This ICCV paper is the Open Access version, provided by the Computer Vision Foundation.
Except for this watermark, it is identical to the accepted version;
the final published version of the proceedings is available on IEEE Xplore.

Learning Discriminative Model Prediction for Tracking

Goutam Bhat* Martin Danelljan* Luc Van Gool Radu Timofte
CVL, ETH Zürich, Switzerland

Abstract

The current strive towards end-to-end trainable computer vision systems imposes major challenges for the task of visual tracking. In contrast to most other vision problems, tracking requires the learning of a robust target-specific appearance model online*, during the inference stage. To be end-to-end trainable, the online learning of the target model thus needs to be embedded in the tracking architecture itself. Due to the imposed challenges, the popular Siamese paradigm simply predicts a target feature template, while ignoring the background appearance information during inference. Consequently, the predicted model possesses limited target-background discriminability.*

We develop an end-to-end tracking architecture, capable of fully exploiting both target and background appearance information for target model prediction. Our architecture is derived from a discriminative learning loss by designing a dedicated optimization process that is capable of predicting a powerful model in only a few iterations. Furthermore, our approach is able to learn key aspects of the discriminative loss itself. The proposed tracker sets a new state-of-the-art on 6 tracking benchmarks, achieving an EAO score of 0.440 *on VOT2018, while running at over* 40 *FPS. The code and models are available at* https://github.com/visionml/pytracking.

1. Introduction

Generic object tracking is the task of estimating the state of an arbitrary target in each frame of a video sequence. In the most general setting, the target is only defined by its initial state in the sequence. Most current approaches address the tracking problem by constructing a target model, capable of differentiating between the target and background appearance. Since target-specific information is only available at test-time, the target model cannot be learned in an offline training phase, as in for instance object detection. Instead, the target model must be constructed during the inference stage itself by exploiting the target information given at test-time. This unconventional nature of the visual tracking problem imposes significant challenges when pursuing an end-to-end learning solution.

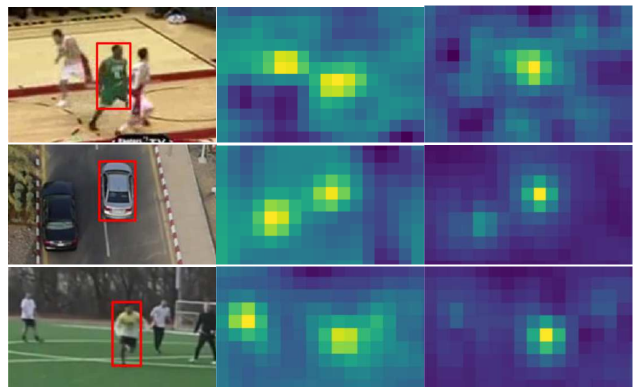

Figure 1. Confidence maps of the target object (red box) provided by the target model obtained using i) a Siamese approach (middle), and ii) Our approach (right). The model predicted in a Siamese fashion, using only target appearance, struggles to distinguish the target from distractor objects in the background. In contrast, our model prediction architecture also integrates background appearance, providing superior discriminative power.

The aforementioned problems have been most successfully addressed by the Siamese learning paradigm [2, 22]. These approaches first learn a feature embedding, where the similarity between two image regions is computed by a simple cross-correlation. Tracking is then performed by finding the image region most similar to the target template. In this setting, the target model simply corresponds to the template features extracted from the target region. Consequently, the tracker can easily be trained end-to-end using pairs of annotated images.

Despite its recent success, the Siamese learning framework suffers from severe limitations. Firstly, Siamese trackers only utilize the target appearance when inferring the model. This completely ignores background appearance information, which is crucial for discriminating the target from similar objects in the scene (see figure 1). Secondly, the learned similarity measure is not necessarily reliable for objects that are not included in the offline training set, leading to poor generalization. Thirdly, the Siamese formulation

*Both authors contributed equally.

does not provide a powerful model update strategy. Instead, state-of-the-art approaches resort to simple template averaging [45]. These limitations result in inferior robustness [20] compared to other state-of-the-art tracking approaches.

In this work, we introduce an alternative tracking architecture, trained in an end-to-end manner, that directly addresses all aforementioned limitations. In our design, we take inspiration from the *discriminative* online learning procedures that have been successfully applied in recent trackers [6, 9, 29]. Our approach is based on a target model prediction network, which is derived from a discriminative learning loss by applying an iterative optimization procedure. The architecture is carefully designed to enable effective end-to-end training, while maximizing the discriminative ability of the predicted model. This is achieved by ensuring a minimal number of optimization steps through two key design choices. First, we employ a steepest descent based methodology that computes an optimal step length in each iteration. Second, we integrate a module that effectively initializes the target model. Furthermore, we introduce significant flexibility into our final architecture by learning the discriminative learning loss itself.

Our entire tracking architecture, along with the backbone feature extractor, is trained using annotated tracking sequences by minimizing the prediction error on future frames. We perform comprehensive experiments on 7 tracking benchmarks: VOT2018 [20], LaSOT [10], TrackingNet [26], GOT10k [16], NFS [12], OTB-100 [42], and UAV123 [25]. Our approach achieves state-of-the-art results on all 7 datasets, while running at over 40 FPS. We also provide an extensive experimental analysis of the proposed architecture, showing the impact of each component.

2. Related Work

Generic object tracking has undergone astonishing progress in recent years, with the development of a variety of approaches. Recently, methods based on Siamese networks [2, 22, 38] have received much attention due to their end-to-end training capabilities and high efficiency. The name derives from the deployment of a Siamese network architecture in order to learn a similarity metric *offline*. Bertinetto *et al*. [2] utilize a fully-convolutional architecture for similarity prediction, thereby attaining high tracking speeds of over 100 FPS. Wang *et al*. [41] learn a residual attention mechanism to adapt the tracking model to the current target. Li *et al*. [22] employ a region proposal network [33] to obtain accurate bounding boxes.

A key limitation in Siamese approaches is their inability to incorporate information from the background region or previous tracked frames into the model prediction. A few recent attempts aim to address these issues. Guo *et al*. [13] learn a feature transformation to handle the target appearance changes and to suppress background. Zhu *et al*. [45] handle background distractors by subtracting corresponding image features from the target template during online tracking. Despite these attempts, the Siamese trackers are yet to reach high level of robustness attained by state-of-the-art trackers employing online learning [20].

In contrast to Siamese methods, another family of trackers [6, 7, 29] learn a discriminative classifier *online* to distinguish the target object from the background. These approaches can effectively utilize background information, thereby achieving impressive robustness on multiple tracking benchmarks [20, 42]. However, such methods rely on more complicated online learning procedures that cannot be easily formulated in an end-to-end learning framework. Thus, these approaches are often restricted to features extracted from deep networks pre-trained for image classification [9, 24] or hand-crafted alternatives [8].

A few recent works aim to formulate existing discriminative online learning based trackers as a neural network component in order to benefit from end-to-end training. Valmadre *et al*. [40] integrate the single-sample closed-form solution of the correlation filter (CF) [15] into a deep network. Yao *et al*. [44] unroll the ADMM iterations in BACF [18] tracker to learn the feature extractor and a few tracking hyper-parameters in a complex multi-stage training procedure. The BACF model learning is however restricted to the single-sample variant of the Fourier-domain CF formulation which cannot exploit multiple samples, requiring ad-hoc linear combination of filters for model adaption.

The problem of learning to predict a target model using only a few images is closely related to meta-learning [11, 27, 28, 32, 34, 35, 39]. A few works have already pursued this direction for tracking. Bertinetto *et al*. [1] meta-train a network to predict the parameters of the tracking model. Choi *et al*. [5] utilize a meta-learner to predict a target-specific feature space to complement the general target-independent feature space used for estimating the similarity in Siamese trackers. Park *et al*. [31] develop a meta-learning framework employing an initial target independent model, which is then refined using gradient descent with learned step-lengths. However, constant step-lengths are only suitable for fast initial adaption of the model and does not provide optimal convergence when applied iteratively.

3. Method

In this work, we develop a discriminative model prediction architecture for tracking. As in Siamese trackers, our approach benefits from end-to-end training. However, unlike Siamese, our architecture can fully exploit background information and provides natural and powerful means of updating the target model with new data. Our model prediction network is derived from two main principles: (i) A discriminative learning loss promoting robustness in the learned target model; and (ii) a powerful optimization strat-

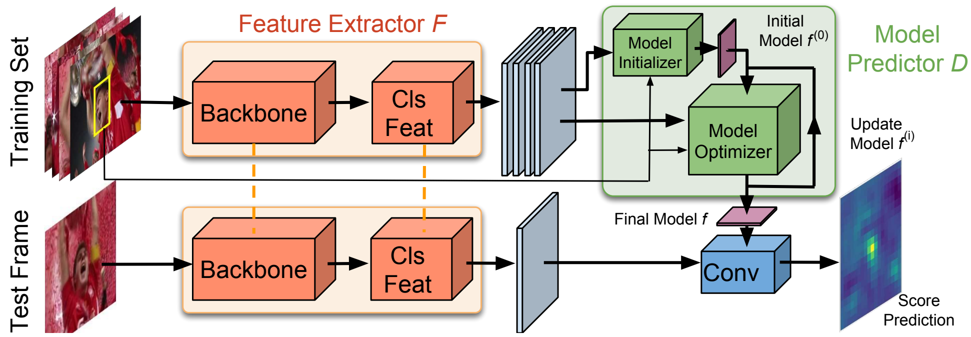

Figure 2. An overview of the target classification branch in our tracking architecture. Given an annotated training set (top left), we extract deep feature maps using a backbone network followed by an additional convolutional block (Cls Feat). The feature maps are then input to the model predictor D, consisting of the initializer and the recurrent optimizer module. The model predictor outputs the weights of the convolutional layer which performs target classification on the feature map extracted from the test frame.

egy ensuring rapid convergence. By such careful design, our architecture can predict the target model in only a few iterations, without compromising its discriminative power.

In our framework, the target model constitutes the weights of a convolutional layer, providing target classification scores as output. Our model prediction architecture computes these weights by taking a set of bounding-box annotated image samples as input. The model predictor includes an initializer network that efficiently provides an initial estimate of the model weights, using only the target appearance. These weights are then processed by the optimizer module, taking both target and background appearance into account. By design, our optimizer module possesses few learnable parameters in order to avoid overfitting to certain classes and scenes during offline training. Our model predictor can thus generalize to unseen objects, which is crucial in *generic* object tracking.

Our final tracking architecture consists of two branches: a target classification branch (see figure 2) for distinguishing the target from background, and a bounding box estimation branch for predicting an accurate target box. Both branches input deep features from a common backbone network. The target classification branch contains a convolutional block, extracting features on which the classifier operates. Given a training set of samples and corresponding target boxes, the model predictor generates the weights of the target classifier. These weights are then applied to features extracted from the test frame, in order to compute the target confidence scores. For the bounding box estimation branch, we utilize the overlap maximization based architecture introduced in [6]. The entire tracking network, including the target classification, bounding box estimation and backbone modules, is trained offline on tracking datasets.

3.1. Discriminative Learning Loss

In this section, we describe the discriminative learning loss used to derive our model prediction architecture. The input to our model predictor D consists of a training set $S_{\text{train}} = \{(x_j, c_j)\}_{j=1}^n$ of deep feature maps $x_j \in \mathcal{X}$ generated by the feature extractor network F. Each sample is paired with the corresponding target center coordinate $c_j \in \mathbb{R}^2$. Given this data, our aim is to predict a target model $f = D(S_{\text{train}})$. The model f is defined as the filter weights of a convolutional layer tasked with discriminating between target and background appearance in the feature space $\mathcal{X}$. We gather inspiration from the least-squares-based regression take on the tracking problem, that has seen tremendous success in the recent years [6, 7, 15]. However, in this work we generalize the conventional least-squares loss applied for tracking in several directions, allowing the final tracking network to *learn* the optimal loss from data.

In general, we consider a loss of the form,

$$L(f) = \frac{1}{|S_{\text{train}}|} \sum_{(x,c) \in S_{\text{train}}} \|r(x * f, c)\|^2 + \|\lambda f\|^2 . \quad (1)$$

Here, $*$ denotes convolution and λ is a regularization factor. The function $r(s, c)$ computes the residual at every spatial location based on the target confidence scores $s = x * f$ and the ground-truth target center coordinate c. The most common choice is $r(s, c) = s - y_c$, where y_c are the desired target scores at each location, popularly set to a Gaussian function centered at c [4]. However, simply taking the difference forces the model to regress calibrated confidence scores, usually zero, for all negative samples. This requires substantial model capacity, forcing the learning to focus on the negative data samples instead of achieving the best discriminative abilities. Furthermore, taking the naïve difference does not address the problem of data imbalance between target and background.

To alleviate the latter issue of data imbalance, we use a spatial weight function v_c. The subscript c indicates the dependence on the center location of the target, as detailed in section 3.4. To accommodate the first issue, we modify the loss following the philosophy of Support Vector Machines. We employ a hinge-like loss in r, clipping the scores at zero as $\max(0, s)$ in the background region. The model is thus free to predict large negative values for easy samples in the background without increasing the loss. For the target region on the other hand, we found it disadvantageous to add an analogous hinge loss $\max(0, 1-s)$. Although contradictory at a first glance, this behavior can be attributed to the fundamental asymmetry between the target and background class, partially due to the numerical imbalance. Moreover, accurately calibrated target confidences are indeed advantageous in the tracking scenario, e.g. for detecting target loss. We therefore desire the properties of standard least-squares regression in the target neighborhood.

To accommodate the advantages of both least-squares regression and the hinge loss, we define the residual function,

$$r(s, c) = v_c \cdot (m_c s + (1 - m_c) \max(0, s) - y_c) \,. \quad (2)$$

The target region is defined by the mask m_c, having values in the interval $m_c(t) \in [0, 1]$ at each spatial location $t \in \mathbb{R}^2$. Again, the subscript c indicate the dependence on the target center coordinate. The formulation in (2) is capable of continuously changing the behavior of the loss from standard least squares regression to a hinge loss depending on the image location relative to the target center c. Setting $m_c \approx 1$ at the target and $m_c \approx 0$ in the background region yields the desired behavior described above. However, how to optimally set m_c is not clear, in particular at the transition region between target and background. While the classical strategy is to manually set the mask parameters using trial and error, our end-to-end formulation allows us to learn the mask in a data-driven manner. In fact, as detailed in section 3.4, our approach learns all free parameters in the loss: the target mask m_c, the spatial weight v_c, the regularization factor λ, and even the regression target y_c itself.

3.2. Optimization-Based Architecture

Here, we derive the network architecture D that predicts the filter $f = D(S_{\text{train}})$ by implicitly minimizing the error (1). The network is designed by formulating an optimization procedure. From eqs. (1) and (2) we can easily derive a closed-form expression for the gradient of the loss ∇L with respect to the filter f (see supplementary material). The straight-forward option is to then employ gradient descent using a step length α,

$$f^{(i+1)} = f^{(i)} - \alpha \nabla L(f^{(i)}) \,. \quad (3)$$

However, we found this simple approach to be insufficient, even if the learning rate α (either a scalar or coefficient-specific) is learned by the network itself (see section 4.1). It experiences slow adaption of the filter parameters f, requiring a vast increase in the number of iterations. This harms efficiency and complicates offline learning.

The slow convergence of gradient descent is largely due to the constant step length α, which does not depend on data or the current model estimate. We solve this issue by deriving a more elaborate optimization approach, requiring only a handful of iterations to predict a strong discriminative filter f. The core idea is to compute the step length α based on the steepest descent methodology, which is a common optimization technique [30, 36]. We first approximate the loss with a quadratic function at the current estimate $f^{(i)}$,

$$\begin{aligned} L(f) \approx \tilde{L}(f) = &\frac{1}{2}(f - f^{(i)})^{\mathrm{T}} Q^{(i)} (f - f^{(i)}) + \\ &(f - f^{(i)})^{\mathrm{T}} \nabla L(f^{(i)}) + L(f^{(i)}) \,. \end{aligned} \quad (4)$$

Here, the filter variables f and $f^{(i)}$ are seen as vectors and $Q^{(i)}$ is positive definite square matrix. The steepest descent then proceeds by finding the step length α that minimizes the approximate loss (4) in the gradient direction (3). This is found by solving $\frac{\mathrm{d}}{\mathrm{d}\alpha} \tilde{L}\left(f^{(i)} - \alpha \nabla L(f^{(i)})\right) = 0$, as

$$\alpha = \frac{\nabla L(f^{(i)})^{\mathrm{T}} \nabla L(f^{(i)})}{\nabla L(f^{(i)})^{\mathrm{T}} Q^{(i)} \nabla L(f^{(i)})} \,. \quad (5)$$

In steepest descent, the formula (5) is used to compute the scalar step length α in each iteration of the filter update (3).

The quadratic model (4), and consequently the resulting step length (5), depends on the choice of $Q^{(i)}$. For example, by using a scaled identity matrix $Q^{(i)} = \frac{1}{\beta} I$ we retrieve the standard gradient descent algorithm with a fixed step length $\alpha = \beta$. On the other hand, we can now integrate second order information into the optimization procedure. The most obvious choice is setting $Q^{(i)} = \frac{\partial^2 L}{\partial f^2}(f^{(i)})$ to the Hessian of the loss (1), which corresponds to a second order Taylor approximation (4). For our least-squares formulation (1) however, the Gauss-Newton method [30] provides a powerful alternative, with significant computational benefits since it only involves first-order derivatives. We thus set $Q^{(i)} = (J^{(i)})^{\mathrm{T}} J^{(i)}$, where $J^{(i)}$ is the Jacobian of the residuals at $f^{(i)}$. In fact, neither the matrix $Q^{(i)}$ or Jacobian $J^{(i)}$ need to be constructed explicitly, but rather implemented as a sequence of neural network operations. See the supplementary material for details. Algorithm 1 describes our target model predictor D. Note that our optimizer module can easily be employed for online model adaption as well. This is achieved by continuously extending the training set S_{train} with new samples from the previously tracked frames. The optimizer module is then applied on this extended training set, using the current target model as the initialization $f^{(0)}$.

3.3. Initial Filter Prediction

To further reduce the number of optimization recursions required in D, we introduce a small network module that predicts an initial model estimate $f^{(0)}$. Our initializer network consists of a convolutional layer followed by a precise ROI pooling [17]. The latter extracts features from the

Algorithm 1 Target model predictor D.

Input: Samples $S_{\text{train}} = \{(x_j, c_j)\}_{j=1}^{n}$, iterations N_{iter}

$f^{(0)} \leftarrow \texttt{ModelInit}(S_{\text{train}})$ # Initialize filter (sec 3.3)
for $i = 0, \dots, N_{\text{iter}} - 1$ **do** # Optimizer module loop
  $\nabla L(f^{(i)}) \leftarrow \texttt{FiltGrad}(f^{(i)}, S_{\text{train}})$ # Using (1)-(2)
  $h \leftarrow J^{(i)} \nabla L(f^{(i)})$ # Apply Jacobian of (2)
  $\alpha \leftarrow \|\nabla L(f^{(i)})\|^2 / \|h\|^2$ # Compute step length (5)
  $f^{(i+1)} \leftarrow f^{(i)} - \alpha \nabla L(f^{(i)})$ # Update filter
end for

target region and pools them to the same size as the target model f. The pooled feature maps are then averaged over all the samples in S_{train} to obtain the initial model $f^{(0)}$. As in Siamese trackers, this approach only utilizes the target appearance. However, rather than predicting the final model, our initializer network is tasked with only providing a reasonable initial estimate, which is then processed by the optimizer module to provide the final model.

3.4. Learning the Discriminative Learning Loss

Here, we describe how the free parameters in the residual function (2), defining the loss (1), are learned. Our residual function includes the label confidence scores y_c, the spatial weight function v_c and the target mask m_c. While such variables are constructed by hand in current discriminative online learning based trackers, our approach in fact learns these functions from data. We parametrize them based on the distance from the target center. This is motivated by the radial symmetry of the problem, where the direction to the sample location relative to the target is of little significance. In contrast, the distance to the sample location plays a crucial role, especially in the transition from target to background. Thus, we parameterize y_c, m_c and v_c using radial basis functions ρ_k and learn their coefficients ϕ_k. For instance, the label y_c at position $t \in \mathbb{R}^2$ is given by

$$y_c(t) = \sum_{k=0}^{N-1} \phi_k^y \rho_k(\|t - c\|)\,. \tag{6}$$

We use triangular basis functions ρ_k, defined as

$$\rho_k(d) = \begin{cases} \max(0, 1 - \frac{|d - k\Delta|}{\Delta}), & k < N-1 \\ \max(0, \min(1, 1 + \frac{d - k\Delta}{\Delta})), & k = N-1 \end{cases} \tag{7}$$

The above formulation corresponds to a continuous piecewise linear function with a knot displacement of Δ. Note that the final case $k = N-1$ represents all locations that are far away from the target center and thus can be treated identically. We use a small Δ to enable accurate representation of the regression label at the target-background transition. The functions v_c and m_c are parameterized analogously using coefficients ϕ_k^v and ϕ_k^m respectively in (6). For the target mask m_c, we constrain the values to the interval $[0, 1]$ by passing the output from (6) through a Sigmoid function.

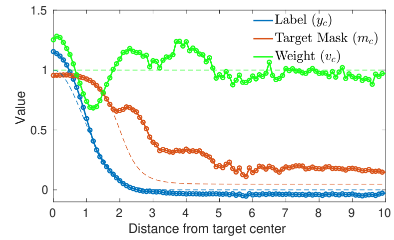

Figure 3. Plot of the learned regression label (y_c), target mask (m_c), and spatial weight (v_c). The markers show the knot locations. The initialization of each quantity is shown in dotted lines.

We use $N = 100$ basis functions and set the knot displacement to $\Delta = 0.1$ in the resolution of the deep feature space $\mathcal{X}$. For offline training, the regression label y_c is initialized to the same Gaussian z_c used in the offline classification loss, described in section 3.6. The weight function v_c is initialized to constant $v_c(t) = 1$. Lastly, we initialize the target mask m_c using a scaled $\tanh$ function. The coefficients ϕ_k, along with λ, are learned as part of the model prediction network D (see section 3.6). The initial and learned values for y_c, m_c and v_c are visualized in figure 3. Notably, our network learns to increase the weight v_c at the target center and reduce it in the ambiguous transition region.

3.5. Bounding Box Estimation

We utilize the overlap maximization strategy introduced in [6] for the task of accurate bounding box estimation. Given a reference target appearance, the bounding box estimation branch is trained to predict the IoU overlap between the target and a set of candidate boxes on a test image. The target information is integrated into the IoU prediction by computing a modulation vector from the reference appearance of the target. The computed vector is used to modulate the features from the test image, which are then used for IoU prediction. The IoU prediction network is differentiable w.r.t. the input box co-ordinates, allowing the candidates to be refined during tracking by maximizing the predicted IoU. We use the same network architecture as in [6].

3.6. Offline Training

Here, we describe our offline training procedure. In Siamese approaches, the network is trained with image pairs, using one image to predict the target template and the other for evaluating the tracker. In contrast, our model prediction network D inputs a *set* S_{train} of multiple data samples from the sequence. To better exploit this advantage, we train our full tracking architecture on pairs of sets $(M_{\text{train}}, M_{\text{test}})$. Each set $M = \{(I_j, b_j)\}_{j=1}^{N_{\text{frames}}}$ consists of images I_j paired with their corresponding target bounding boxes b_j. The target model is predicted using M_{train} and then evaluated on the test frames M_{test}. Uniquely, our training allows the model predictor D to learn how to better utilize multiple samples. The sets are constructed by sampling a random segment of length T_{ss} in the sequence. We then construct M_{train} and M_{test} by sampling N_{frames} frames each from the first and second halves of the segment respectively.

Given the pair $(M_{\text{train}}, M_{\text{test}})$, we first pass the images through the backbone feature extractor to construct the train S_{train} and test S_{test} samples for our target model. Formally, the train set is obtained as $S_{\text{train}} = \{(F(I_j), c_j) : (I_j, b_j) \in M_{\text{train}}\}$, where c_j is the center coordinate of the box b_j. This is input to the target predictor $f = D(S_{\text{train}})$. The aim is to predict a model f that is discriminative and that generalizes well to future unseen frames. We therefore only evaluate the predicted model f on the test samples S_{test}, obtained analogously using M_{test}. Following the discussion in section 3.1, we compute the regression errors using a hinge for the background samples,

$$\ell(s,z) = \begin{cases} s - z\,, & z > T \\ \max(0, s)\,, & z \leq T \end{cases}. \tag{8}$$

Here, the threshold T defines the target and background region based on the label confidence value z. For the target region $z > T$ we take the difference between the predicted confidence score s and the label z, while we only penalize positive confidence values for the background $z \leq T$.

The total target classification loss is computed as the mean squared error (8) over all test samples. However, instead of only evaluating the final target model f, we average the loss over the estimates $f^{(i)}$ obtained in each iteration i by the optimizer (see alg. 1). This introduces intermediate supervision to the target prediction module, benefiting training convergence. Furthermore, we do not aim to train for a specific number of recursions, but rather be free to set the desired number of optimization recursions online. It is thus natural to evaluate each iterate $f^{(i)}$ equally. The target classification loss used for offline training is given by,

$$L_{\text{cls}} = \frac{1}{N_{\text{iter}}} \sum_{i=0}^{N_{\text{iter}}} \sum_{(x,c)\in S_{\text{test}}} \left\| \ell\big(x * f^{(i)}, z_c\big) \right\|^2. \tag{9}$$

Here, regression label z_c is set to a Gaussian function centered as the target c. Note that the output $f^{(0)}$ from the filter initializer (section 3.3) is also included in the above loss. Although not denoted explicitly to avoid clutter, both x and $f^{(i)}$ in (9) depend on the parameters of the feature extraction network F. The model iterates $f^{(i)}$ additionally depend on the parameters in the model predictor network D.

For bounding box estimation, we extend the training procedure in [6] to image sets by computing the modulation vector on the first frame in M_{train} and sampling candidate boxes from all images in M_{test}. The bounding box estimation loss L_{bb} is computed as the mean squared error between the predicted IoU overlaps in M_{test} and the ground truth. We train the full tracking architecture by combining this with the target classification loss (9) as $L_{\text{tot}} = \beta L_{\text{cls}} + L_{\text{bb}}$.

Training details: We use the training splits of the TrackingNet [26], LaSOT [10], GOT10k [16] and COCO [23] datasets. The backbone network is initialized with the ImageNet weights. We train for 50 epochs by sampling 20,000 videos per epoch, giving a total training time of less than 24 hours on a single Nvidia TITAN X GPU. We use ADAM [19] with learning rate decay of 0.2 every 15th epoch. The target classification loss weight is set to $\beta = 10^2$ and we use $N_{\text{iter}} = 5$ optimizer module recursions in (9) during training. The image patches in $(M_{\text{train}}, M_{\text{test}})$ are extracted by sampling a random translation and scale relative to the target annotation. We set the base scale to 5 times the target size to incorporate significant background information. For each sequence, we sample $N_{\text{frames}} = 3$ test and train frames, using a segment length of $T_{\text{ss}} = 60$. The label scores z_c are constructed using a standard deviation of $1/4$ relative to the base target size, and we use $T = 0.05$ for the regression error (8). We employ the ResNet architecture for the backbone. For the model predictor D, we use features extracted from the third block, having a spatial stride of 16. We set the kernel size of the target model f to 4×4.

3.7. Online Tracking

Given the first frame with annotation, we employ data augmentation strategies [3] to construct an initial set S_{train} containing 15 samples. The target model is then obtained using our discriminative model prediction architecture $f = D(S_{\text{train}})$. For the first frame, we employ 10 steepest descent recursions, after the initializer module. Our approach allows the target model to be easily updated by adding a new training sample to S_{train} whenever the target is predicted with sufficient confidence. We ensure a maximum memory size of 50 by discarding the oldest sample. During tracking, we refine the target model f by performing two optimizer recursions every 20 frames, or a single recursion whenever a distractor peak is detected. Bounding box estimation is performed using the same settings as in [6].

4. Experiments

Our approach is implemented in Python using PyTorch, and operates at 57 FPS with a ResNet-18 backbone and 43 FPS with ResNet-50 on a single Nvidia GTX 1080 GPU.

4.1. Analysis of our Approach

Here, we perform an extensive analysis of the proposed model prediction architecture. Experiments are performed on a combined dataset containing the entire OTB-100 [42], NFS (30 FPS version) [12] and UAV123 [25] datasets. This pooled dataset contains 323 diverse videos to enable thorough analysis. The trackers are evaluated using the AUC

	Init	GD	SD
AUC	58.2	61.6	63.8

Table 1. Analysis of different model prediction architectures on the combined OTB-100, NFS and UAV123 datasets. The architecture using only the target information for model prediction (**Init**) achieves an AUC score of 58.2%. The proposed steepest descent based architecture (**SD**) provides the best results, outperforming the gradient descent method (**GD**) by over 2.2% AUC score.

	SD	+Init	+FT	+Cls	+Loss
AUC	58.7	60.0	62.6	63.3	63.8

Table 2. Analysis of the impact of initializer module (**+Init**), training the backbone (**+FT**), using extra conv. block (**+Cls**) and offline learning of the loss (**+Loss**), by incrementally adding them one at a time. The baseline **SD** constitutes our steepest descent based optimizer module along with a ResNet-18 trained on ImageNet.

[42] metric. Due to the stochastic nature of the tracker, we always report the average AUC score over 5 runs. We employ ResNet-18 as the backbone network for this analysis.

Impact of optimizer module: We compare our proposed method, utilizing the steepest descent (**SD**) based architecture, with two alternative approaches. **Init:** Here, we only use the initializer module to predict the final target model, which corresponds to removing the optimizer module in our approach. Thus, similar to the Siamese approaches, only target appearance information is used for model prediction, while background information is discarded. **GD:** In this approach, we replace steepest descent with the gradient descent (GD) algorithm using learned coefficient-wise step-lengths α in (3). All networks are trained using the same settings. The results for this analysis are shown in table 1.

The model predicted by the initializer network, which uses only target information, achieves an AUC score of 58.2%. The gradient descent approach, which can exploit background information, provides a substantial improvement, achieving an AUC score of 61.6%. This highlights the importance of employing discriminative learning for model prediction. Our steepest descent approach obtains the best results, outperforming GD by 2.2%. This is due to the superior convergence properties of steepest descent, important for offline learning and fast online tracking.

Analysis of model prediction architecture: Here, we analyze the impact of key aspects of the proposed discriminative online learning architecture, by incrementally adding them one at a time. The results are shown in table 2. The baseline **SD** constitutes our steepest descent based optimizer module along with a fixed ResNet-18 network trained on ImageNet. That is, similar to the current state-of-the-art discriminative approaches, we do not fine-tune the backbone. Instead of learning the discriminative loss, we employ the regression error (8) in the optimizer module. This baseline approach achieves an AUC score of 58.7%. By adding the model initializer module (**+Init**), we achieve a significant gain of 1.3% in AUC score. Further training the entire network, including backbone feature extractor, (**+FT**) leads to a major improvement of 2.6% in AUC score. This demonstrates the advantages of learning specialized features suitable for tracking through end-to-end learning. Using an additional convolutional block to extract classification specific features (**+Cls**) yields a further improvement of 0.7% AUC score. Finally, learning the discriminative loss (2) itself (**+Loss**), as described in section 3.4, improves the AUC score by another 0.5%. This shows the benefit of learning the implicit online loss by maximizing the generalization capabilities of the model on future frames.

	No update	Model averaging	Ours
AUC	61.7	61.7	63.8

Table 3. Comparison of different model update strategies on the combined OTB-100, NFS and UAV123 datasets.

	DRT [37]	RCO [20]	UPDT [3]	DaSiam-RPN [45]	MFT [20]	LADCF [43]	ATOM [6]	SiamRPN++ [21]	**DiMP-18**	**DiMP-50**
EAO	0.356	0.376	0.378	0.383	0.385	0.389	0.401	**0.414**	0.402	**0.440**
Robustness	0.201	0.155	0.184	0.276	**0.140**	0.159	0.204	0.234	0.182	**0.153**
Accuracy	0.519	0.507	0.536	0.586	0.505	0.503	0.590	**0.600**	0.594	**0.597**

Table 4. State-of-the-art comparison on the VOT2018 dataset in terms of expected average overlap (EAO), accuracy & robustness.

Impact of online model update: Here, we analyze the impact of updating the target model online, using information from previous tracked frames. We compare three different model update strategies. i) **No update:** The model is not updated during tracking. Instead, the model predicted in the first frame by our model predictor D, is employed for the entire sequence. ii) **Model averaging:** In each frame, the target model is updated using the linear combination of the current and newly predicted model, as commonly employed in tracking [15, 18, 40]. iii) **Ours:** The target model is obtained using the training set constructed online, as described in section 3.7. The naïve model averaging fails to improve over the baseline method with no updates (see table 3). In contrast, our approach obtains a significant gain of about 2% in AUC score over both methods, indicating that our approach can effectively adapt the target model online.

4.2. State-of-the-art Comparison

We compare our proposed approach **DiMP** with the state-of-the-art methods on seven challenging tracking benchmarks. Results for two versions of our approach are shown: DiMP-18 and DiMP-50 employing ResNet-18 and ResNet-50 respectively as the backbone network.

VOT2018 [20]: We evaluate our approach on the 2018 version of the Visual Object Tracking (VOT) challenge consisting of 60 challenging videos. Trackers are evaluated using the measures accuracy (average overlap over successfully tracked frames) and robustness (failure rate). Both these measures are combined to get the EAO (Expected Average Overlap) score used to rank trackers. The results are shown in table 4. Among previous approaches,

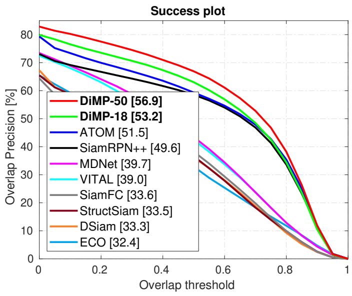

Figure 4. Success plot on the LaSOT dataset.

SiamRPN++ achieves the best accuracy and EAO. However, it attains much inferior robustness compared to the discriminative learning based approaches, such as MFT and LADCF. Similar to the aforementioned approaches, SiamRPN++ employs ResNet-50 for feature extraction. Our approach DiMP-50, employing the same backbone network, significantly outperforms SiamRPN++ with a relative gain of 6.3% in terms of EAO. Further, compared to SiamRPN++, our approach has a 34% lower failure rate, while achieving similar accuracy. This shows that discriminative model prediction is crucial for robust tracking.

LaSOT [10]: We evaluate our approach on the test set consisting of 280 videos. The success plots are shown in figure 4. Compared to other datasets, LaSOT has longer sequences, with an average of 2500 frames per sequence. Thus, online model adaption is crucial for this dataset. The previous best approach ATOM [6] employs online discriminative learning with with pre-trained ResNet-18 features. Our end-to-end trained approach, using the same backbone architecture, outperforms ATOM with a relative gain of 3.3%, showing the impact of end-to-end training. DiMP-50 further improves the results with an AUC score of 56.9%. These results demonstrate the powerful model adaption capabilities of our method on long sequences.

TrackingNet [26]: We evaluate our approach on the test set of the large-scale TrackingNet dataset. The results are shown in table 5. SiamRPN++ achieves an impressive AUC score of 73.3%. Our approach, with the same ResNet-50 backbone as in SiamRPN++, outperforms all previous methods by achieving AUC score of 74.0%.

GOT10k [16]: This is large-scale dataset containing over 10,000 videos, 180 of which form the test set used for evaluation. Interestingly, there is no overlap in object classes between the train and test splits, promoting the importance of generalization to *unseen* object classes. To ensure fair evaluation, the trackers are forbidden from using external datasets for training. We follow this protocol by retraining our trackers using only the GOT10k train split. Results are shown in table 6. ATOM achieves an average overlap (AO) score of 55.6%. Our ResNet-18 version outperforms ATOM with a relative gain of 4.1%. Our ResNet-50 version achieves the best AO score of 61.1%, verifying the strong generalization abilities of our tracker.

	ECO [7]	SiamFC [2]	CFNet [40]	MDNet [29]	UPDT [3]	DaSiam-RPN [45]	ATOM [6]	SiamRPN++ [21]	**DiMP-18**	**DiMP-50**
Precision (%)	49.2	53.3	53.3	56.5	55.7	59.1	64.8	**69.4**	66.6	**68.7**
Norm. Prec. (%)	61.8	66.6	65.4	70.5	70.2	73.3	77.1	**80.0**	78.5	**80.1**
Success (AUC) (%)	55.4	57.1	57.8	60.6	61.1	63.8	70.3	**73.3**	72.3	**74.0**

Table 5. State-of-the-art comparison on the TrackingNet test set in terms of precision, normalized precision, and success.

	MDNet [29]	CF2 [24]	ECO [7]	CCOT [9]	GOTURN [14]	SiamFC [2]	SiamFCv2 [40]	ATOM [6]	**DiMP-18**	**DiMP-50**
$SR_{0.50}$ (%)	30.3	29.7	30.9	32.8	37.5	35.3	40.4	63.4	**67.2**	**71.7**
$SR_{0.75}$ (%)	9.9	8.8	11.1	10.7	12.4	9.8	14.4	40.2	**44.6**	**49.2**
AO (%)	29.9	31.5	31.6	32.5	34.7	34.8	37.4	55.6	**57.9**	**61.1**

Table 6. State-of-the-art comparison on the GOT10k test set in terms of average overlap (AO), and success rates (SR) at overlap thresholds 0.5 and 0.75.

	ECOhc [7]	DaSiam-RPN [45]	ATOM [6]	CCOT [9]	MDNet [29]	ECO [7]	SiamRPN++ [21]	UPDT [3]	**DiMP-18**	**DiMP-50**
NFS	-	-	58.4	48.8	42.2	46.6	-	53.7	**61.0**	**62.0**
OTB-100	64.3	65.8	66.9	68.2	67.8	69.1	**69.6**	**70.2**	66.0	68.4
UAV123	50.6	58.6	**64.4**	51.3	52.8	52.5	61.3	54.5	64.3	**65.4**

Table 7. State-of-the-art comparison on the NFS, OTB-100 and UAV123 datasets in terms of AUC score.

Need for Speed [12]: We evaluate our approach on the 30 FPS version of the dataset, containing challenging videos with fast-moving objects. The AUC scores over all the 100 videos are shown in table 7. The previous best method ATOM achieves an AUC score of 58.4% . Our approach outperforms ATOM with relative gains of 4.4% and 6.2% using ResNet-18 and ResNet-50 respectively.

OTB-100 [42]: Table 7 shows the AUC scores over all the 100 videos in the dataset. Among the compared methods, UPDT achieves the best results with an AUC score of 70.2%. Our DiMP-50 achieves an AUC score of 68.4%, competitive with the other state-of-the-art approaches.

UAV123 [25]: This dataset consists of 123 low altitude aerial videos captured from a UAV. Results in terms of AUC are shown in table 7. Among previous methods, SiamRPN++ achieves an AUC score of 61.3%. Both DiMP-18 and DiMP-50 significantly outperform SiamRPN++, achieving AUC scores of 64.3% and 65.4%, respectively.

5. Conclusions

We propose a tracking architecture that is trained offline in an end-to-end manner. Our approach is derived from a discriminative learning loss by applying an iterative optimization procedure. By employing a steepest descent based optimizer and an effective model initializer, our approach can predict a powerful model in only a few optimization steps. Further, our approach learns the discriminative loss during offline training by minimizing the prediction error on unseen test frames. Our approach sets a new state-of-the-art on 6 tracking benchmarks, while operating at over 40 FPS.

Acknowledgments: This work was supported by ETH General Fund (OK), and Nvidia through a hardware grant.

References

[1] Luca Bertinetto, João F. Henriques, Jack Valmadre, Philip H. S. Torr, and Andrea Vedaldi. Learning feed-forward one-shot learners. In *NIPS*, 2016. 2

[2] Luca Bertinetto, Jack Valmadre, João F Henriques, Andrea Vedaldi, and Philip HS Torr. Fully-convolutional siamese networks for object tracking. In *ECCV workshop*, 2016. 1, 2, 8

[3] Goutam Bhat, Joakim Johnander, Martin Danelljan, Fahad Shahbaz Khan, and Michael Felsberg. Unveiling the power of deep tracking. In *ECCV*, 2018. 6, 7, 8

[4] David S. Bolme, J. Ross Beveridge, Bruce A. Draper, and Yui Man Lui. Visual object tracking using adaptive correlation filters. In *CVPR*, 2010. 3

[5] Janghoon Choi, Junseok Kwon, and Kyoung Mu Lee. Deep meta learning for real-time visual tracking based on target-specific feature space. *CoRR*, abs/1712.09153, 2017. 2

[6] Martin Danelljan, Goutam Bhat, Fahad Shahbaz Khan, and Michael Felsberg. ATOM: Accurate tracking by overlap maximization. In *CVPR*, 2019. 2, 3, 5, 6, 7, 8

[7] Martin Danelljan, Goutam Bhat, Fahad Shahbaz Khan, and Michael Felsberg. ECO: efficient convolution operators for tracking. In *CVPR*, 2017. 2, 3, 8

[8] Martin Danelljan, Gustav Häger, Fahad Shahbaz Khan, and Michael Felsberg. Learning spatially regularized correlation filters for visual tracking. In *ICCV*, 2015. 2

[9] Martin Danelljan, Andreas Robinson, Fahad Shahbaz Khan, and Michael Felsberg. Beyond correlation filters: Learning continuous convolution operators for visual tracking. In *ECCV*, 2016. 2, 8

[10] Heng Fan, Liting Lin, Fan Yang, Peng Chu, Ge Deng, Sijia Yu, Hexin Bai, Yong Xu, Chunyuan Liao, and Haibin Ling. Lasot: A high-quality benchmark for large-scale single object tracking. *CoRR*, abs/1809.07845, 2018. 2, 6, 8

[11] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *ICML*, 2017. 2

[12] Hamed Kiani Galoogahi, Ashton Fagg, Chen Huang, Deva Ramanan, and Simon Lucey. Need for speed: A benchmark for higher frame rate object tracking. In *ICCV*, 2017. 2, 6, 8

[13] Qing Guo, Wei Feng, Ce Zhou, Rui Huang, Liang Wan, and Song Wang. Learning dynamic siamese network for visual object tracking. In *ICCV*, 2017. 2

[14] David Held, Sebastian Thrun, and Silvio Savarese. Learning to track at 100 fps with deep regression networks. In *ECCV*, 2016. 8

[15] João F. Henriques, Rui Caseiro, Pedro Martins, and Jorge Batista. High-speed tracking with kernelized correlation filters. *TPAMI*, 37(3):583–596, 2015. 2, 3, 7

[16] Lianghua Huang, Xin Zhao, and Kaiqi Huang. Got-10k: A large high-diversity benchmark for generic object tracking in the wild. *arXiv preprint arXiv:1810.11981*, 2018. 2, 6, 8

[17] Boru Jiang, Ruixua Luo, Jiayuan Mao, Tete Xiao, and Yuning Jiang. Acquisition of localization confidence for accurate object detection. In *ECCV*, 2018. 4

[18] Hamed Kiani Galoogahi, Ashton Fagg, and Simon Lucey. Learning background-aware correlation filters for visual tracking. In *ICCV*, 2017. 2, 7

[19] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR*, 2014. 6

[20] Matej Kristan, Ales Leonardis, Jiri Matas, Michael Felsberg, Roman Pfugfelder, Luka Cehovin Zajc, Tomas Vojir, Goutam Bhat, Alan Lukezic, Abdelrahman Eldesokey, Gustavo Fernandez, and et al. The sixth visual object tracking vot2018 challenge results. In *ECCV workshop*, 2018. 2, 7

[21] 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. 7, 8

[22] 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

[23] Tsung-Yi Lin, Michael Maire, Serge J. Belongie, Lubomir D. Bourdev, Ross B. Girshick, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft COCO: common objects in context. In *ECCV*, 2014. 6

[24] Chao Ma, Jia-Bin Huang, Xiaokang Yang, and Ming-Hsuan Yang. Hierarchical convolutional features for visual tracking. In *ICCV*, 2015. 2, 8

[25] Matthias Mueller, Neil Smith, and Bernard Ghanem. A benchmark and simulator for uav tracking. In *ECCV*, 2016. 2, 6, 8

[26] 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. 2, 6, 8

[27] Tsendsuren Munkhdalai and Hongtao Yu. Meta networks. *Proceedings of machine learning research*, 70:2554–2563, 2017. 2

[28] Devang Balvantrai Naik and Richard J. Mammone. Meta-neural networks that learn by learning. *[Proceedings 1992] IJCNN International Joint Conference on Neural Networks*, 1:437–442 vol.1, 1992. 2

[29] Hyeonseob Nam and Bohyung Han. Learning multi-domain convolutional neural networks for visual tracking. In *CVPR*, 2016. 2, 8

[30] Jorge Nocedal and Stephen J. Wright. *Numerical Optimization*. Springer, 2nd edition, 2006. 4

[31] Eunbyung Park and Alexander C. Berg. Meta-tracker: Fast and robust online adaptation for visual object trackers. In *ECCV*, 2018. 2

[32] Sachin Ravi and Hugo Larochelle. Optimization as a model for few-shot learning. In *ICLR*, 2017. 2

[33] Shaoqing Ren, Kaiming He, Ross B. Girshick, and Jian Sun. Faster R-CNN: towards real-time object detection with region proposal networks. In *NIPS*, 2015. 2

[34] Jurgen Schmidhuber. Evolutionary principles in self-referential learning. on learning now to learn: The meta-meta-meta...-hook. Diploma thesis, Technische Universitat Munchen, Germany, 14 May 1987. 2

[35] Jürgen Schmidhuber. Learning to control fast-weight memories: An alternative to dynamic recurrent networks. *Neural Comput.*, 4(1):131–139, Jan. 1992. 2

[36] Jonathan R Shewchuk. An introduction to the conjugate gradient method without the agonizing pain. Technical report, Pittsburgh, PA, USA, 1994. 4

[37] Chong Sun, Dong Wang, Huchuan Lu, and Ming-Hsuan Yang. Correlation tracking via joint discrimination and reliability learning. In *CVPR*, 2018. 7

[38] Ran Tao, Efstratios Gavves, and Arnold W. M. Smeulders. Siamese instance search for tracking. In *CVPR*, 2016. 2

[39] Sebastian Thrun and Lorien Pratt, editors. *Learning to Learn*. Kluwer Academic Publishers, Norwell, MA, USA, 1998. 2

[40] Jack Valmadre, Luca Bertinetto, João F Henriques, Andrea Vedaldi, and Philip H. S. Torr. End-to-end representation learning for correlation filter based tracking. In *CVPR*, 2017. 2, 7, 8

[41] Qiang Wang, Zhu Teng, Junliang Xing, Jin Gao, Weiming Hu, and Stephen J. Maybank. Learning attentions: Residual attentional siamese network for high performance online visual tracking. In *CVPR*, 2018. 2

[42] Yi Wu, Jongwoo Lim, and Ming-Hsuan Yang. Object tracking benchmark. *TPAMI*, 37(9):1834–1848, 2015. 2, 6, 7, 8

[43] Tianyang Xu, Zhen-Hua Feng, Xiao-Jun Wu, and Josef Kittler. Learning adaptive discriminative correlation filters via temporal consistency preserving spatial feature selection for robust visual tracking. *CoRR*, abs/1807.11348, 2018. 7

[44] Yingjie Yao, Xiaohe Wu, Shiguang Shan, and Wangmeng Zuo. Joint representation and truncated inference learning for correlation filter based tracking. In *ECCV*, 2018. 2

[45] Zheng Zhu, Qiang Wang, Li Bo, Wei Wu, Junjie Yan, and Weiming Hu. Distractor-aware siamese networks for visual object tracking. In *ECCV*, 2018. 2, 7, 8