2\tau, \cdots, 1 + (T-1)\tau\}$ **do**
8:       Compute adaptive learning rate $\boldsymbol{\lambda}^{(t-1)}$ using neural optimizer $\mathcal{O}$ as in (11).
9:       Compute updated model $\boldsymbol{\theta}^{(t)}$ using (9).
10:       Compute meta loss $\mathcal{L}^{(t)}$ using (13).
11:     **end for**
12:   **end for**
13:   Compute averaged meta loss $\bar{\mathcal{L}}$ using (14).
14:   Update $\boldsymbol{\theta}^{(0)}, \boldsymbol{\lambda}^{(0)}, \boldsymbol{\omega}$ by computing gradient of $\bar{\mathcal{L}}$.
15: **end while**

3.3. Random Filter Scaling

Neural optimizers have difficulty to generalize well on new tasks due to overfitting as discussed in [1, 28, 46]. By analyzing the learned behavior of the neural optimizer, we found that our preliminary trained optimizer will predict similar learning rates (see Fig. 2 left). We attribute this to overfitting to network inputs with similar magnitude scales. The following simple example illustrates the overfitting problem. Suppose the objective function[4] that the neural optimizer minimizes is $g(\boldsymbol{\theta}) = \|\boldsymbol{x}\boldsymbol{\theta} - y\|^2$. The optimal element-wise learning rate is $1/2\boldsymbol{x}^2$ since we can achieve the lowest loss of 0 in one gradient descent step $\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - 1/2\boldsymbol{x}^2 \nabla g(\boldsymbol{\theta}^{(t)}) = y/\boldsymbol{x}$. Note that the optimal learning rate depends on the network input $\boldsymbol{x}$, and thus the learned neural optimizer is prone to overfitting if it does not see enough magnitude variations of $\boldsymbol{x}$. To address this problem, we multiply the tracking model $\boldsymbol{\theta}$ with a randomly sampled vector $\boldsymbol{\epsilon}$, which has the same dimension as $\boldsymbol{\theta}$ [28] during each iteration of offline training,

$$\boldsymbol{\epsilon} \sim \exp(\mathcal{U}(-\kappa, \kappa)), \quad \boldsymbol{\theta}_{\boldsymbol{\epsilon}} = \boldsymbol{\epsilon} \odot \boldsymbol{\theta}, \qquad (15)$$

where $\mathcal{U}(-\kappa, \kappa)$ is a uniform distribution in the interval $[-\kappa, \kappa]$, and κ is the range factor to control the scale extent. The objective function is then modified as $g_{\boldsymbol{\epsilon}}(\boldsymbol{\theta}) = g(\frac{\boldsymbol{\theta}}{\boldsymbol{\epsilon}})$. In this way, the network input $\boldsymbol{x}$ is indirectly scaled without modifying the training samples $(\boldsymbol{x}, y)$ in practice. Thus, the learned neural optimizer is forced to predict adaptive learning rates for inputs with different magnitudes (see Fig. 2

[4]Our actual loss function in (7) includes an L2 loss and smooth L1 loss; for simplicity here we consider a simple linear model with a L2 loss.
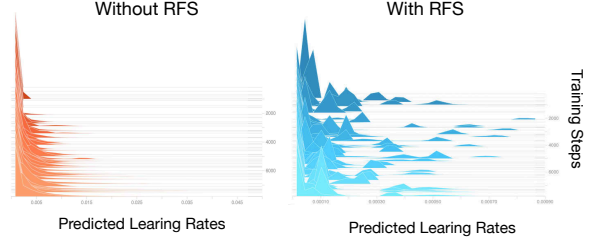

Figure 2: Histogram of predicted learning rates during offline training

right), rather than to produce similar learning rates for similar scaled inputs, which improves its generalization ability.

4. Online Tracking via Proposed Framework

The offline training produces a neural optimizer $\mathcal{O}$, initial tracking model $\boldsymbol{\theta}^{(0)}$ and learning rate $\boldsymbol{\lambda}^{(0)}$, which we then use to perform online tracking. The overall process is similar to offline training except that we do not compute the meta loss or its gradient.

Model Initialization. Given the first frame, the initial image patch is cropped and centered on the provided groundtruth bounding box. We then augment the initial image patch to 9 samples by stretching the object to different aspect ratios and scales. Specifically, the target is stretched by $[swr, sh/r]$, where (w, h) are the initial object width and height, and (s, r) are scale and aspect ratio factors. Then we use these examples, as well as the offline learned $\boldsymbol{\theta^{(0)}}$ and $\boldsymbol{\lambda^{(0)}}$, to perform one-step gradient update to build an initial model $\boldsymbol{\theta^{(1)}}$ as in (9).

Bounding Box Estimation. We estimate the object bounding box by first finding the presence of target through the response generation, and then predict accurate box by bounding box regression. We employ the penalty strategy used in [23] on the generated response to suppress estimated boxes with large changes in scale and aspect ratio. In addition, we also multiply the response map by a Gaussian-like motion map to suppress large movement. The bounding box computed by the anchor that corresponds to the maximum score of response map is the final prediction. To smoothen the results, we linearly interpolate this estimated object size with the previous one. We denote our neural-optimized tracking model with bounding box regression as **ROAM++**. We also design a baseline variant of our tracker, which uses multi-scale search to the estimate object box instead of bounding box regression, and denote it as **ROAM**.

Model Updating. We update the model every τ frame. Although offline training uses the previous τ frames to perform a one-step gradient update of the model, in practice, we find that using more than one step could further improve the performance during tracking (see Sec. 6.2). Therefore, we adopt two-step gradient update using the previous 2τ frames in our experiments.

5. Implementation Details

Patch Cropping. Given a bounding box of the object (x_0, y_0, w, h), the ROI of the image patch has the same center (x_0, y_0) and takes a larger size $S = S_w = S_h = \gamma \sqrt{wh}$, where γ is the ROI scale factor. Then, the ROI is resized to a fixed size $\mathcal{S} \times \mathcal{S}$ for batch processing.

Network Structure. We use the first 12 convolution layers of the pretrained VGG-16 [40] as the feature extractor. The top max-pooling layers are removed to increase the spatial resolution of the feature map. Both the response generation network $\mathcal{C}$ and bounding box regression network consists of two convolutional layers with a dimension reduction layer of $512 \times 64 \times 1 \times 1$ (in-channels, out-channels, height, width) as the first layer, and either a correlation layer of $64 \times 1 \times 21 \times 21$ or a regression layer of $64 \times 4 \times 21 \times 21$ as the second layer respectively. We use two stacked LSTM layers with 20 hidden units for the neural optimizer $\mathcal{O}$.

The ROI scale factor is $\gamma = 5$ and the search size is $\mathcal{S} = 281$. The scale factor of filter size is $\rho = 1.5$. The response generation uses $\alpha = 20$ and the feature stride of the CNN feature extractor is $c = 4$. The scale and aspect ratio factors s, r used for initial image patch augmentation are selected from $\{0.8, 1, 1.2\}$, generating a combination of 9 pairs of (s, r). The range factors used in RFS are $\kappa_{cf} = 1.6, \kappa_{reg} = 1.3$ [5].

Training Details. We use ADAM [18] optimization with a mini-batch of 16 video clips of length 31 on 4 GPUs (4 videos per GPU) to train our framework. We use the training splits of ImageNet VID [21], TrackingNet [30], LaSOT [10], GOT10k [17], ImageNet DET [21] and COCO [25] for training. During training, we randomly extract a continuous sequence clip for video datasets, and repeat the same still image to form a video clip for image datasets. Note we randomly augment all frames in a training clip by slightly stretching and scaling the images. We use a learning rate of 1e-6 for the initial regression parameters $\boldsymbol{\theta}^{(0)}$ and the initial learning rate $\boldsymbol{\lambda}^{(0)}$. For the recurrent neural optimizer $\mathcal{O}$, we use a learning rate of 1e-3. Both learning rates are multiplied by 0.5 every 5 epochs. We implement our tracker in Python with the PyTorch toolbox [35], and conduct the experiments on a computer with an NVIDIA RTX2080 GPU and Intel(R) Core(TM) i9 CPU @ 3.6 GHz. Our tracker ROAM and ROAM++ run at 13 FPS and 20 FPS respectively. (See supplementary for detailed speed comparison)

6. Experiments

We evaluate our trackers on six benchmarks: OTB-2015 [47], VOT-2016 [19], VOT-2017 [20], LaSOT [10], GOT-10k [17] and TrackingNet [30].
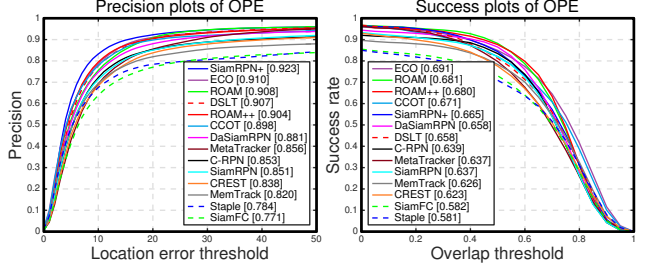
Figure 3: Precision and success plot on OTB-2015.

	VOT-2016			VOT-2017		
	EAO(↑)	A(↑)	R(↓)	EAO(↑)	A(↑)	R(↓)
ROAM++	0.441	0.599	0.174	0.380	0.543	0.195
ROAM	0.384	0.556	0.183	0.331	0.505	0.226
MetaTracker	0.317	0.519	-	-	-	-
DaSiamRPN	0.411	0.61	0.22	0.326	0.56	0.34
SiamRPN+	0.37	0.58	0.24	0.30	0.52	0.41
C-RPN	0.363	0.594	-	0.289	-	-
SiamRPN	0.344	0.56	0.26	0.244	0.49	0.46
ECO	0.375	0.55	0.20	0.280	0.48	0.27
DSLT	0.343	0.545	0.219	-	-	-
CCOT	0.331	0.54	0.24	0.267	0.49	0.32
Staple	0.295	0.54	0.38	0.169	0.52	0.69
CREST	0.283	0.51	0.25	-	-	-
MemTrack	0.272	0.531	0.373	0.248	0.524	0.357
SiamFC	0.235	0.532	0.461	0.188	0.502	0.585

Table 1: Results on VOT-2016/2017. The evaluation metrics are expected average overlap (EAO), accuracy value (A), robustness value (R). The top 3 performing trackers are colored with red, green and blue respectively.

6.1. Comparison Results with State-of-the-art

We compare our ROAM and ROAM++ with recent response generation based trackers including MetaTracker [34], DSLT [26], MemTrack [49], CREST [41], SiamFC [4], CCOT [9], ECO [8], Staple [3], as well as recent state-of-the-art trackers including SiamRPN[23], DaSiamRPN [53], SiamRPN+ [52] and C-RPN [11] on both OTB and VOT datasets. For the methods using SGD updates, the number of SGD steps followed their implementations.

OTB. Fig. 3 presents the experiment results on OTB-2015 dataset, which contains 100 sequences with 11 annotated video attributes. Both our ROAM and ROAM++ achieve similar AUC compared with top performer ECO and outperform all other trackers. Specifically, our ROAM and ROAM++ surpass MetaTracker [34], which is the baseline for meta learning trackers that uses traditional optimization method for model updating, by 6.9% and 6.7% on the success plot respectively, demonstrating the effectiveness of the proposed recurrent model optimization algorithm and resizable bounding box regression. In addition, both our ROAM and ROAM++ outperform the very recent meta learning based tracker MLT [7] by a large margin (ROAM/ROAM++: 0.681/0.680 vs MLT: 0.611) under the AUC metric on OTB-2015.

VOT. Table 1 shows the comparison performance on VOT-2016 and VOT-2017 datasets. Our ROAM++ achieves the best EAO on both VOT-2016 and VOT-2017. Specially,
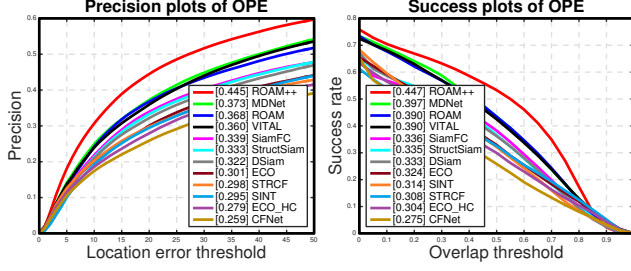
Figure 4: Precision and success plot on LaSOT test dataset

	MDNet [33]	CF2 [29]	ECO [8]	CCOT [9]	GOTURN [15]	SiamFC [4]	SiamFCv2 [44]	**ROAM**	**ROAM++**
AO($\uparrow$)	0.299	0.315	0.316	0.325	0.342	0.348	0.374	0.436	0.465
SR$_{0.5}$($\uparrow$)	0.303	0.297	0.309	0.328	0.372	0.353	0.404	0.466	0.532
SR$_{0.75}$($\uparrow$)	0.099	0.088	0.111	0.107	0.124	0.098	0.144	0.164	0.236

Table 2: Results on GOT-10k. The evaluation metrics include average overlap (AO), success rate at 0.5 overlap threshold. (SR$_{0.5}$), success rate at 0.75 overlap threshold. (SR$_{0.75}$). The top three performing trackers are colored with red, green and blue respectively.

both our ROAM++ and ROAM shows superior performance on robustness value compared with RPN based trackers which have no model updating, demonstrating the effectiveness of our recurrent model optimization scheme. In addition, our ROAM++ and ROAM outperform the baseline MetaTracker [34] by 39.2% and 21.1% on EAO of VOT-2016, respectively.

LaSOT. LaSOT [10] is a recently proposed large-scale tracking dataset. We evaluate our ROAM against the top-10 performing trackers of the benchmark [10], including MDNet [33], VITAL [42], SiamFC [4], StructSiam [51], DSiam [14], SINT [14], ECO [8], STRCF [24], ECO_HC [8] and CFNet [44], on the testing split which consists of 280 videos. Fig. 4 presents the comparison results of precision plot and success plot on LaSOT testset. Our ROAM++ achieves the best result compared with state-of-the-art trackers on the benchmark, outperforming the second best MDNet with an improvement of 19.3% and 12.6% on precision plot and success plot respectively.

GOT-10k. GOT-10k [17] is large highly-diverse dataset for object tracking that is also proposed recently. There is no overlap in object categories between the training set and test set, which follows the *one-shot learning setting* [12]. Therefore, using external training data is strictly forbidden when testing trackers on their online server. Following their protocol, we train our ROAM by using only the training split of this dataset. Table 2 shows the detailed comparison results on GOT-10k test dataset. Both our ROAM++ and ROAM surpass other trackers with a large margin. In particular, our ROAM++ obtains an AO of 0.465, a SR$_{0.5}$ of 0.532 and a SR$_{0.75}$ of 0.236, outperforming SiamFCv2 with an improvement of 24.3%, 31.7% and 63.9% respectively.

TrackingNet. TrackingNet [30] provides more than 30K videos with around 14M dense bounding box annotations by filtering shorter video clips from Youtube-BB [5]. Table 3 presents the detailed comparison results on TrackingNet test dataset. Our ROAM++ surpasses other state-of-

	Staple [3]	CSRDCF [27]	ECOhc [8]	ECO [8]	SiamFC [44]	CFNet [44]	MDNet [33]	**ROAM**	**ROAM++**
AUC($\uparrow$)	0.528	0.534	0.541	0.554	0.571	0.578	0.606	0.620	0.670
Prec.($\uparrow$)	0.470	0.480	0.478	0.492	0.533	0.533	0.565	0.547	0.623
Norm. Prec.($\uparrow$)	0.603	0.622	0.608	0.618	0.663	0.654	0.705	0.695	0.754

Table 3: Results on TrackingNet. The evaluation metrics include area under curve (AUC) of success plot, Precision, Normalized Precison. The top three performing trackers are colored with red, green and blue respectively.
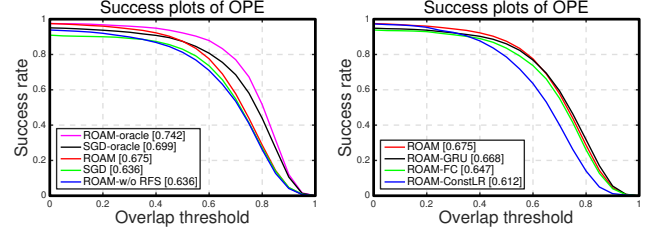

Figure 5: Ablation Study on OTB-2015 with different variants of ROAM.

the-art tracking algorithms on all three evaluation metrics. In detail, our ROAM++ obtains an improvement of 10.6%, 10.3% and 6.9% on AUC, precision and normalized precision respectively compared with top performing tracker MDNet on the benchmark.

6.2. Ablation Study

For a deeper analysis, we study our trackers from various aspects. Note that all these ablations are trained only on ImageNet VID dataset for simplicity.

Impact of Different Modules. To verify the effectiveness of different modules, we design four variants of our framework: 1) *SGD*: replacing recurrent neural optimizer with traditional SGD for model updating (using the same number of gradient steps as ROAM); 2) *ROAM-w/o RFS*: training a recurrent neural optimizer without RFS; 3) *SGD-Oracle*: using the ground-truth bounding boxes to build updating samples for SGD during the testing phase (using the same number of gradient steps as ROAM); 4) *ROAM-Oracle*: using the ground-truth bounding boxes to build updating samples for ROAM during the testing phase. The results are presented in Fig. 5 (Left). ROAM gains about 6% improvement on AUC compared with the baseline SGD, demonstrating the effectiveness of our recurrent model optimization method on model updating. Without RFS during offline training, the tracking performance drops substantially due to overfitting. ROAM-Oracle performs better than SGD-Oracle, showing that our offline learned neural optimizer is more effective than traditional SGD method given the same updating samples. In addition, these two oracles (SGD-oracle and ROAM-oracle) achieve higher AUC score compared with their normal versions, indicating that the tracking accuracy could be boosted by improving the quality of the updating samples.

Architecture of Neural Optimizer. To investigate more architectures of the neural optimizer, we presents three variants of our method: 1) *ROAM-GRU*: using two stacked Gated Recurrent Unit (GRU) [6] as our neural optimizer; 2) *ROAM-FC*: using two linear fully-connected layers fol-
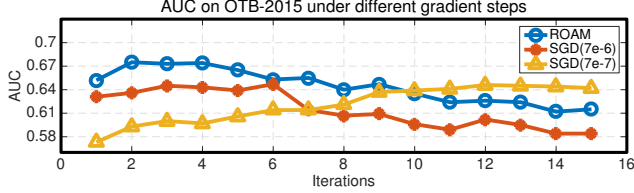
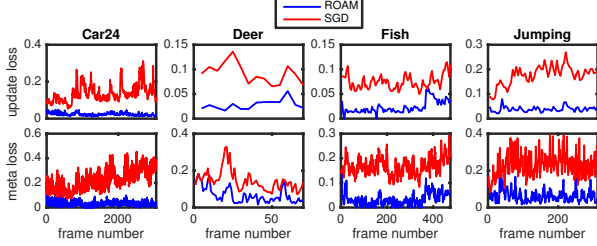Figure 6: AUC vs. gradient steps on OTB-2015.

Figure 7: Update loss (top row) and meta loss (bottom row) comparison between ROAM and SGD.

lowed by tanh activation function as the neural optimizer; 3) *ROAM-ConstLR*: using a learned *constant* element-wise learning rate for model optimization instead of the adaptively generated one. Fig. 5 (Right) presents the results. using ROAM-GRU decreases the AUC slightly, while ROAM-FC has significantly lower AUC compared with ROAM, showing the importance of our recurrent structure. Moreover, the performance drop of ROAM-ConstLR verifies the necessity of using an adaptable learning rate for model updating.

More Steps in Updating. During offline training, we only perform one-step gradient decent to optimize the updating loss. We investigate the effect of using more than one gradient step on tracking performance during the test phase for both ROAM and SGD (see Fig. 6). Our method can be further improved with more than one step, but will gradually decrease when using too many steps. This is because our framework is not trained to perform so many steps during the offline stage. We also use two fixed learning rates for SGD, where the larger one is 7e-6 [6] and the smaller one is 7e-7. Using a larger learning rate, SGD could reach its best performance much faster than using a smaller learning rate, while both have similar best AUC. Our ROAM consistently outperforms SGD(7e-6), showing the superiority of adaptive element-wise learning rates. Furthermore, ROAM with 1-2 gradient steps outperforms SGD(7-e7) using a large number of steps, which shows the improved generalization of ROAM.

Update Loss and Meta Loss Comparison. To show the effectiveness of our neural optimizer, we compare the update loss and meta loss over time between ROAM and SGD after two gradient steps for a few videos of OTB-2015 in Fig. 7 (see supplementary for more examples). Under the same number of gradient updates, our neural optimizer obtains lower loss compared with traditional SGD, demonstrating its faster converge and better generalization ability

---

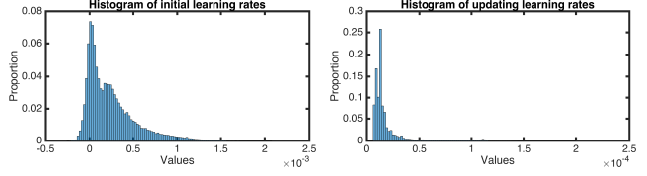[6]MetaTracker uses this learning rate.

Figure 8: Histogram of initial learning rates and updating learning rates on OTB-2015.

for model optimization.

Why does ROAM work? As discussed in [34], directly using the learned initial learning rate $\boldsymbol{\lambda}^{(0)}$ for model optimization in subsequent frames could lead to divergence. This is because the learning rates for model initialization are relatively larger than the ones needed for subsequent frames, which therefore causes unstable model optimization. In particular, the initial model $\boldsymbol{\theta}^{(0)}$ is offline trained to be broadly applicable to different videos, which therefore needs relatively larger gradient step to adapt to a specific task, leading to a relative larger $\boldsymbol{\lambda}^{(0)}$. For the subsequent frames, the appearance variations could be sometimes small or sometimes large, and thus the model optimization process needs an adaptive learning rate to handle different situations. Fig. 8 presents the histograms of initial learning rates and updating learning rates on OTB-2015. Most of updating learning rates are relatively small because usually there are only minor appearance variations between updates. As is shown in Fig. 3, our ROAM, which performs model updating for subsequent frames with adaptive learning rates, obtains substantial performance gain compared with MetaTracker [34], which uses a traditional SGD with a constant learning rate for model updating.

7. Conclusion

In this paper, we propose a novel tracking model consisting of resziable response generator and bounding box regressor, where the first part generates a score map to indicate the presence of target at different spatial locations and the second module regresses bounding box coordinates shift to the anchors densely mounted on sliding window positions. To effectively update the tracking model for appearance adaption, we propose a recurrent model optimization method within a meta learning setting in a few gradient steps. Instead of producing an increment for parameter updating, which is prone to overfit due to different gradient scales, we recurrently generate adaptive learning rates for tracking model optimization using an LSTM. Extensive experiments on OTB, VOT, LaSOT, GOT-10k and TrackingNet datasets demonstrate superior performance compared with state-of-the-art tracking algorithms.

References

[1] Marcin Andrychowicz, Misha Denil, Sergio Gomez, Matthew W. Hoffman, David Pfau, Tom Schaul, and Nando de Freitas. Learning to Learn by Gradient Descent by Gradient Descent. In *NeurIPS*, 2016. 2, 4, 5

[2] Samy Bengio, Yoshua Bengio, Jocelyn Cloutier, and Jan Gecsei. On the Optimization of a Synaptic Learning Rule. In *Preprints Conf. Optimality in Artificial and Biological Neural Networks*, 1992. 2

[3] Luca Bertinetto, Jack Valmadre, Stuart Golodetz, Ondrej Miksik, and Philip Torr. Staple: Complementary Learners for Real-Time Tracking. In *CVPR*, 2016. 2, 6, 7

[4] Luca Bertinetto, Jack Valmadre, João F. Henriques, Andrea Vedaldi, and Philip H. S. Torr. Fully-Convolutional Siamese Networks for Object Tracking. In *ECCV Workshop on VOT*, 2016. 1, 2, 6, 7

[5] Google Brain, Jonathon Shlens, Stefano Mazzocchi, Google Brain, and Vincent Vanhoucke. YouTube-BoundingBoxes : A Large High-Precision Human-Annotated Data Set for Object Detection in Video Esteban Real bear dog airplane zebra. In *CVPR*, 2017. 7

[6] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning Phrase Representations using RNN Encoder-decoder for Statistical Machine Translation. *arXiv preprint arXiv:1406.1078*, 2014. 7

[7] Janghoon Choi, Junseok Kwon, and Kyoung Mu Lee. Deep Meta Learning for Real-Time Target-Aware Visual Tracking. In *ICCV*, 2019. 6

[8] Martin Danelljan, Goutam Bhat, Fahad Shahbaz Khan, and Michael Felsberg. ECO: Efficient Convolution Operators for Tracking. In *CVPR*, 2017. 1, 6, 7

[9] Martin Danelljan, Andreas Robinson, Fahad Shahbaz Khan, and Michael Felsberg. Beyond correlation filters: Learning continuous convolution operators for visual tracking. In *ECCV*, 2016. 1, 2, 6, 7

[10] Heng Fan, Liting Lin, Fan Yang, and Peng Chu. LaSOT : A High-quality Benchmark for Large-scale Single Object Tracking. In *CVPR*, 2019. 6, 7

[11] Heng Fan and Haibin Ling. Siamese Cascaded Region Proposal Networks for Real-Time Visual Tracking. In *CVPR*, 2019. 2, 6

[12] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks. In *ICML*, 2017. 2, 7

[13] Hamed Kiani Galoogahi, Terence Sim, and Simon Lucey. Correlation Filters with Limited Boundaries. In *CVPR*, 2015. 2

[14] Qing Guo, Wei Feng, Ce Zhou, Rui Huang, Liang Wan, and Song Wang. Learning Dynamic Siamese Network for Visual Object Tracking. In *ICCV*, 2017. 7

[15] David Held, Sebastian Thrun, and Silvio Savarese. Learning to Track at 100 FPS with Deep Regression Networks. In *ECCV*, 2016. 7

[16] Joao F. Henriques, Rui Caseiro, Pedro Martins, and Jorge Batista. High-speed tracking with kernelized correlation filters. *TPAMI*, 2015. 1, 2

[17] Lianghua Huang, Xin Zhao, and Kaiqi Huang. GOT-10k : A Large High-Diversity Benchmark for Generic Object Tracking in the Wild. *arXiv*, 2019. 6, 7

[18] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 6

[19] Matej Kristan, Aleš Leonardis, Jiri Matas, and Michael Felsberg. The Visual Object Tracking VOT2016 Challenge Results. In *ECCV Workshop on VOT*, 2016. 1, 6

[20] Matej Kristan, Aleš Leonardis, Jiri Matas, Michael Felsberg, Roman Pflugfelder, Luka Čehovin Zajc, Tomáš Vojír, Gustav Häger, Alan Lukežič, Abdelrahman Eldesokey, Gustavo Fernández, and Others. The Visual Object Tracking VOT2017 Challenge Results. In *ICCV Workshop on VOT*, 2017. 1, 6

[21] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *NeurIPS*, 2012. 6

[22] Bo Li, Wei Wu, Qiang Wang, Fangyi Zhang, Junliang Xing, and Junjie Yan. SiamRPN++: Evolution of Siamese Visual Tracking with Very Deep Networks. In *CVPR*, 2019. 2

[23] Bo Li, Junjie Yan, Wei Wu, Zheng Zhu, and Xiaolin Hu. High Performance Visual Tracking with Siamese Region Proposal Network. In *CVPR*, 2018. 1, 2, 5, 6

[24] Feng Li, Cheng Tian, Wangmeng Zuo, Lei Zhang, and Ming-Hsuan Yang. Learning Spatial-Temporal Regularized Correlation Filters for Visual Tracking. In *CVPR*, pages 4904–4913, 2018. 7

[25] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft COCO: Common Objects in Context. In *ECCV*, pages 740–755, 2014. 6

[26] Xiankai Lu, Chao Ma, Bingbing Ni, Xiaokang Yang, Ian Reid, and Ming-hsuan Yang. Deep Regression Tracking with Shrinkage Loss. In *ECCV*, 2018. 1, 2, 6

[27] Alan Lukežič, Tomáš Vojí, Luka Čehovin, Jií Matas, and Matej Kristan. Discriminative Correlation Filter with Channel and Spatial Reliability. In *CVPR*, 2017. 7

[28] Kaifeng Lv, Shunhua Jiang, and Jian Li. Learning Gradient Descent: Better Generalization and Longer Horizons. In *ICML*, 2017. 2, 5

[29] Chao Ma, Jia-bin Huang, Xiaokang Yang, and Ming-hsuan Yang. Hierarchical Convolutional Features for Visual Tracking. In *ICCV*, 2015. 7

[30] Matthias Müller, Adel Bibi, Silvio Giancola, Salman Al-Subaihi, and Bernard Ghanem. TrackingNet: A Large-Scale Dataset and Benchmark for Object Tracking in the Wild. In *ECCV*, 2018. 6, 7

[31] Tsendsuren Munkhdalai and Hong Yu. Meta Networks. In *ICML*, 2017. 2

[32] Devang K Naik and RJ Mammone. Meta-Neural Networks that Learn by Learning. In *Neural Networks, 1992. IJCNN., International Joint Conference on*, 1992. 2

[33] Hyeonseob Nam and Bohyung Han. Learning Multi-Domain Convolutional Neural Networks for Visual Tracking. In *CVPR*, 2016. 1, 7

[34] Eunbyung Park and Alexander C. Berg. Meta-Tracker: Fast and Robust Online Adaptation for Visual Object Trackers. In *ECCV*, 2018. 2, 3, 6, 7, 8

[35] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic Differentiation in PyTorch. In *NeurIPS Workshop*, 2017. 6

[36] Sachin Ravi and Hugo Larochelle. Optimization As a Model for Few-Shot Learning. In *ICLR*, 2017. 2

[37] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun.

Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. In *NeurIPS*, 2015. 1, 2, 4

[38] Jürgen Schmidhuber. *Evolutionary Principles in Self-referential Learning: On Learning how to Learn: the Meta-meta-meta...-hook*. PhD thesis, 1987. 2

[39] John Schulman, Jonathan Ho, Xi Chen, and Pieter Abbeel. Meta Learning Shared Hierarchies. *arXiv*, 2018. 2

[40] Karen Simonyan and Andrew Zisserman. Very Deep Convolutional Networks for Large-scale Image Recognition. In *ICLR*, 2015. 6

[41] Yibing Song, Chao Ma, Lijun Gong, Jiawei Zhang, Rynson Lau, and Ming-Hsuan Yang. CREST: Convolutional Residual Learning for Visual Tracking. In *ICCV*, 2017. 1, 2, 6

[42] Yibing Song, Chao Ma, Xiaohe Wu, Lijun Gong, Linchao Bao, Wangmeng Zuo, Chunhua Shen, Rynson Lau, and Ming-Hsuan Yang. VITAL: VIsual Tracking via Adversarial Learning. In *CVPR*, 2018. 7

[43] Ran Tao, Efstratios Gavves, and Arnold W. M. Smeulders. Siamese Instance Search for Tracking. In *CVPR*, 2016. 1

[44] Jack Valmadre, Luca Bertinetto, F Henriques, Andrea Vedaldi, and Philip H S Torr. End-to-end representation learning for Correlation Filter based tracking. In *CVPR*, 2017. 1, 7

[45] Ning Wang, Yibing Song, Wengang Zhou, Wei Liu, and Houqiang Li. Unsupervised Deep Tracking. In *CVPR*, 2019. 2

[46] Olga Wichrowska, Niru Maheswaranathan, Matthew W. Hoffman, Sergio Gomez Colmenarejo, Misha Denil, Nando de Freitas, and Jascha Sohl-Dickstein. Learned Optimizers that Scale and Generalize. In *ICML*, 2017. 2, 5

[47] Yi Wu, Jongwoo Lim, and Ming-Hsuan Yang. Object Tracking Benchmark. *TPAMI*, 2015. 6

[48] Tianyu Yang and Antoni B. Chan. Recurrent Filter Learning for Visual Tracking. In *ICCV Workshop on VOT*, 2017. 2

[49] Tianyu Yang and Antoni B. Chan. Learning Dynamic Memory Networks for Object Tracking. In *ECCV*, 2018. 2, 6

[50] Tianyu Yang and Antoni B. Chan. Visual Tracking via Dynamic Memory Networks. *TPAMI*, 2019. 2

[51] Yunhua Zhang, Lijun Wang, Dong Wang, and Huchuan Lu. Structured Siamese Network for Real-Time Visual Tracking. In *ECCV*, 2018. 7

[52] Zhipeng Zhang, Houwen Peng, and Qiang Wang. Deeper and Wider Siamese Networks for Real-Time Visual Tracking. In *CVPR*, 2019. 2, 6

[53] Zheng Zhu, Qiang Wang, Bo Li, Wei Wu, Junjie Yan, and Weiming Hu. Distractor-aware Siamese Networks for Visual Object Tracking. In *ECCV*, 2018. 2, 6