This ICCV workshop paper is the Open Access version, provided by the Computer Vision Foundation.
Except for this watermark, it is identical to the version available on IEEE Xplore.

Recurrent Filter Learning for Visual Tracking

Tianyu Yang Antoni B. Chan
City University of Hong Kong
tianyyang8-c@my.cityu.edu.hk, abchan@cityu.edu.hk

Abstract

Recently using convolutional neural networks (CNNs) has gained popularity in visual tracking, due to its robust feature representation of images. Recent methods perform online tracking by fine-tuning a pre-trained CNN model to the specific target object using stochastic gradient descent (SGD) back-propagation, which is usually time-consuming. In this paper, we propose a recurrent filter generation methods for visual tracking. We directly feed the target's image patch to a recurrent neural network (RNN) to estimate an object-specific filter for tracking. As the video sequence is a spatiotemporal data, we extend the matrix multiplications of the fully-connected layers of the RNN to a convolution operation on feature maps, which preserves the target's spatial structure and also is memory-efficient. The tracked object in the subsequent frames will be fed into the RNN to adapt the generated filters to appearance variations of the target. Note that once the off-line training process of our network is finished, there is no need to fine-tune the network for specific objects, which makes our approach more efficient than methods that use iterative fine-tuning to online learn the target. Extensive experiments conducted on widely used benchmarks, OTB and VOT, demonstrate encouraging results compared to other recent methods.

1. Introduction

Given an object of interest labeled by a bounding box in the first frame, the goal of generic visual tracking is to locate this target in subsequent frames automatically. Since the type of object may not be known in advance under some scenarios, it is infeasible to gather a lot of data to train an object-specific tracker. Hence, generic visual tracking should be robust enough to work with any type of object, while also being sufficiently adaptable to handle the appearance of the specific object and variations of appearance during tracking. Visual tracking plays a crucial role in numerous vision application such smart surveillance, autonomous driving and human-computer interaction.

Most existing algorithms locate the object by online training a discriminative model to classify the target from the background. This self-updating paradigm assumes that the object's appearance changes smoothly, but is inappropriate in challenging situations such as heavy occlusion, illumination changes and abrupt motion. Several methods adopt multiple experts [46], multiple instance learning [2], or short and long term memory stores [19] to address the problem of drastic appearance changes. Recent advances using CNNs for object recognition and detection has inspired tracking algorithms to employ the discriminative features learned by CNNs. In particular, [27, 33, 8] feed the CNN features into a traditional visual tracker, the correlation filter [18], to get a response map for target's estimated location.

However, the application domain of object recognition/detection is quite different from visual tracking. In object recognition, the networks are trained to recognize specific categories of objects, whereas in visual tracking the type of object is unknown, and varies from sequence to sequence. Furthermore, because the CNNs are trained to recognize object classes (e.g., dogs), the high-level features are *invariant* to appearance variations within the object class (e.g., a brown dog vs. a white dog). In contrast, visual tracking is concerned about recognizing a *specific instance* of an object (a brown dog), possibly among distractor objects in the same class (white dogs). Thus, naively applying CNNs trained on the object recognition task is not suitable. One way to address this problem is to fine-tune a pre-trained CNN models for each test sequence from the first frame, but this is time-consuming and is prone to overfitting due to the limited available labeled data. Therefore, using a smaller learning rate, constraining the number of iteration for SGD back-propagation, and only fine-tuning fully-connected layers [31, 30] have been proposed for adapting the CNN to the specific target, while also alleviating the risk of ruining the weights of model.

In contrast to these object-specific methods, we propose a *recurrent filter learning* (RFL) algorithm by maintaining the target appearance and tracking filter through a Long Short Term Memory (LSTM) network. A fully convolutional neural networks is used to encode the target appear-

AIRSPACE SYS. INC. EX2047, Page 1
(Axon Enter., Inc. and Dedrone Holdings, Inc. and Airspace Sys., Inc.), IPR2025-00629

ance information, while preserving the spatial structure of the target. Naively flattening the CNN feature map into a vector in order to pass it to an LSTM would obfuscate the structure of the target. Instead, inspired by [37], we change the input, output, cell and hidden states of the LSTM into feature maps, and use convolution layers instead of fully-connected layers. The output of the convolutional-LSTM is itself a filter, which is then convolved with the feature map of the next frame to produce the target response map.

Our RFL network has several interesting properties. First, once the offline training process is finished, there is no need to retrain the network for specific objects during test time. Instead, a purely feed-forward process is used, where, for each frame, the image patch of the tracked target is input into the network, which then updates its appearance model and produces a target-specific filter for finding the target in the next frame. Second, the object does not have to be at the center of exemplar patch due the fully convolutional structure of our LSTM, which is shift invariant. This is very useful for updating the tracker, since the estimated bounding box for each frame may not be accurate, resulting in shifting or stretching the target in the exemplar image. Furthermore, the convolution of the search image and the generated filter is actually a dense sliding window searching strategy, implemented in a more efficient way. The contribution of this paper are three-fold:

1. We propose a novel recurrent filter learning framework that captures both spatial and temporal information of sequence for visual tracking, and does not require fine-tuning during tracking.
2. We design an efficient and effective method for initializing and updating the target appearance for a specific object by using a convolutional LSTM as the memory unit.
3. We conduct extensive experiments on the widely used benchmark OTB [44] and VOT [24], which contains 100 challenging sequences and 60 videos respectively to evaluate the effectiveness of our tracker.

2. Related Work

2.1. CNN-based Trackers

The past several years have seen great success of CNNs on various computer vision tasks, especially on object recognition and detection[38, 16, 35, 34]. Recently, several trackers embed CNNs into their frameworks due to its outstanding power of its feature representations. One straightforward way of using CNNs is treating it as a feature extractor. [27] adopts the output of some specific layers of VGG-net [38] including high-level layers, which encodes the semantic information, and low-level layers which preserve the spatial details. [8] uses a similar approach by using the activation of the 1st layer (low-level) and 5th layer (high-level) of VGG-M[4] as the tracking features. [33] uses the features extracted on different layers to build correlation filters and then combine these weak trackers into a strong one for tracking. In these three methods, CNNs are solely treated as a feature extractor, and they all adopt correlation filter as their base tracker. In addition, [19] utilizes Support Vector Machine (SVM) to model the target appearance based on the extracted CNN features.

Because the labeled training data for tracking is limited, online training a convolution neural networks is prone to overfitting, which makes it becomes a challenging task. [43] first pre-trains a neural network to distinguish objects from non-objects on ImageNet [25], and then updates this model online with a differentially-paced fine-tuning scheme. A recent proposed tracker [31] trained a multi-domain network, which has shared CNN layers to capture a generic feature represetnation, and separate branches of subsequent domain-specific layers to do the binary classification (target vs. background) for each sequence. For each sequence, the domain-specific layers must be fine-tuned to learn the target. In contrast to these CNN based methods that require running back-propagation to online train the network during tracking, our approach does not require online training, and instead uses a recurrent network to update the target appearance model with each frame. As a result, our tracker is faster than the CNN-based trackers that use online training.

An alternative approach to target classification is to train a similarity function for pairs of images, which regards visual tracking as an instance searching problem, i.e. using the target image patch on first frame as query image to search the object in the subsequent frames. Specifically, [41] adopts a Siamese neural networks for tracking, which is a two-stream architecture, originally used for signature and face verification [5], and stereo matching [45]. [17] proposed a similar framework called GOTURN, which is trained to regress the target's position and size directly by inputting the network with a search image (current frame) and a query image (previous frame) that contains the target. [3] introduced a fully-convolutional Siamese network for visual tracking, which maps an exemplar of the target and a larger search area of second frame to a response map. In contrast to these methods, which do not have an online updating scheme that adapts the tracker to variations in the target appearance, our approach take the advantage of the LSTM's ability to capture and remember the target's appearance.

2.2. RNN-based Trackers

Recurrent Neural Networks (RNNs), especially its variants Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU) are widely used in applications involve sequential data, such as language modeling [28], machine translation [40], handwriting generation and recognition

AIRSPACE SYS. INC. EX2047, Page 2 (IPR2025-00629)

[14], video analysis [10] and image captioning [42]. Several recent works propose to solve the visual tracking problem as a sequential position prediction process by training an RNN to model the temporal relationship among frames. [12] adopts a GRU to model the features extracted from a CNN, while [21] embeds an attention mechanism into the RNN to guide target searching. However these two methods mainly focus on conducting experiments on synthesized data and have not demonstrate competitive results on commonly used tracking benchmark like OTB [44] and VOT [23]. [32] employs the recent efficient and effective object detection method, YOLO [34], to infer the preliminary position, and then use LSTM to estimate the target's bounding box. A heat map, constructed using the detected box from YOLO, is also used to help the RNN to distinguish object from non-object region. However, they only choose 30 sequences from OTB dataset for experiments, among which some of them are used as training data while other are used for testing.

The major limitation of the aforementioned methods is that they directly feed the output of the fully connected layers into the original RNN or its variants (LSTM or GRU), which obfuscates the spatial information of target. In contrast, our approach better preserves the target's spatial structure, by changing the matrix multiplication in the fully-connected layers of the LSTM into convolution layers on feature maps. As a result, our approach better preserves the spatial information of the target, while also being more memory-efficient. Furthermore, most existing RNN-based trackers estimate the target's bounding box by directly regressing it from the RNN hidden state, which is not effective because it requires enumerating all possible locations and scales of the target during training. In contrast, our approach is based on estimating a response map of the target, which makes the training simpler and more effective.

In contrast to modeling the temporal information of sequences as in [12, 21, 32], [6] uses RNN to capture the object's intrinsic structure by traversing a candidate region of interest from different directions. [11] adopts similar strategy to capture spatial information of the target and incorporates it into MDNet [31] to improve its robustness. In addition, a few recent works on multiple person detection [39] and tracking [29] use RNNs to model the sequential relationship among different targets.

3. Recurrent Filter Learning

This section describes our proposed filter generation framework for visual tracking.

3.1. Network Architecture

The architecture of our networks is shown in Figure 1. We design a recurrent fully convolutional network that extracts features from the object exemplar image and then generates a correlation filter, which is convolved with features extracted from the search image, resulting in a response map that indicates the position of the target. The network uses a convolutional LSTM to store the target appearance information from previous frames.

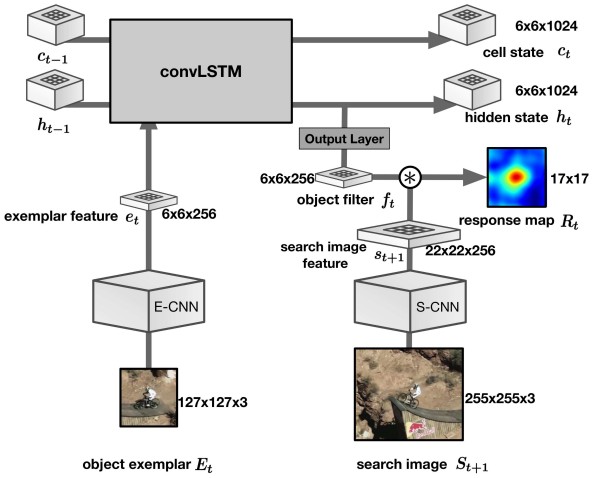

Figure 1. The architecture of our recurrent filter learning networks. At time step t, a CNN (E-CNN) extracts features from the exemplar image patch. Using the previous hidden and cell states, h_{t-1} and c_{t-1}, as well as the current exemplar feature map e_t, the convolutional LSTM memorizes the appearance information of the target by updating its cell and hidden states, c_t and h_t. The target object filter f_t is generated by passing the new hidden state h_t through an output convolutional layer. A feature map s_{t+1} is extracted from the searching image (next frame) using another CNN (S-CNN), which is convolved by the target object filter f_t, resulting in a response map that is used to locate the target.

Our framework uses two CNN feature extractors: 1) the E-CNN is used to capture an intermediate representation of the object exemplar for generating the target filter; 2) the S-CNN is used to extract a feature map from the search image, which is convolved with the generated filter to produce the response map. The E-CNN and S-CNN have the same architecture with different input image size, 127x127x3 and 255x255x3, respectively. As stated in [31], relatively smaller CNN are more appropriate for visual tracking because deeper and larger CNNs are prone to overfitting and dilute the spatial information. Therefore, for the two CNN feature extractors, we adopt a similar network as in [3], which has five convolutional layers (see Table 1). Note that we do not share the parameters between these two CNNs because the optimal features used to generate the target filter could be different from the features needed to discriminate the target from the background. Indeed experiments in Section 5.3 show that sharing the CNNs' parameters decreases the performance.

AIRSPACE SYS. INC. EX2047, Page 3 (IPR2025-00629)

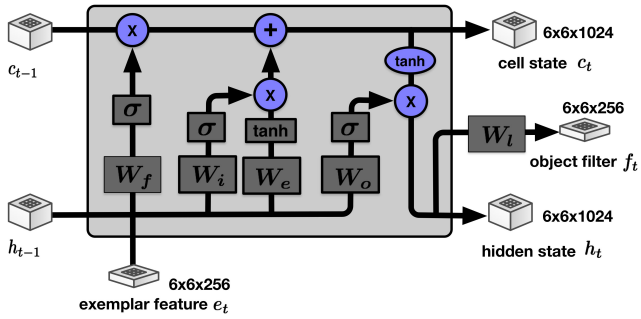

Figure 2. The architecture of convolutional LSTM. σ represents the sigmoid activation function and **tanh** denotes the tanh activation function. W_f, W_i, W_e, W_o are the convolutional filter weights for the forget gate, input gate, estimated cell state, and output gate. All these convolutional filters have the same size 3x3x1024. W_l is the convolutional weight of output layer with size 1x1x256.

	filter size	channels	stride
conv1	11x11	96	2
pool1	3x3	-	2
conv2	5x5	256	1
pool2	3x3	-	2
conv3	3x3	384	1
conv4	3x3	384	1
conv5	3x3	256	1

Table 1. Architecture of the two CNN feature extractors.

The architecture of the convolutional LSTM is shown in Figure 2. The hidden h_t and cell states c_t are feature maps that are each 6x6x1024. The output of the E-CNN is a 6x6x256 convolutional feature map e_t, which is actually a grid of 256 dimensional feature vectors. Within the conv-LSTM, the exemplar feature map e_t is concatenated to the previous hidden state h_{t-1} along the channel dimension. The various gates and update layers of the LSTM are implemented using convolutional layers with 3x3 filters, which helps to capture spatial relationships between features. Note that zero-padding is used in the convolution layers so that the hidden state and cell state have the same size with exemplar feature map. The target filter f_t is generated via the output layer of the LSTM, which is a convolution layer with a 1x1 filter that transforms the hidden state from 6x6x1024 to 6x6x256.

Finally, to find the target in the next frame, the search image patch is input to the S-CNN to extract the search image feature map s_{t+1}, which is then convolved with the generated filter f_t to produce a target response map. Batch normalization [20] is employed after each linear convolution to accelerating the converge of our network. Each convolutional layers except conv5 is followed by *Rectified linear units (ReLu)*. *Sigmoid* functions are used after all gate convolution layers, and *tanh* activation functions are used after the convolution layer for the estimated cell state. There is no activation function for the output layer of the LSTM.

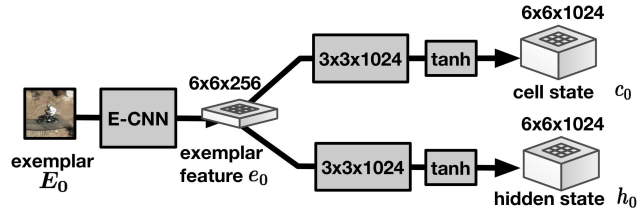

Figure 3. Initialization network for memory units of the convolutional LSTM.

3.2. Initialization of Memory States

To start the recurrent filter learning process, we need to first initialize the memory state of the convLSTM using the exemplar target from the first frame. The initial exemplar target E_0 is input into the E-CNN to generate an exemplar feature map e_0, which is then fed into a convolutional layer to produce the initial hidden state h_0 (see Figure 3). The convolutional layer has filter size of 3x3, and output channel size of 1024. A similar architecture is used to produce the initial cell state c_0 from the exemplar feature map e_0. Here we use a tanh activation function after the convolution. Although the numerical value of cell state may be out of the range [-1,1] (the output range of tanh) due to the addition operation between the input state and previous state, we experimentally find that this initialization method captures the information of initial target well. Our initialization network obtains about 10% performance gain compared with initializing the cell state to zeros (see Section 5.3).

3.3. Loss Function

The response map represents the probability of the target position on a 17x17 grid placed evenly around the center square of search image, with stride of 8. A loss function is applied on the response map to train the network. In particular, we generate a ground-truth response map, which consists of binary values that are computed from the overlap ratio between the ground-truth bounding box and a bounding box centered at each position,

$$y_i = \begin{cases} 1, & \text{if } \Phi(P_i, G) > \alpha \\ 0, & \text{otherwise} \end{cases} \tag{1}$$

where, P_i is a virtual bounding box centered on the i-th position of the response map, G is the ground-truth bounding box, $\Phi(P, G) = \frac{P \cap G}{P \cup G}$ is the Intersection-over-Union (IoU) between regions P and G, and $\alpha = 0.7$ is the threshold.

The loss between the predicted response map and ground-truth reponse map is the sum of the element-wise

AIRSPACE SYS. INC. EX2047, Page 4 (IPR2025-00629)

sigmoid cross-entropies,

$$L(p, y) = \sum_i -(1 - y_i)\log(1 - p_i) - y_i \log(p_i), \quad (2)$$

where p is the predicted response map, and y is the ground-truth.

3.4. Data and Training

ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [36] recently released a video data for evaluating object detection from video. This dataset contains more than 4000 sequences, where ∼3800 are used for training and ∼500 for validation. The dataset contains 1.3 million labeled frames, and nearly two million object image patches marked with bounding boxes. This large dataset can be safely used to learn neural networks without worrying about overfitting, compared with traditional tracking benchmarks like OTB [44] and VOT [23]. As is discussed in [3], the fairness of using sequences from the same domain to train and test neural networks for tracking is controversial. We instead train our model from the ImageNet domain to show our network generalization ability.

The input of our framework is a sequence of cropped image patches. During training, we uniformly sample N+1 frames from each video. Frames 1 through N are used to generate the object exemplars, and frames 2 to N+1 are the corresponding search image patches. Both the object exemplar and search image patches are cropped around the center of groundtruth bounding box. The exemplar size is two times that of the target, and the search image size is four times the target size, as is shown in Figure 4. The reason that the exemplar size is larger than the target size is to cover some context information, which may provide some negative examples when generating the filter. If the patch extends beyond the image boundary, we pad using the mean RGB value of the image. Note that our architecture is fully convolutional, which is translation invariant. Hence, it suffices to train the network using search images where the targets are always located at the center.

3.5. Training Details

The recurrent model is trained by minimizing the loss in Eq. 2 using the ADAM [22] method with a mini-batches of 10 video clips of length 10. The initial learning rate is 1e-4, which is decayed by a multiplier 0.8 every 5000 iterations. In case of gradient explosion, gradients are clipped to a maximum magnitude of 10.

The exemplar and search image patches are preprocessed by resizing them to their corresponding CNN input sizes. We use various forms of data augmentation. Following [25], we randomly distort colors by randomly adjusting image brightness, contrast, saturation and hue. In addition, we apply minor image translation and stretching

Figure 4. Samples of target exemplar patches (1st and 3rd rows) and search images (2nd and 4th rows), from VID of ILSVRC. Targets are marked by a red bounding box on each search image.

to both the exemplar and search images. Search images are shifted around the center with a range of 4 pixels (within the stride of the response map grid). The target in exemplar patches can be translated anywhere as long as its bounding box is within the exemplar patch. For stretching, we randomly crop the image patches, and resize them into their network input size. We set a small maximum stretch range (0.05 times of the target size) for search images, and larger stretch range (0.5 times of the target size) for exemplar patch. This setting accounts for the situations where the estimated target position and scale may not be very accurate, and hence the target in the exemplar is shifted from the center or stretched due to resizing. The first frame of the clip for exemplar has no data augmentation since we assume that the initial bounding box is always accurate in visual tracking. Although the object in the exemplar patch may appear in various places within the patch because of augmentation, the memory unit of our model can still capture the information of target due to its convolutional structure.

4. Online Tracking

We use a simple strategy for online tracking with our RFL framework, which is presented in Algorithm 1. Unlike other CNN-based algorithms [31, 30, 43], we do not online fine-tune the parameters of our networks using SGD back-propagation, as the target appearance is represented by the hidden and cell states of the conv-LSTM. Furthermore, we do not refine our predictions using bounding box regression as in [31, 41, 11]. Instead, we directly upsample the response map into a larger size using bicubic interpolation as in [3], and choose the position with maximum value as the target's location. To account for scale changes, we calculate response maps at different scales by building a pyramid of

AIRSPACE SYS. INC. EX2047, Page 5 (IPR2025-00629)

scaled search images, resizing them to the same input size, and then assembling them into a mini-batch to be processed by the network in a single feed-forward pass.

During tracking, we first use the target exemplar to initialize the memory state of LSTM using the initialization network (Figure 3). For the subsequent frames, we convolve the filter generated by the conv-LSTM with the search images, which are centered at the previous predicted position of object, to get the response map. Let $\{S_t^{-1}, S_t^0, S_t^1\}$ be three scales of the search image. Using the S-CNN, we obtain the corresponding feature maps $\{s_t^{-1}, s_t^0, s_t^1\}$. The response map for scale m is calculated as,

$$R^m(f_t, s_{t+1}) = f_t * s_{t+1}^m. \tag{3}$$

Let v^m be the maximum score of the response map R^m at scale m. Then, the predicted target scale is the scale with largest maximum score,

$$m_{best} = \underset{m}{\operatorname{argmax}}\, v^m \tag{4}$$

The predicted target position is obtained by averaging positions with the top K response values on the score map $R^{m_{best}}$ of the predicted scale,

$$p^* = \frac{1}{K}\sum_k^K p_k \tag{5}$$

where p_k are the top K locations.

After we get the estimated position, a target exemplar is cropped from the frame, and used to update the conv-LSTM. As seen in recent works [41, 7], excessive model updating is prone to overfitting to recent frames. Hence, we dampen the memory states of the conv-LSTM using exponential decay smoothing, $A_{est} = (1-\beta) * A_{old} + \beta A_{new}$, where β is the decay rate and A is the memory state. Furthermore, we also penalize the score map of other scales (scales ± 1) by multiplying those score maps by a penalty factor γ. To smoothen the scale estimation and penalize large displacements, we use exponential decay smoothing.

4.1. Tracking Parameters

We adopt three scales $1.03^{[-1,0,1]}$ to build the scale pyramid of search images. The penalty factor on score maps of non-original scales is $\gamma = 0.97$. Object scale is damped with decay rate of $\beta = 0.6$, while memory states are damped with a decay rate of $\beta = 0.06$. Score map is damped with a cosine window by the decay rate of $\beta = 0.11$. What' more, the number of candidate position used for averaging final prediction is $K = 5$.

5. Experiments

We evaluate our proposed RFL tracker on OTB100 [44], VOT2016 [24] and VOT2017. We implement our algorithm in Python using the TensorFlow toolbox [1], and test it on a computer with four Intel(R) Core(TM) i7-6700 CPU @ 3.40GHz and a single NVIDIA GTX 1080 with 8G RAM. The running speed of our algorithm is about 15 fps.

Algorithm 1 Online Tracking Algorithm with RFL

Input:
Pretrained RFL model M;
Initial target exemplar E_0;
Output:
Predicted object's bounding box

1: Initialize the memory state of conv-LSTM using the initialization network on exemplar E_0;
2: Generate an initial filter f_0 using the conv-LSTM.
3: **for** $t = \{0, \cdots, T-1\}$ **do**
4: Get search image S_{t+1} from frame $t+1$.
5: Build a pyramid of scaled search images $\{S_{t+1}^{-1}, S_{t+1}^0, S_{t+1}^1\}$, and apply S-CNN on each scaled image to produce feature maps $\{s_{t+1}^{-1}, s_{t+1}^0, s_{t+1}^1\}$.
6: Convolve the filter f_t with $\{s_{t+1}^{-1}, s_{t+1}^0, s_{t+1}^1\}$ as in Eq. 3.
7: Upsample the responses using bicubic interpolation and penalize scale changes.
8: Normalize and apply cosine window on score maps $\{R_{t+1}^{-1}, R_{t+1}^0, R_{t+1}^1\}$.
9: Estimate the target's scale by Eq. 4.
10: Predict the target's position by Eq. 5.
11: Crop a new target exemplar E_{t+1} centered at the estimated position in frame $t+1$.
12: Update the conv-LSTM using exemplar E_{t+1}.
13: Generate a new filter f_{t+1} from the conv-LSTM.
14: **end for**

5.1. Tracking Results on OTB100

There are two common performance metrics for evaluating tracking, center location error (CLE) and Intersection-over-Union (IoU). Specifically, CLE measures the pixel distance error between the predicted and groundtruth center positions. IoU measures the overlap ratio between the predicted and ground-truth bounding boxes. However, as discussed in [44], CLE only evaluates the pixel difference, which is actually not fair for small objects, and does not consider the scale and size changes of objects. We thus use IoU success plots as our evaluation metric on OTB.

We compare our tracking method with six other state-of-the-art methods, including SiamFC [3], CF2 [27], HDT [33], MEEM [46], DSST [9], and KCF [18] on OTB100. We present the OPE success plots of in Figure 5 (Left), where our method is denoted as RFL. Trackers are ranked by their *area under curve* (AUC) score. Overall, SiamFC has slightly higher AUC (ours .581 vs .582). Looking

AIRSPACE SYS. INC. EX2047, Page 6 (IPR2025-00629)

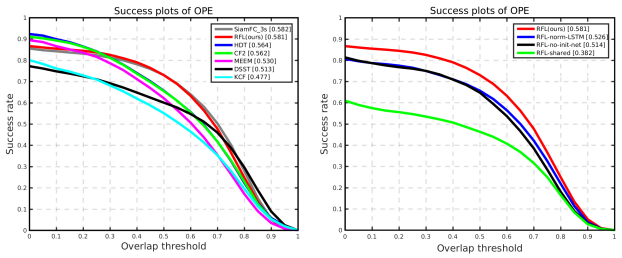

Figure 5. Left: Success plots of OPE on OTB100 compared with seven trackers. Right: Success plot of OPE on OTB100 for different variants of proposed RFL. Trackers are ranked by their AUC scores in the legend. Our proposed method is RFL.

Tracker	AUC	P@0.3	P@0.5	P@0.7
RFL	0.581	**0.825**	**0.730**	0.477
SiamFC_3s	**0.582**	0.812	**0.730**	**0.503**
HDT	0.564	0.812	0.657	0.417
CF2	0.562	0.811	0.655	0.417
MEEM	0.530	0.786	0.622	0.355
DSST	0.513	0.691	0.601	0.460
KCF	0.477	0.681	0.551	0.353

Table 2. AUC and success rates at different IoU thresholds.

closer (see Figure 7), our RFL has higher AUC on videos where the target or background appearance changes compared with SiamFC: fast motion (ours .602 vs .568), background clutter (ours .575 vs .523), out-of-view (ours .532 vs .506). I.e. our RFL can better adapt the tracking filter over time to better discriminate the changing target/background.

We also report the average success rates at various thresholds in Table 2. In addition to our superiority in AUC score, our method's success rates at IoU=0.3 and IoU=0.5 is much higher than other methods or on par with SiamFC, which suggests that our RFL predicts a more accurate bounding box than other trackers.

We further analyze the performance of our method under different sequence attributes annotated in OTB. Figure 7 shows the OPE results comparison of 6 challenge attributes: *out-of-view, scale variation, in-plane rotation, illumination variation, background clutter and fast motion*. Our RFL tracker shows better or similar performance compared with other methods. Note that our method works especially well under the *fast motion* and *out of view* attribute. Figure 8 shows some example frames of the results generated by our RFL and the other six trackers on several challenging sequences (see supplemental for vidoes). From these videos, we can see that our method handles fast motion, scale changes and illumination variation better than others.

5.2. Tracking Results on VOT2016 and VOT2017

We presents a comparison of our trackers with 7 state-of-the-art trackers, including DSST [9], SiamFC [3], SAMF

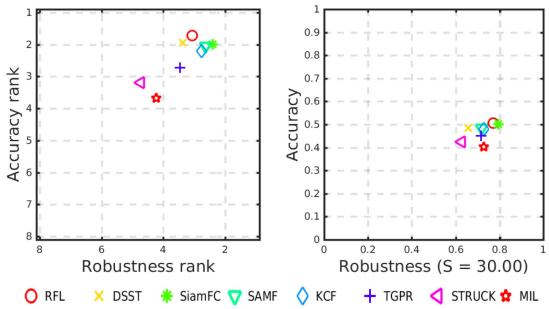

Figure 6. A state-of-the-art comparison on the VOT2016 benchmark. Left: Ranking plot includes the accuracy and robustness rank for each tracker. Right: AR plot shows the accuracy and robustness scores

Tracker	Accuracy	Robustness	EAO
RFL	**1.72**	3.07	0.2222
DSST	1.95	3.38	0.1814
SiamFC_3s	2.00	**2.42**	**0.2352**
SAMF	2.05	2.65	0.1856
KCF	2.20	2.77	0.1924
TGPR	2.73	3.47	0.1811
STRUCK	3.20	4.73	0.1416
MIL	3.68	4.23	0.1645

Table 3. AR ranking and EAO for each tracker.

[26], KCF [18]TGPR [13], STRUCK [15] and MIL [2] on VOT2016 [24]. Two performance metrics, which are *accuracy* and *robustness*, are used to evaluate our algorithm. The *accuracy* measures the averaged per-frame overlap between predicted bounding box and ground truth, while *robustness* reveals the number of failures over sequences. We generate these results using the VOT-toolkit. Figure 6 visualizes independent ranks for each metric on two axes. Table 3 reports the AR ranking (Accuracy and Robustness) and EAO (expected averaged overlap) for each tracker. Our RFL tracker achieves the best accuracy rank and shows competitive EAO compared with other trackers.

We also evaluate our method on VOT2017 challenge. In baseline experiment, we obtain an average overlap of 0.48 and failure of 3.29. In unsupervised experiment, we get an average overlap of 0.23. Our average speed in realtime experiment is 15.01 fps.

5.3. Experiments on Architecture Variations

To show the effectiveness of our recurrent filter learning framework, we also conduct comparisons among three variations of our framework:

- Using shared the weights for E-CNN and S-CNN (denoted as *RFL-shared*).
- Initializing the memory states to all zeros, rather than

AIRSPACE SYS. INC. EX2047, Page 7 (IPR2025-00629)

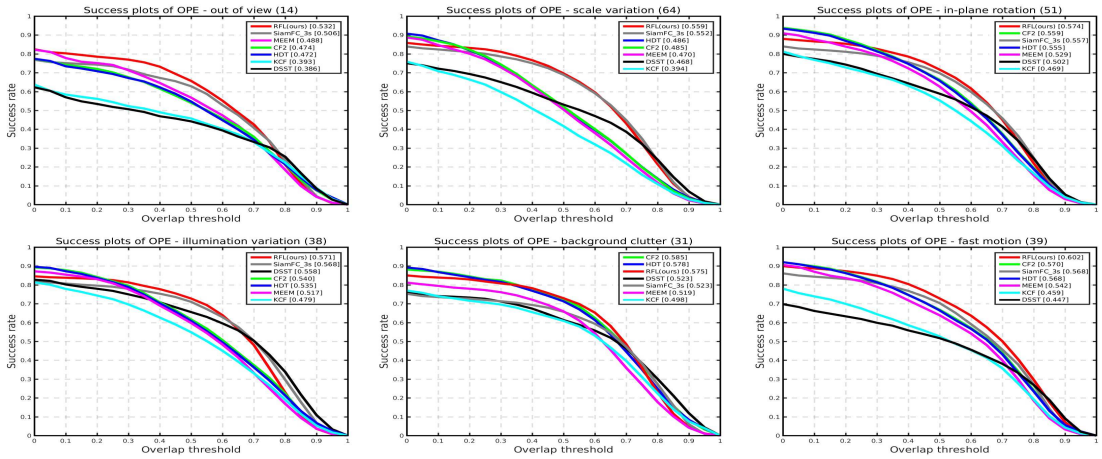

Figure 7. Success plot for 6 challenge attributes: out-of-view, scale variation, in-plane rotation, illumination variation, background clutter and fast motion. Trackers are ranked by their AUC scores in the legend.

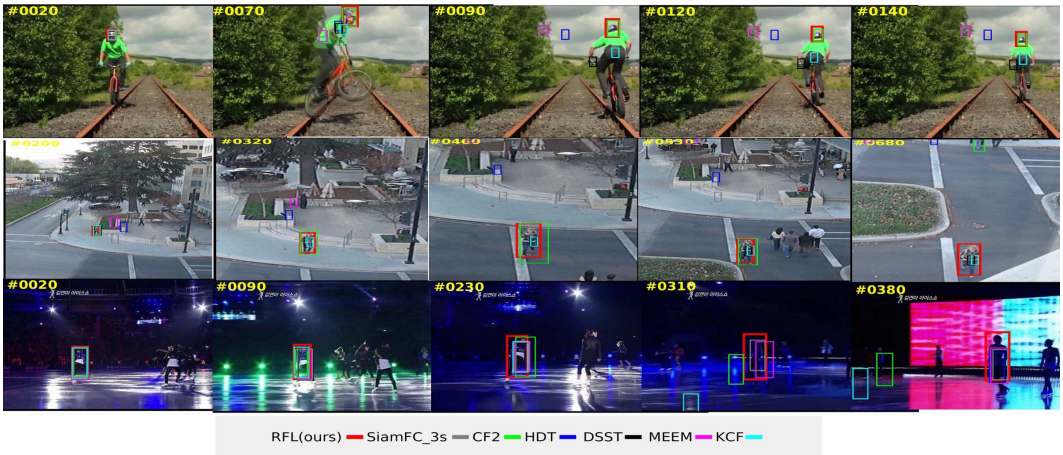

Figure 8. Qualitative results of our method on some challenging videos (from top to botton, *biker, human5, skating1*)

using the intialization network (denoted as *RFL-no-init-net*).

- Use a normal LSTM on each channel vector of the feature map, i.e. by setting the filter sizes in the convolutional LSTM to 1x1 (denoted as *RFL-norm-LSTM*)

Figure 5 (Right) shows the success plot of OPE on OTB100 for different variants of RFL. Sharing parameters between the E-CNN and S-CNN significantly decreases the tracking performance. Also, we are able to obtain about 8-10% performance gain by using the initialization network for the memory state and a convolutional LSTM.

6. Conclusion

In this paper, we have explored the effectiveness of generating tracking filters recurrently by sequentially feeding target exemplars into a convolutional LSTM for visual tracking. To best of our knowledge, we are the first to demonstrate improved and competitive results on large-scale tracking datasets (OTB100, VOT2016 and VOT2017) using RNN to model temporal sequences. Instead of initializing and updating the neural network using time-consuming SGD back-propagation for each specific video as in other CNN-based methods [31, 30, 43], our tracker estimates the target filter using only feed-forward computation. Extensive experiments on well-known tracking benchmarks have validated the efficacy of our RFL tracker.

Acknowledgements

This work was supported by the Research Grants Council of the Hong Kong Special Administrative Region, China (CityU 11200314), and by a Strategic Research Grant from City University of Hong Kong (Project No. 7004682). We gratefully acknowledge the support of NVIDIA Corporation with the donation of the Tesla K40 GPU used for this research.

AIRSPACE SYS. INC. EX2047, Page 8 (IPR2025-00629)

References

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv*, 2016. 6

[2] B. Babenko, S. Member, M.-h. Yang, and S. Member. Robust Object Tracking with Online Multiple Instance Learning. *PAMI*, 2011. 1, 7

[3] L. Bertinetto, J. Valmadre, J. F. Henriques, A. Vedaldi, and P. H. S. Torr. Fully-Convolutional Siamese Networks for Object Tracking. In *ECCVW*, 2016. 2, 3, 5, 6, 7

[4] K. Chatfield, K. Simonyan, A. Vedaldi, and A. Zisserman. Return of the Devil in the Details: Delving Deep into Convolutional Nets. In *BMVC*, 2014. 2

[5] S. Chopra, R. Hadsell, and Y. LeCun. Learning a similarity metric discriminatively, with application to face verification. In *CVPR*, volume 1, pages 539–546. IEEE, 2005. 2

[6] Z. Cui, S. Xiao, J. Feng, and S. Yan. Recurrently Target-Attending Tracking. In *CVPR*, pages 1449–1458, 2016. 3

[7] M. Danelljan, G. Bhat, F. S. Khan, and M. Felsberg. ECO: Efficient Convolution Operators for Tracking. *arXiv*, 2016. 6

[8] M. Danelljan, H. Gustav, F. S. Khan, and M. Felsberg. Convolutional Features for Correlation Filter Based Visual Tracking. In *ICCVW*, pages 58–66, 2015. 1, 2

[9] M. Danelljan, G. Häger, F. Khan, and M. Felsberg. Accurate Scale Estimation for Robust Visual Tracking. In *BMVC*, 2014. 6, 7

[10] J. Donahue, L. Anne Hendricks, S. Guadarrama, M. Rohrbach, S. Venugopalan, K. Saenko, and T. Darrell. Long-term recurrent convolutional networks for visual recognition and description. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2625–2634, 2015. 3

[11] H. Fan and H. Ling. SANet: Structure-Aware Network for Visual Tracking. *arXiv*, 2016. 3, 5

[12] Q. Gan, Q. Guo, Z. Zhang, and K. Cho. First Step toward Model-Free, Anonymous Object Tracking with Recurrent Neural Networks. *arXiv*, pages 1–13, 2015. 3

[13] J. Gao, H. Ling, W. Hu, and J. Xing. Transfer Learning Based Visual Tracking with Gaussian Processes Regression. In *ECCV*, volume 8691, pages 188–203, 2014. 7

[14] A. Graves. Generating sequences with recurrent neural networks. *arXiv*, 2013. 3

[15] S. Hare, S. Golodetz, A. Saffari, V. Vineet, M.-M. Cheng, S. Hicks, and P. Torr. Struck: Structured Output Tracking with Kernels. In *ICCV*, pages 1–1, 2011. 7

[16] K. He, X. Zhang, S. Ren, and J. Sun. Deep Residual Learning for Image Recognition. In *CVPR*, volume 7, 2016. 2

[17] D. Held, S. Thrun, and S. Savarese. Learning to Track at 100 FPS with Deep Regression Networks. In *ECCV*, 2016. 2

[18] J. F. Henriques, R. Caseiro, P. Martins, and J. Batista. High-speed tracking with kernelized correlation filters. *TPAMI*, 2015. 1, 6, 7

[19] Z. Hong, Z. Chen, C. Wang, X. Mei, D. Prokhorov, and D. Tao. MUlti-Store Tracker (MUSTer): a Cognitive Psychology Inspired Approach to Object Tracking. In *CVPR*, 2015. 1, 2

[20] S. Ioffe and C. Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In *ICML*, pages 1–11, 2015. 4

[21] S. E. Kahoú, V. Michalski, and R. Memisevic. RATM: Recurrent Attentive Tracking Model. In *ICLR*, pages 1–9, 2016. 3

[22] D. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv*, 2014. 5

[23] M. Kristan, J. Matas, A. Leonardis, M. Felsberg, L. Cehovin, G. Fernández, T. Vojir, G. Hager, G. Nebehay, and R. Pflugfelder. The visual object tracking vot2015 challenge results. In *ICCVW*, pages 1–23, 2015. 3, 5

[24] M. Kristan, R. Pflugfelder, A. Leonardis, J. Matas, L. Čehovin, G. Nebehay, T. Vojí\vr, G. Fernandez, and Others. The visual object tracking VOT2016 challenge results. In *ECCVW*, 2016. 2, 6, 7

[25] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, pages 1097–1105, 2012. 2, 5

[26] Y. Li and J. Zhu. A Scale Adaptive Kernel Correlation Filter Tracker with Feature Integration. In *ECCV*, 2014. 7

[27] C. Ma, J.-b. Huang, X. Yang, and M.-h. Yang. Hierarchical Convolutional Features for Visual Tracking. In *ICCV*, pages 3074–3082, 2015. 1, 2, 6

[28] T. Mikolov, M. Karafiát, L. Burget, J. Cernockỳ, and S. Khudanpur. Recurrent neural network based language model. In *Interspeech*, volume 2, page 3, 2010. 2

[29] A. Milan, S. H. Rezatofighi, A. Dick, K. Schindler, and I. Reid. Online Multi-target Tracking using Recurrent Neural Networks. *arXiv*, 2016. 3

[30] H. Nam, M. Baek, and B. Han. Modeling and Propagating CNNs in a Tree Structure for Visual Tracking. *arXiv*, 2016. 1, 5, 8

[31] H. Nam and B. Han. Learning Multi-Domain Convolutional Neural Networks for Visual Tracking. In *CVPR*, 2016. 1, 2, 3, 5, 8

[32] G. Ning, Z. Zhang, C. Huang, Z. He, X. Ren, and H. Wang. Spatially Supervised Recurrent Convolutional Neural Networks for Visual Object Tracking. *arXiv*, 2016. 3

[33] Y. Qi, S. Zhang, L. Qin, H. Yao, Q. Huang, and J. L. M.-h. Yang. Hedged Deep Tracking. In *CVPR*, 2016. 1, 2, 6

[34] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. You only look once: Unified, real-time object detection. In *CVPR*, 2016. 2, 3

[35] S. Ren, K. He, R. Girshick, and J. Sun. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. In *NIPS*, 2015. 2

[36] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, et al. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015. 5

[37] X. Shi, Z. Chen, H. Wang, D.-Y. Yeung, W.-k. Wong, and W.-c. Woo. Convolutional LSTM network: A machine learning approach for precipitation nowcasting. *NIPS*, pages 802–810, 2015. 2

AIRSPACE SYS. INC. EX2047, Page 9 (IPR2025-00629)

[38] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In *ICLR*, 2015. 2

[39] R. Stewart and M. Andriluka. End-to-end people detection in crowded scenes. In *CVPR*, page 9, 2016. 3

[40] I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014. 2

[41] R. Tao, E. Gavves, and A. W. M. Smeulders. Siamese Instance Search for Tracking. In *CVPR*, pages 1420–1429. 2, 5, 6

[42] O. Vinyals, A. Toshev, S. Bengio, and D. Erhan. Show and tell: A neural image caption generator. In *CVPR*, pages 3156–3164, 2015. 3

[43] N. Wang, S. Li, A. Gupta, and D.-Y. Yeung. Transferring Rich Feature Hierarchies for Robust Visual Tracking. In *ICCV*, 2015. 2, 5, 8

[44] Y. Wu, J. Lim, and M.-H. Yang. Object Tracking Benchmark. *PAMI*, 2015. 2, 3, 5, 6

[45] J. Zbontar and Y. LeCun. Computing the stereo matching cost with a convolutional neural network. In *CVPR*, pages 1592–1599, 2015. 2

[46] J. Zhang, S. Ma, and S. Sclaroff. MEEM: Robust Tracking via Multiple Experts Using Entropy Minimization. In *ECCV*, 2014. 1, 6

AIRSPACE SYS. INC. EX2047, Page 10 (IPR2025-00629)