

Learning Spatial-Aware Regressions for Visual Tracking

Chong Sun^{1,2}, Huchuan Lu¹, Ming-Hsuan Yang²

¹Dalian University of Technology

²University of California, Merced

Abstract

In this paper, we analyze the spatial information of deep features, and propose two complementary regressions for robust visual tracking. First, we propose a kernelized ridge regression model wherein the kernel value is defined as the weighted sum of similarity scores of all pairs of patches between two samples. We show that this model can be formulated as a neural network and thus can be efficiently solved. Second, we propose a fully convolutional neural network with spatially regularized kernels, through which the filter kernel corresponding to each output channel is forced to focus on a specific region of the target. Distance transform pooling is further exploited to determine the effectiveness of each output channel of the convolution layer. The outputs from the kernelized ridge regression model and the fully convolutional neural network are combined to obtain the ultimate response. Experimental results on three benchmark datasets validate the effectiveness of the proposed method.

1. Introduction

Visual tracking, which aims to continuously estimate the positions and scales of a pre-specified target, has been a hot topic for the last decades. It is widely used in numerous computer vision tasks, such as video surveillance, video based person re-identification and so on. Current algorithms have achieved very impressive performance, while at the same time, many problems remain to be solved.

With the emergence of large scale dataset, deep neural networks have shown their great capacity in object classification, image identification and so on. It has been verified in many prior papers [28, 24] that methods based on convolutional neural networks (CNNs) can greatly improve the tracking performance. Usually, these methods pretrain their networks on a large scale dataset, and finetune the networks with the ground-truth data in the first frame of a sequence. In addition to the CNN based trackers, methods based on the kernelized correlation filter (KCF) are also very popular in recent years for the efficiency and capacity to utilize large numbers of negative samples. As is described in [15], the

KCF method is essentially the kernelized ridge regression (KRR) with cyclically shifted samples. Methods based on the KCF usually take a region of interest as the input, which makes it very difficult to exploit the structural information of the target. In addition, the cyclically constructed samples also introduce the unwanted boundary effects. Compared to the KCF, the dominating reason why the conventional KRR is not widely applied is that it has to compute a kernel matrix for large numbers of samples, which results in heavy computation load.

Both the CNN and KRR (including correlation filters) based trackers have limitations and they have complementarities. The CNN based trackers usually contain tens of thousands of parameters which are difficult to be finetuned in the visual tracking problem. As a result, the trained filter kernels in convolution layer are usually highly correlated and tend to overfit the training data. On the contrary, the KRR based trackers have limited model parameters (equal to the number of training samples), and cannot learn discriminative enough models when training samples are correlated. In addition, the existing KRR based methods assume that each part of the target object is equally important, and do not consider the relationship among different parts. Though the CNN and KRR based methods have some complementarities, the performance is not satisfactory when two irrelevant models are directly combined (see Section 6.5 for experimental analysis).

In this paper, we exploit both the KRR and CNN as two complementary regressions for visual tracking, wherein the CNN focuses on the small localized region, and the KRR focuses on the holistic target. First, we propose a kernelized ridge regression with cross-patch similarities. We assign each similarity score a weight, and simultaneously learn this weight and ridge regression model parameters. We show that the proposed ridge regression model can be reformulated as a neural network, which is more efficiently optimized than the original form. Second, we introduce the spatially regularized kernels into the fully convolutional neural network. By imposing spatial constraints on the filter kernels, we force each output channel of the convolution layer to have response for a specific localized region. We exploit

the distance transform pooling layer to determine the effectiveness of the outputs from the convolution layer, and combine the responses from both regression models to obtain a ultimate response map. We exploit the two-stage update strategy to update the CNN model, by which means we first update the convolution layers and then update the distance transform pooling layers with the fixed convolution weights. Experiments on several datasets validate the effectiveness of the proposed method.

2. Related Work

Algorithms of visual tracking mainly focus on designing robust appearance models, which are roughly categorized as generative and discriminative models. With the progress of deep convolutional neural networks and correlation filters, discriminative appearance models are preferred in the recent papers.

Correlation filters have attracted more and more attentions for the advantages in efficiency and robustness. Bolme *et al.* [4] propose to exploit the correlation filter with minimum output sum of squared error (MOSSE) for visual tracking. As fast Fourier transform is used, the tracker runs at 669 frames per second (FPS) in the tracking process. In [14], Henriques *et al.* first incorporate kernel functions into the correlation filter, which is named as CSK. The CSK tracker can also be solved via fast Fourier transform, thus is efficient. Based on [14], the tracking method [15] further improves the CSK tracker by using the histogram of gradient (HOG). Ma *et al.* [22] exploit complementary nature of features extracted from three layers of CNN, and use the coarse-to-fine fashion for target searching. Based on [22], an online adaptive Hedge method [25] is proposed, which takes both the short-term and long-term regrets into consideration. In this tracker, they use the CF based tracker defined on a single CNN layer as an expert and learn the adaptive weights for different experts. Danelljan *et al.* propose several CF-based trackers which have very good performance. The method [7] tries to suppress the boundary effects of the correlation filter by multiplying the filter coefficients with a Gaussian shaped spatial regularization weights. This paper achieves very good performance even with hand-crafted features. Based on [7], the tracking method [8] proposes an adaptive decontamination method for the correlation filter, which adaptively learns the reliability of each training sample and eliminates the influence of contaminated ones. In [9], the learning process for the correlation filter is conducted in the continuous spatial domain of various feature maps, which incorporates the sub-pixel accurate information.

Compared to correlation filters, models based on CNNs also achieve good performance. In [21], a shallow network with two convolution layers is proposed, which learns the feature representation and classifier simultaneously. In [28],

Wang *et al.* transfer the model pre-trained on image classification dataset to visual tracking and propose to use the fully convolutional neural network for target location. The importance of each feature channel is estimated with the method in [20]. Wang *et al.* [29] propose a sequentially training fashion for neural network to avoid the overfitting problem. Tao *et al.* [27] train the Siamese deep neural network on large amounts of extra video sequences. In the tracking process, the method directly matches the patches between the first frame and the new frame, and select the best one as the target. In [24], the network containing both shared layers and domain specific layers is proposed, wherein the shared layers obtain the generic target representations and the domain specific layers are used for classification.

3. KRR with Cross-Patch Similarity

Given training samples $\{(\mathbf{x}_i, y_i)\}_{i=1}^N$ in frame t , the conventional ridge regression can be formulated as follows:

$$\mathbf{w}_t = \arg \min_{\mathbf{w}_t} \sum_i (y_i - \mathbf{w}_t^\top \mathbf{x}_i)^2 + \lambda \|\mathbf{w}_t\|^2, \quad (1)$$

where $\mathbf{x}_i \in \mathbb{R}^{d \times 1}$ denotes the feature vector for sample i , and y_i is the sample label. In Eq. 1, $\mathbf{w}_t \in \mathbb{R}^{d \times 1}$ denotes the linear weight, and λ denotes the trade-off parameter for the regularization term. The model parameter \mathbf{w}_t can be rewritten as the weighted sum of the training samples, $\mathbf{w}_t = \sum_{i=1}^N \alpha_i^t \mathbf{x}_i$, and thus Eq. 1 can be rewritten as

$$\alpha_t^* = \arg \min_{\alpha_t} \sum_{i=1}^N (y_i - \sum_{j=1}^N \alpha_j^t k_{ij})^2 + \frac{\lambda}{2} \sum_{i=1, j=1}^N \alpha_i^t \alpha_j^t k_{i,j}, \quad (2)$$

where α_i^t is the weight for sample i , $\alpha_t = [\alpha_1^t, \dots, \alpha_N^t]^\top \in \mathbb{R}^{N \times 1}$, and $k_{i,j}$ is the kernel value computed between features \mathbf{x}_i and \mathbf{x}_j .

The existing kernel definitions (*e.g.*, Gaussian kernel, linear kernel, *etc.*) do not fully consider the spatial layouts of the target, which limits the tracking performance. In this paper, we define a kernel function which considers the similarity of all pairs of patches between two samples. Specially, we divide each sample i into M patches, and obtain features for each patch m as \mathbf{x}_i^m , then the kernel value between sample i and j is computed as

$$k_{ij} = \sum_{m=1, n=1}^M \beta_{m,n}^t \mathbf{x}_i^m \mathbf{x}_j^n, \quad (3)$$

where $\beta_{m,n}^t$ denotes the weight for the similarity score between patches m and n . Note that the kernel definition (*i.e.* $k_{ij} = k_{ji}$) can be easily satisfied by computing $\beta_{m,n}^t =$

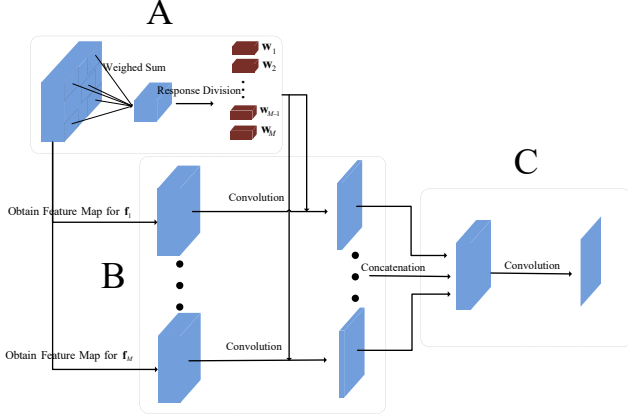


Figure 1. Structure of our reformulated network for kernelized ridge regression.

$\frac{1}{2}(\beta_{m,n}^t + \beta_{n,m}^t)$ after each update step of the model. Our kernel function has two advantages compared to the previous ones: 1. the kernel function assigns a weight to each similarity score, thus can adaptively focus on the similarity score between reliable regions through learning; 2. more similarity pairs between patches are considered, which enhances the discriminant ability of the model. In Figure 9 in the experimental section, we show that the proposed kernel function significantly improves the tracking performance.

By substituting Eq. 3 into Eq. 2, we obtain the following optimization problem:

$$\begin{aligned} \alpha_t^*, \beta_t^* = \arg \min_{\alpha_t, \beta_t} & \sum_{i=1}^N (y_i - \sum_{j=1}^N \alpha_j^t \sum_{m=1, n=1}^M \beta_{m,n}^t \mathbf{x}_j^m \mathbf{x}_i^n)^2 \\ & + \frac{1}{2} \lambda_1 \sum_{i=1, j=1}^N \alpha_i^t k_{i,j} \alpha_j^t + \frac{1}{2} \lambda_2 \sum_{m=1, n=1}^M \beta_{m,n}^t{}^2, \end{aligned} \quad (4)$$

where $\beta_t = [\beta_{1,1}^t, \dots, \beta_{M,M}^t]$. Eq. 4 can be equivalently written in the matrix form as

$$\begin{aligned} \alpha_t^*, \beta_t^* = \arg \min_{\alpha_t, \beta_t} & \left\| \mathbf{y} - \sum_{m=1, n=1}^M \mathbf{f}_m^\top \beta_{m,n}^t \mathbf{f}_n \alpha_t \right\|_2^2 \\ & + \frac{1}{2} \lambda_1 \alpha_t^\top \mathbf{K} \alpha_t + \frac{1}{2} \lambda_2 \|\beta_t\|_2^2, \end{aligned} \quad (5)$$

where $\mathbf{f}_m = [\mathbf{x}_1^m, \dots, \mathbf{x}_N^m]$ is the concatenated feature matrix for the m -th patches of N samples, $\mathbf{r} = \sum_{m=1, n=1}^M \mathbf{f}_m^\top \beta_{m,n} \mathbf{f}_n \alpha_t$ is the response of the proposed kernelized ridge regression.

A conventional solver for Eq. 4 is the alternating iteration algorithm which optimizes α_t and β_t iteratively. The analytical solution for α_t can be obtained as

$$\alpha_t = (\mathbf{K} + \lambda_1 \mathbf{I})^{-1} \mathbf{y}, \quad (6)$$

where \mathbf{K} is the kernel matrix of samples with its (i, j) -th element $k_{ij} = \sum_{m=1, n=1}^M \beta_{m,n}^t \mathbf{x}_i^m \mathbf{x}_j^n$. β_t can be updated

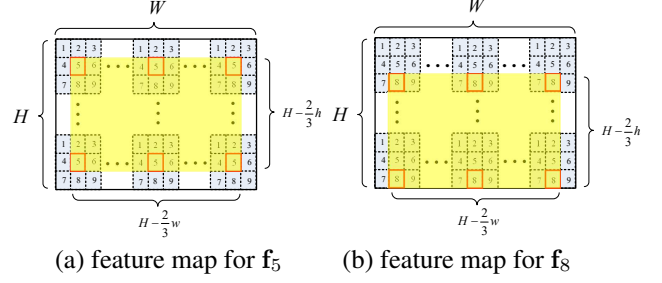


Figure 2. Feature maps corresponding to \mathbf{f}_5 and \mathbf{f}_8 are highlighted in yellow regions in (a) and (b). The $W \times H$ rectangle region denotes the entire input feature map, and the blue rectangle region denotes the target object with 9 patches. We obtain \mathbf{f}_m by dense sampling on the feature map corresponding to it.

via the gradient descent algorithm, and the gradient with respects to $\beta_{m,n}^t$ is computed as

$$\begin{aligned} \dot{\beta}_{m,n}^t = & -\mathbf{y}^\top \mathbf{f}_m^\top \mathbf{k}_n - \mathbf{k}_n^\top \mathbf{f}_m \mathbf{y} + 2\mathbf{k}_n^\top \mathbf{f}_m \left(\sum_{a,b} \mathbf{f}_a^\top \mathbf{k}_b \beta_{a,b} \right) \\ & + \lambda_2 \beta_{m,n}^t, \end{aligned} \quad (7)$$

where $\mathbf{k}_n = \mathbf{f}_n \alpha$ and $\dot{\beta}_{m,n}^t$ denotes the gradient with respects to $\beta_{m,n}^t$. The computation complexity for one step of update is $\mathcal{O}(dN^2 + N^3)$, which is too time consuming for the online update process.

In this paper, instead of exploiting the alternate iteration method, we try to learn α_t and β_t by reformulating the proposed ridge regression to a neural network. In Eq. 5, the response term $\mathbf{r} = \sum_{m=1, n=1}^M \mathbf{f}_m^\top \beta_{m,n}^t \mathbf{f}_n \alpha_t$ can be sequentially computed with the following three steps:

$$\begin{aligned} \text{A: } & \mathbf{w}_n^t = \mathbf{f}_n \alpha_t \\ \text{B: } & \mathbf{b}_{m,n} = \mathbf{f}_m^\top \mathbf{w}_n^t \\ \text{C: } & \mathbf{r} = \sum_{m,n} \beta_{m,n}^t \mathbf{b}_{m,n} \end{aligned} \quad (8)$$

Based on Eq. 8, we illustrate the equivalent neural network of our regression model as in Figure 1, which takes the pre-trained deep feature map as input and outputs a heat map for target localization. The network consists of three modules, each of which corresponds to one of the three operations in Eq. 8.

Module A. Given the target location, we first crop a rectangle region with twice the size of target object, and obtain the feature map \mathbf{X}_t with size $H \times W \times C$. Therefore, the target size projected on the feature map is $h \times w$, where $h = \frac{1}{2}H$ and $w = \frac{1}{2}W$. Based on the projected target size, we densely crop samples, and reshape each sample to a d ($d = h \times w \times C$) dimensional vector. This results in a feature matrix $\mathbf{D}_t \in \mathbb{R}^{d \times N}$, based on which we obtain the output of the weighted sum layer as

$$\mathbf{Z} = \mathbf{D}_t \alpha_t, \quad (9)$$

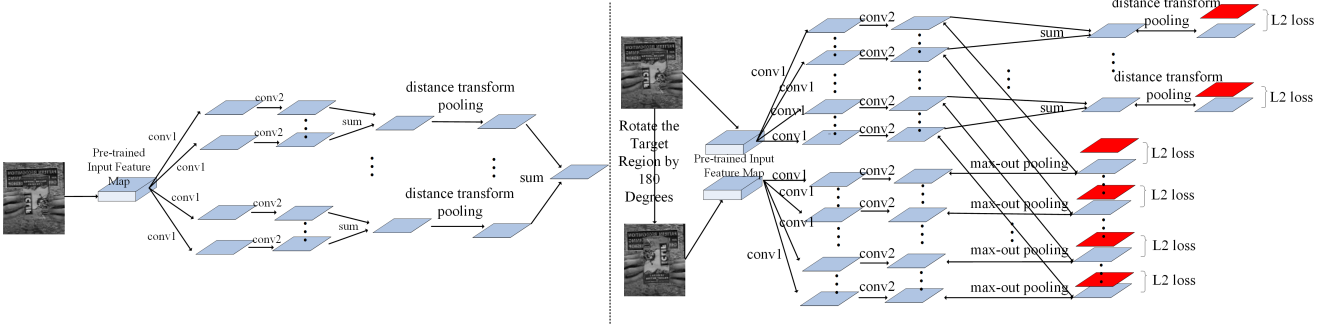


Figure 3. Network structures in the testing and training phases are presented in (a) and (b) respectively. (a) We use a convolutional neural network to estimate the target position, and exploit the distance transform pooling layer to determine the effectiveness of each response map. (b) We exploit the two-stage training strategy to update/train the convolution and distance transform pooling layers separately. A two-stream network is used to learn the rotation information of the target. The red rhombus is used to denote the gaussian shaped ground-truth.

where $\mathbf{Z} \in \mathbb{R}^{d \times 1}$ denotes the response for the current layer, α_t is the weight matrix to be learned. We reshape \mathbf{Z} to a response map with size $h \times w \times C$, and then divide it into $M = \sqrt{M} \times \sqrt{M}$ sub-responses with size $(h/\sqrt{M}) \times (w/\sqrt{M}) \times C$. We use $\mathbf{w}_1^t \dots \mathbf{w}_M^t$ to denote these sub-responses in Figure 1.

Module B. This module corresponds to the second equation of Eq. 8, which is equivalent to a convolution layer. In the reformulated network, we first obtain the feature map corresponding to \mathbf{f}_m (see Figure 2 for example) and feed it into a convolution layer which takes $\mathbf{w}_1^t \dots \mathbf{w}_M^t$ as an ensemble of filter kernels. As we have M patches in total, the convolution layer has M outputs, each of which has the size $(H - h + 1) \times (W - w + 1) \times M$.

Module C. We concatenate the M outputs of Module B through the Concat layer, and input the concatenated feature maps into a convolution layer, whose kernel size is $1 \times 1 \times M^2$. This module corresponds to operation C in Eq. 8, and the filter kernel corresponds to β_t .

We use the backpropagation algorithm to solve this network. The computation complexity for both forward and backward propagations is $\mathcal{O}(dN)$, which is much more efficient than the original solver. Note that the above network is defined based on Eq. 5 for model learning. At the detection stage in frame t , we just need to replace \mathbf{D}_t , α_t and β_t in the network with $\hat{\mathbf{D}}_t$, $\hat{\alpha}_t$ and $\hat{\beta}_t$ which are iteratively updated as

$$\begin{aligned} \hat{\mathbf{D}}_t &= \eta \hat{\mathbf{D}}_{t-1} + (1 - \eta) \mathbf{D}_{t-1} \\ \hat{\alpha}_t &= \eta \hat{\alpha}_{t-1} + (1 - \eta) \alpha_{t-1} \\ \hat{\beta}_t &= \eta \hat{\beta}_t + (1 - \eta) \beta_{t-1} \end{aligned} \quad (10)$$

where η is the update rate. In the following, we use Net-A to denote the reformulated neural network.

4. CNN with Spatially Regularized Kernels

As is described in previous papers (e.g., [1, 18]), the spatial information plays an important role in a visual tracking system. However, the spatial layouts of the target are usually ignored in the current tracking algorithms based on the convolutional neural network. What is more, the existing algorithms do not perform well when the target object has a severe in-plane rotation. In this work, we propose a convolution layer with spatially regularized filter kernels, through which each convolution kernel only focuses on a specific region of the target. In addition, as the training samples in visual tracking are very limited, we use different network structures for the training and testing processes to avoid overfitting, and consider rotation information when training the network. We name the network in the testing and training processes as Net-B and Net-C respectively. An overview of the two networks is visualized in Figure 3. Net-B consists of two convolution layers interleaved with an ReLU layer and several distance transform pooling layers. We use the conv4-3 layer of VGG-16 as the input feature map (reshaped to size $46 \times 46 \times 512$), following which we add a convolution layer (named as conv1) with spatially regularized filter kernels. This convolution layer has a kernel size of 5×5 , and outputs a feature map with size $46 \times 46 \times 100$. The second convolution layer (named as conv2) has a kernel size of 3×3 (we divide the input feature into 100 groups), and outputs a $46 \times 46 \times 100$ response map, each channel of which is a heat map for the target localization. We divide the response map into groups (25 groups in the implementation), and sum the responses in each group through the channel dimension, which results in a $46 \times 46 \times 1$ output in each group. This response map is fed into a distance transform pooling layer, which will be detailed later. For model learning, we exploit the architecture of Net-C to learn the weights of convolution and distance transform pooling layers. Based on the observa-

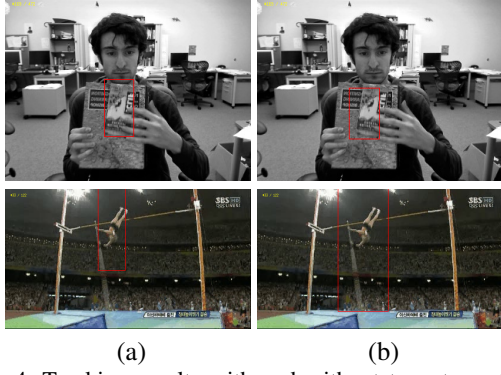


Figure 4. Tracking results with and without two-stream training process are illustrated in (a) and (b) respectively. The two-stream training process facilitates better tracking performance when severe in-plane rotation occurs.

tion that tracking performance is influenced when severe in-plane-rotation (rotation angle is larger than 90 degrees) occurs, we consider using the two-stream network to handle this problem with shared weights in the convolution layers. The upper branch of the network exploits the same feature map as Net-B, and the lower branch uses the input feature map corresponding to the rotated target object. The previous two layers are the same as Net-B, which results in two $46 \times 46 \times 100$ response maps in the upper and lower branches. We perform max-out pooling operation for the two response maps, and obtain a $46 \times 46 \times 100$ response. We compute the loss for each channel of the response map, and propagate the loss backward to learn the filters in convolution layers. Then we fix the convolution layer and learn the model parameters of the distance transform pooling layers.

With the two stream network, each output channel of conv2 only has response to one rotation angle of the target (see Figure 5 for example), thus is helpful when the object experiences a severe in-plane rotation (Figure 4). In addition, the reason why we do not directly perform model learning on Net-B is to avoid the overfitting problem. For example, Net-B has a total of $5 * 5 * 512 * 100 + 3 * 3 * 100 + 25 * 4 = 1281000$ parameters which is very difficult to train with limited training data in visual tracking. Net-C essentially decomposes the network into several sub-parts, and train each part separately to avoid overfitting.

4.1. Convolution Layer with Spatially Regularized Filter Kernels

The target object may sometimes experience deformations and occlusions, which makes parts of the target object more important than others. Wang *et al.* [29] revise the implementation of the Dropout layer, and keep the dropped activations fixed in the training process. By doing this, they force the learned convolution layers to focus on different parts of the input feature map. The limitation of this method

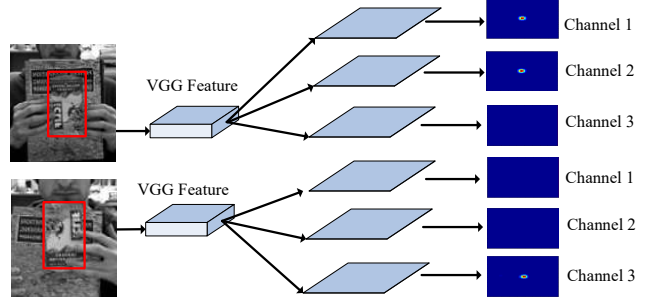


Figure 5. An example showing that each filter kernel in the convolution layer focuses on a specific rotation angle of the target. In this example, channels 1 and 2 correspond to the original target and channel 3 corresponds to the rotated target.

is that it only takes part of the input feature map for consideration in the training process, and has weak discriminant ability. In this work, instead of performing constraints on the input feature map, we propose to enforce constraints on the filter kernels in the convolution layer.

Let $\mathbf{F}_c \in \mathbb{R}^{K_h \times K_w \times K_c}$ denotes the filter kernel weight associated with the c -th channel of the output feature map \mathbf{O}_c . We introduce the spatial regularization weights \mathbf{W}_c into the convolution layer, which has the same size as \mathbf{F}_c . Considering the spatial regularization weights, the output feature map \mathbf{O}_c can be computed as

$$\mathbf{O}_c = (\mathbf{F}_c \odot \mathbf{W}_c) * \mathbf{X}_c + b, \quad (11)$$

where $*$ denotes the convolution operation, \mathbf{X}_c is the input feature map, b is the bias term in the convolution layer. For construction of \mathbf{W}_c , we first generate a binary mask \mathbf{M}_c of size $K_h \times K_w$ through Bernoulli distribution $B(0.3)$. Based on this binary mask, we construct \mathbf{W}_c as

$$\mathbf{W}_c(p, q, r) = \mathbf{M}_c(p, q), \quad (12)$$

where p, q and r denote the indexes of a 3-dimensional matrix. Clearly, only parts of the spatial regions in \mathbf{W}_c have non-zero values, which forces the filter kernels to focus on different regions.

4.2. Distance Transform Pooling Layer

The distance transform has been used in many previous papers on object detection (*e.g.*, [10, 26]). Recently, Girshick *et al.* [12] announce that distance transform is indeed a generalization of the max pooling layer, and can be expressed in a similar formula as max pooling.

Given a function $y = f(x)$ defined on a regular grid \mathcal{G} , the distance transform of $f(x)$ can be computed as

$$D_f(s) = \max_{t \in \mathcal{G}} (f(t) - d(s - t)). \quad (13)$$

Here $d(s - t)$ is a convex quadratic function with $d(s - t) = \varpi(s - t)^2 + \theta(s - t)$, where ϖ and θ are learnable parameters. The distance transform pooling layer can be used

to estimate the reliability of the input feature map. Generally speaking, the larger the learned value ϖ is, the more reliable the input feature map is. When $\varpi = \theta = 0$, this layer outputs a response with constant values, which means that the input feature map does not influence the tracking result. In implementation, the pooling region is usually bounded to save computation. We implement the distance transform pooling layer in Caffe according to [13].

5. Visual Tracking with KRR and CNN

In this section, we describe how we exploit the kernelized ridge regression and CNN for robust visual tracking.

5.1. Target Location Estimation

We perform visual tracking by combing the responses of Net-A and Net-B. Net-A captures the holistic information of the target, while Net-B focuses more on the localized region. In frame t , we crop an ROI centered in the estimated location of last frame, and obtain a feature map \mathbf{X}_t for this ROI. Then the heat map of our tracker can be computed as

$$\mathbf{f}(\mathbf{X}_t) = \gamma_1 \mathbf{f}_A(\mathbf{X}_t) + \gamma_2 \mathbf{f}_B(\mathbf{X}_t), \quad (14)$$

where γ_1 and γ_2 are the trade-off parameters, $\mathbf{f}_A(\mathbf{X}_t)$ and $\mathbf{f}_B(\mathbf{X}_t)$ denote the heat maps produced by Net-A and Net-B respectively. We obtain the position with the highest heat map score in $\mathbf{f}(\mathbf{X}_t)$, and use it as the estimated target location for the current frame.

5.2. Scale Estimation

It is not enough to only provide the location for a target object, which may experience drastic scale variation. In this paper, we further estimate the scale variation of the target after location estimation. We use S to denote the candidate scale size, use $H \times W$ to denote the input feature map size. For each $s_l \in \{\lfloor -\frac{S-1}{2} \rfloor, \dots, \lfloor \frac{S-1}{2} \rfloor\}$, we crop or pad the input feature map to size $a^{s_l}H \times a^{s_l}W$, and reshape it to $H \times W$ (we use $\mathcal{T}(\mathbf{X}_t, s_l)$ to denote the transformed feature map). These transformed feature maps are then fed into a fully connected layer to output the scale scores. We use the scale corresponding to the largest score as the estimated target state. After the scale estimation, we further exploit the bounding box regression method [11, 24] to refine the tracking result. The experimental settings are the same as [24], with the difference that we use the Conv4-3 feature of VGG-16 as input. In this work, we use Net-D to denote the scale estimation net with a fully connected layer.

5.3. Model Update

For Net-A and Net-C, we perform online update in each frame in the tracking process. In frame t , we crop a new ROI based on the estimated target state, and obtain the feature map \mathbf{X}_t . A heat map of Gaussian distribution \mathbf{G} is

generated based on which we define the L2 loss for Net-A and Net-C as follows:

$$\begin{aligned} \mathcal{L}_A &= \frac{1}{2} \|\mathbf{G} - \mathbf{f}_A(\mathbf{X}_t)\|_2^2 \\ \mathcal{L}_C^c &= \frac{1}{2} \left\| \mathbf{G} - \frac{1}{|\mathcal{C}|} \mathbf{f}_C^c(\mathbf{X}_t) \right\|_2^2, \\ \mathcal{L}_D^c &= \frac{1}{2} \left\| \mathbf{G} - \frac{1}{|\mathcal{D}|} \mathbf{f}_D^c(\mathbf{X}_t) \right\|_2^2 \end{aligned} \quad (15)$$

where $\mathbf{f}_C^c(\mathbf{X}_t)$ and $\mathbf{f}_D^c(\mathbf{X}_t)$ denote the c -th channel of output from conv2 and the distance transform pooling layer respectively, $|\cdot|$ denotes the number of output channels in one layer. With the computed loss, we use the stochastic gradient descent (SGD) method to update both networks. After model parameters α_t and β_t in Net-A are updated, we further update $\hat{\mathbf{D}}_t$, $\hat{\alpha}_t$ and $\hat{\beta}_t$ as in Eq. 10, which is similar to [22].

When scale change is detected, Net-D is updated with the following loss

$$\mathcal{L}_S = \frac{1}{2} \|y_{s_l} - \mathbf{f}_S(\mathcal{T}(\mathbf{X}_t, s_l))\|_2^2, \quad (16)$$

where \mathbf{f}_S denotes the score obtained by Net-D, $y_{s_l} = \exp(-\frac{1}{2\sigma^2} s_l^2)$ is a Gaussian function.

6. Experimental Results

In this section, we first introduce the experimental setups, and then report the experimental results of our method on three benchmark datasets, including OTB-2013 [30], OTB-2015 [31] and VOT-2016 [19]. In addition, extensive experiments are conducted to test the effectiveness of each component of the tracker. For space limitation, we only provide part of the results in the paper, more experimental results can be found in the supplementary materials. We will make the source codes available to the public.

6.1. Implementation Details

The proposed tracker is implemented with MATLAB2014a on an Intel 4.0 GHz CPU with 32G memory, and runs at around 1fps without optimization. We use the Caffe toolbox [17] to implement the networks, whose forward and backward operations are conducted on a Nvidia Titan X GPU. Considering the efficiency, we exploit the VGG-16 net for feature extraction, and only use the output of the Conv4-3 layer. We divide the target into 9 (3×3) patches in the kernelized ridge regression model. The trade-off parameters λ_1 , λ_2 , γ_1 , γ_2 are set as 0.001, 0.001, 1 and 0.45 respectively. The learning rate η is set as 0.8 in the first ten frames, and change to 0.999 in the following tracking process. All the networks in this paper are trained with the SGD method, and the learning rates for α_t and β_t in Net-A are set as 8e-9 and 1.6, while the learning rates for each layer in Net-C and Net-D are fixed which are respectively 8e-7 and 1e-10.

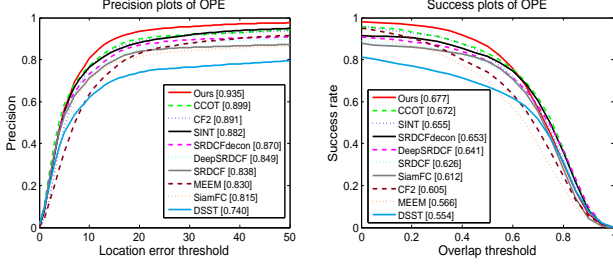


Figure 6. Precision and success plots on the OTB-2013 dataset in terms of OPE rule. In the legend, we show the distance precision rates at threshold 20 and area under curve (AUC) scores, based on which the trackers are ranked.

6.2. OTB-2013 Dataset

The OTB-2013 dataset contains 50 sequences with 11 attributes. We compare the proposed tracker with 29 trackers in [30] and 9 more state-of-the-art trackers including CCOT [9], SRDCF [7], DeepSRDCF [6], SRDCFdecon [8], MEEM [32], DSST [5], CF2 [22], SINT [27], SiamFC [3]. We exploit the one-pass evaluation for all the trackers and report both the precision and success plots for comparison. The precision plots aim to measure the percentage of frames in which the distance between the tracked result and the ground-truth is under a threshold, while the success plots aim to measure the successfully tracked frames with various thresholds. Following [30], in the precision plots, we use the distance precision rate at threshold 20 for ranking, while in the success plots, we use the area under curve (AUC) for ranking. The results for all the trackers can be visualized in Figure 6.

Among all the compared trackers, our method has a 93.5% distance precision rate at threshold 20 and a 67.7% AUC score in success plots, which outperforms the second best tracker CCOT by 3.6% and 0.5% respectively. It is noted that our method only exploits Conv4-3 layer of VGG-16 as input, while CCOT exploits three layers of features. In this figure, we do not provide the results of MDNet, which exploits large numbers of video sequences for pretraining. The reported distance precision rate at threshold 20 and the AUC score for success plots of MDNet are 94.8% and 70.8% which are comparable with the proposed tracker. In addition, on the VOT2016 dataset, we have better performance than MDNet-N, which is a variant of MDNet without features pretrained on videos.

6.3. OTB-2015 Dataset

The OTB-2015 dataset [31] is an extension of OTB-2013 with 50 extra video sequences. In this dataset, we make comparisons with 12 state-of-the-art trackers including DSST [5], CCOT [9], SRDCF [7], KCF [15], CNN-SVM [16], DeepSRDCF [6], CF2 [22], LCT [23], SRDCFdecon [8], HDT [25], Staple [2] and MEEM [32]. The

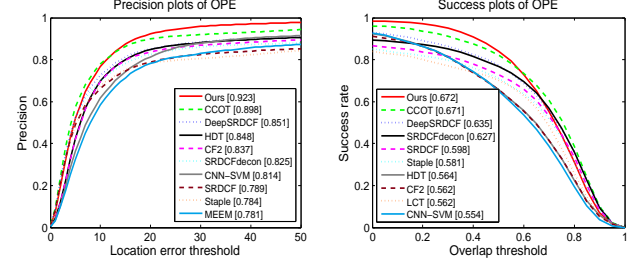


Figure 7. Precision and success plots on the OTB-2015 dataset in terms of OPE rule. In the legend, we show the distance precision rates at threshold 20 and area under curve (AUC) scores, based on which the trackers are ranked.

precision and success plots in terms of OPE rule is provided in Figure 7, in which our tracker has comparable results compared to the state-of-the-art methods. In the precision plots, our method improves the second best tracker (CCOT) by 2.5%, while in success plots, our tracker has comparable performance compared to CCOT.

The OTB-2015 dataset is divided into 11 attributes, each of which corresponds to a challenging factor in visual tracking, *e.g.*, in-plane rotation, deformation, scale variation and so on. The success plots of different trackers in terms of 8 attributes are presented in Figure 8. The results show that our method performs well when deformation and background clutter occur. The existing correlation filter methods do not perform well in such cases as they do not consider the interior structures of targets, and cannot adaptively determine which patch is more reliable. In addition, as the similarities between patches are not considered, these models are not discriminative enough with the cluttered background. In attributes such as scale variation and fast motion, we have inferior performance compared to CCOT. Part of the reason is that CCOT exploits a combination of several convolutional features, which enables it to obtain richer representations for the target.

Table 1. Performance comparison for 6 state-of-the-art algorithms on the VOT-2016 dataset.

	CCOT	Staple	EBT	DeepSRDCF	MDNet_N	Ours
EAO	0.3310	0.2952	0.2913	0.2763	0.2572	0.3242
Ar	1.75	1.78	3.38	2.00	1.68	1.75
Rr	1.97	3.27	2.18	2.85	2.85	2.38

6.4. VOT-2016 Dataset

For more thorough evaluations, we test our tracker on the VOT-2016 dataset against 5 state-of-the-art trackers, including CCOT [9], Staple [2], EBT [33], DeepSRDCF [6], MDNet_N [24]. The results of different trackers in terms of expected average overlap (EAO), accuracy rank (Ar), and robustness rank (Rr) are presented in Table 1.

Our tracker is ranked the second place in terms of EAO, which is the most important metric in the VOT dataset. In

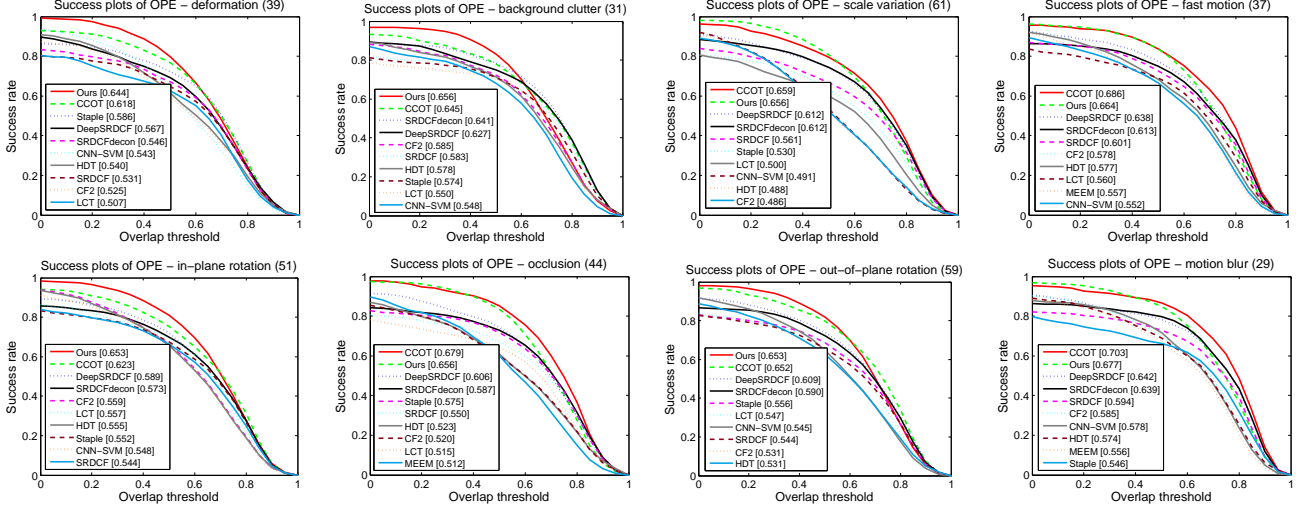


Figure 8. Performance evaluation on different attributes of the benchmark in terms of the OPE criterion.

addition, our method also has good results in terms of the robustness and accuracy ranks. It is noted that the CCOT method exploits different parameter settings for the OTB and VOT datasets, while our method uses the same experimental settings.

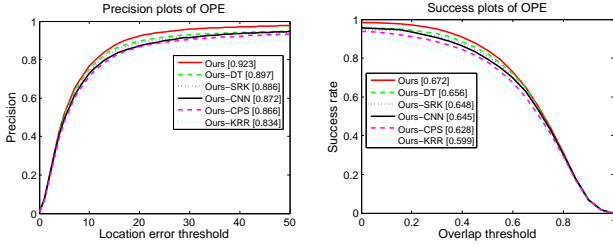


Figure 9. Performance evaluation for each component of the proposed tracker.

6.5. Discussions

In this paper, we propose two complementary regressions for visual tracking, which are respectively the kernelized ridge regression model with cross-patch similarity and convolution neural network with the spatially regularized kernels and distance transform pooling layers. We test the effectiveness for each component of our tracker in this subsection. With different experimental settings, we obtain the following 5 variants of our tracker, which are respectively named as Ours-CNN, Ours-KRR, Ours-CPS, Ours-SRK and Ours-DT. Here, Ours-CNN is our method without using the response from the CNN model for target localization, Ours-KRR is the model without using Net-A, Ours-CPS is the model that does not use the cross-patch similarity, Ours-SRK is the model that does not consider spatially regularized kernels, and Ours-DT is the model that does not exploit the distance transform pooling layer. The results of these variants and our tracker on the OTB-2015 dataset are

visualized in Figure 9.

By comparing our method with Ours-CNN and Ours-KRR, we show that both regressions contribute to the performance improvement. It is noted that the KRR method alone achieves a 87.2% distance precision rate, which outperforms many state-of-the-art trackers. The CNN alone does not perform well, as it loses much holistic information of the target. By comparing our method with Ours-CPS, we show that the cross-patch similarity kernel improves the performance by 6.6% and 4.4% in distance precision rate (at threshold 20) and AUC score. Furthermore, the effectiveness of the distance transform pooling layer and the spatially regularized filters are also validated by comparing our method with Ours-DT and Ours-SRK. An interesting observation is that the CNN model only provides a marginal improvement over our KRR model without using the spatially regularized kernels (comparing Ours-SRK and Ours-CNN). This validates our descriptions that the two models are complementary when they focus on different aspects of the target.

7. Conclusion

In this paper, we propose a novel kernelized ridge regression model with cross-patch similarity. The proposed model considers the interior structure of the target, and can adaptively determine the importance of the similarity score between two patches. We show the model can be reformulated as a neural network, and thus can be more efficiently solved. In addition, we also propose a complementary CNN model which focuses more on the localized region via a spatially regularized filter kernel. Distance transform pooling layer is further exploited to determine the reliability of the convolutional layers. Extensive experiments on several datasets validate the effectiveness of the proposed method.

References

- [1] A. Adam, E. Rivlin, and I. Shimshoni. Robust fragments-based tracking using the integral histogram. In *CVPR*, 2006. 4
- [2] L. Bertinetto, J. Valmadre, S. Golodetz, O. Miksik, and P. H. Torr. Staple: Complementary learners for real-time tracking. In *CVPR*, 2016. 7
- [3] L. Bertinetto, J. Valmadre, J. F. Henriques, A. Vedaldi, and P. H. Torr. Fully-convolutional siamese networks for object tracking. In *ECCV*, 2016. 7
- [4] D. S. Bolme, J. R. Beveridge, B. A. Draper, and Y. M. Lui. Visual object tracking using adaptive correlation filters. In *CVPR*, 2010. 2
- [5] M. Danelljan, G. Häger, F. Khan, and M. Felsberg. Accurate scale estimation for robust visual tracking. In *BMVC*, 2014. 7
- [6] M. Danelljan, G. Hager, F. Shahbaz Khan, and M. Felsberg. Convolutional features for correlation filter based visual tracking. In *ICCV Workshops*, 2015. 7
- [7] M. Danelljan, G. Hager, F. Shahbaz Khan, and M. Felsberg. Learning spatially regularized correlation filters for visual tracking. In *ICCV*, 2015. 2, 7
- [8] M. Danelljan, G. Hager, F. Shahbaz Khan, and M. Felsberg. Adaptive decontamination of the training set: A unified formulation for discriminative visual tracking. In *CVPR*, 2016. 2, 7
- [9] M. Danelljan, A. Robinson, F. S. Khan, and M. Felsberg. Beyond correlation filters: Learning continuous convolution operators for visual tracking. In *ECCV*, 2016. 2, 7
- [10] P. F. Felzenszwalb, R. B. Girshick, D. McAllester, and D. Ramanan. Object detection with discriminatively trained part-based models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(9):1627–1645, 2010. 5
- [11] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *CVPR*, 2014. 6
- [12] R. Girshick, F. Iandola, T. Darrell, and J. Malik. Deformable part models are convolutional neural networks. In *CVPR*, 2015. 5
- [13] I. J. Goodfellow, D. Warde-Farley, M. Mirza, A. C. Courville, and Y. Bengio. Maxout networks. *Journal of Machine Learning Research*, 28:1319–1327, 2013. 6
- [14] J. F. Henriques, R. Caseiro, P. Martins, and J. Batista. Exploiting the circulant structure of tracking-by-detection with kernels. In *ECCV*, 2012. 2
- [15] J. F. Henriques, R. Caseiro, P. Martins, and J. Batista. High-speed tracking with kernelized correlation filters. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 37(3):583–596, 2015. 1, 2, 7
- [16] S. Hong, T. You, S. Kwak, and B. Han. Online tracking by learning discriminative saliency map with convolutional neural network. In *ICML*, 2015. 7
- [17] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. In *ICMM*, 2014. 6
- [18] H.-U. Kim, D.-Y. Lee, J.-Y. Sim, and C.-S. Kim. Sowp: Spatially ordered and weighted patch descriptor for visual tracking. In *ICCV*, 2015. 4
- [19] M. Kristan, J. Matas, A. Leonardis, M. Felsberg, L. Cehovin, G. Fernandez, T. Vojir, G. Hager, G. Nebehay, and R. Pflugfelder. Online object tracking: A benchmark. In *ECCV Workshops*, 2016. 6
- [20] Y. LeCun, J. S. Denker, S. A. Solla, R. E. Howard, and L. D. Jackel. Optimal brain damage. In *NIPS*, 1989. 2
- [21] H. Li, Y. Li, and F. Porikli. Robust online visual tracking with a single convolutional neural network. In *ACCV*, 2014. 2
- [22] C. Ma, J.-B. Huang, X. Yang, and M.-H. Yang. Hierarchical convolutional features for visual tracking. In *ICCV*, 2015. 2, 6, 7
- [23] C. Ma, X. Yang, C. Zhang, and M.-H. Yang. Long-term correlation tracking. In *CVPR*, 2015. 7
- [24] H. Nam and B. Han. Learning multi-domain convolutional neural networks for visual tracking. In *CVPR*, 2016. 1, 2, 6, 7
- [25] Y. Qi, S. Zhang, L. Qin, H. Yao, Q. Huang, J. Lim, and M.-H. Yang. Hedged deep tracking. In *CVPR*, 2016. 2, 7
- [26] P.-A. Savalle, S. Tsogkas, G. Papandreou, and I. Kokkinos. Deformable part models with cnn features. In *ECCV Workshop*, 2014. 5
- [27] R. Tao, E. Gavves, and A. W. Smeulders. Siamese instance search for tracking. In *CVPR*, 2016. 2, 7
- [28] L. Wang, W. Ouyang, X. Wang, and H. Lu. Visual tracking with fully convolutional networks. In *CVPR*, 2015. 1, 2
- [29] L. Wang, W. Ouyang, X. Wang, and H. Lu. Stct: Sequentially training convolutional networks for visual tracking. In *CVPR*, 2016. 2, 5
- [30] Y. Wu, J. Lim, and M.-H. Yang. Online object tracking: A benchmark. In *CVPR*, 2013. 6, 7
- [31] Y. Wu, J. Lim, and M.-H. Yang. Object tracking benchmark. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 37(9):1834–1848, 2015. 6, 7
- [32] J. Zhang, S. Ma, and S. Sclaroff. Meem: robust tracking via multiple experts using entropy minimization. In *ECCV*, 2014. 7
- [33] G. Zhu, F. Porikli, and H. Li. Beyond local search: Tracking objects everywhere with instance-specific proposals. In *CVPR*, 2016. 7