DeepTrack: Learning Discriminative Feature Representations by Convolutional Neural Networks for Visual Tracking

Hanxi Li[12]
hanxi.li@nicta.com.au
Yi Li[1]
http://users.cecs.anu.edu.au/~yili/
Fatih Porikli[1]
http://www.porikli.com/

[1] NICTA and ANU, Canberra, Australia
[2] Jiangxi Normal University, Jiangxi, China

Abstract

Defining hand-crafted feature representations needs expert knowledge, requires time-consuming manual adjustments, and besides, it is arguably one of the limiting factors of object tracking.

In this paper, we propose a novel solution to automatically relearn the most useful feature representations during the tracking process in order to accurately adapt appearance changes, pose and scale variations while preventing from drift and tracking failures. We employ a candidate pool of multiple Convolutional Neural Networks (CNNs) as a data-driven model of different instances of the target object. Individually, each CNN maintains a specific set of kernels that favourably discriminate object patches from their surrounding background using all available low-level cues. These kernels are updated in an online manner at each frame after being trained with just one instance at the initialization of the corresponding CNN. Given a frame, the most promising CNNs in the pool are selected to evaluate the hypothesises for the target object. The hypothesis with the highest score is assigned as the current detection window and the selected models are retrained using a warm-start back-propagation which optimizes a structural loss function.

In addition to the model-free tracker, we introduce a class-specific version of the proposed method that is tailored for tracking of a particular object class such as human faces. Our experiments on a large selection of videos from the recent benchmarks demonstrate that our method outperforms the existing state-of-the-art algorithms and rarely loses the track of the target object.

1 Introduction

Tracking objects across video frames has long been a fundamental challenge in computer vision. In a typical setting, the object location is initialized in the first frame, and the object windows in subsequent frames are reported as tracking results.

In general, there are three core tasks to be carried out by an object tracking method: 1) mathematical modelling of the object of interest, 2) searching for the best location of this

model in a new frame, and 3) updating the model according to the newly acquired object location. The performance of all these three tasks highly depends on how the underlying object region is modelled into a mathematical form, *i.e.* feature representation. While there is a plethora of existing methods exploring various combinations of preferred approaches for these tasks, finding automatically an optimal set of discriminative feature representations to obtain the most robust and accurate tracking performance is still an unsolved problem.

In many early methods, features are manually defined and combined [1, 5, 9, 16]. Even though these methods report satisfactory results on individual datasets, hand-crafted selection of feature representations would limit the performance of visual tracking. For instance, color histogram feature, which would be discriminative when the lighting condition is favourable, might become ineffective when the object moves under shadow. It is preferable to learn a representation, *e.g.* a filter applied to a low-level cue, from the current data rather than predefining a generic filter for all possible scenarios.

Recently, deep neural networks have gained attention as an alternative solution for various computer vision tasks. Here, *deep* indicates a multi-layer neural network architecture that can efficiently capture sophisticated hierarchies describing the raw data. Making the network deeper will raise the learning capacity significantly [3]. In particular, the Convolutional Neural Network (CNN) has shown superior performance on standard object recognition benchmarks [13], [14], [4]. With a deep structure, CNN can effectively learn complicated mappings while utilizing minimal domain knowledge.

Using offline trained CNN for tracking has been proposed in the past [8], [18]. In [18], the CNN learns filters from a large dataset of natural images. However, this method uses an off-line trained single CNN and embeds it into an off-the-shelf tracking framework, treating CNN as a predefined feature extractor.

Immediate adoption of an online CNN idea for visual tracking, however, is not straightforward. First of all, CNN requires a large number of training samples, which unfortunately may not be available in visual tracking as there exist only a few number of positive instances of the target object particularly at the beginning of the video. Secondly, CNN tends to easily overfit to the most recent observation, *e.g.* most recent instance kidnapping the model, which may result in drift problem. Besides, CNN training is computationally very intensive for online visual tracking.

Here, we propose a novel solution to automatically relearn the most useful feature representations by taking advantage of the deep neural networks while addressing the above issues. Our motivation is that by employing a candidate pool of multiple CNNs we can capture and adapt to the underlying changes in object appearance and pose without overfitting to the data. While it is plausible to use a single CNN as the object model, training a pool of separate appearance models for individual instances provides much more flexible and computationally advantageous representation. We consider object tracking as a consecutive object-background labelling process. In this spirit, we use CNN to learn discriminative representations that discriminate object and background regions.

Typically, tracking-by-detection approaches rely on predefined heuristics to sample from the estimated object location and construct a set of positive or negative samples. Often these samples have binary labels, which causes model deterioration in case of a slight inaccuracy during tracking [9]. To overcome this, our method employs a structural loss function, which allows us to use a large number of training samples that have different significance levels, to achieve robustness of the model while preventing from the model deterioration. Individually, each CNN maintains a specific set of kernels that favourably discriminate object patches from their surrounding background using all available low-level cues. These kernels are

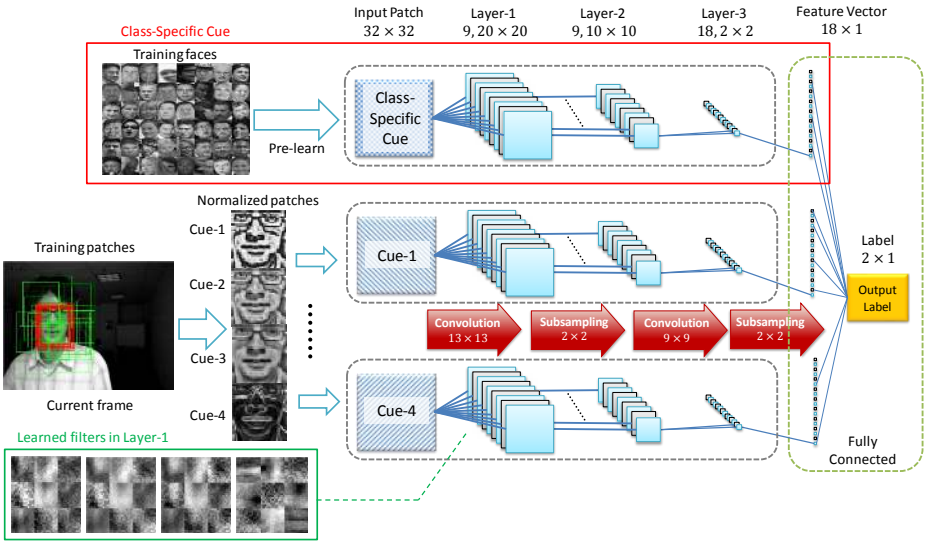

Figure 1: Overall architecture with (red box) and without (rest) the class-specific version.

updated in an online manner at each frame after being trained by just one instance at the initialization of the corresponding CNN.

For a given frame, we apply a subset of CNNs from the pool to the patches surrounding the previous object location. For each patch, we determine the best matching CNNs using the k-NN and assign the best score as the score of the patch. And the patch with the highest score is assigned as the current detection. The selected CNN corresponding to the current detection is retrained with a warm-start back-propagation scheme using all the labelled patches from the current frame. This enables us to significantly reduce the computational complexity and not distort the other CNNs in the pool. Since we use different cues, we apply an iterative training procedure each cues independently then jointly train their fully-connected layers. This makes the training efficient. Empirically we observed that this two-stage iterative procedure is more accurate than jointly training for all cues.

In addition to be able to track any image region, we introduce a class-specific version of the proposed method that is tailored for tracking of a particular object class such as face. In this version, the corresponding kernels of the specific object category cue is pretrained and fixed during the tracking.

Our experiments on 16 videos from recent benchmarks demonstrate that our method consistently outperforms 9 state-of-the-art algorithms and rarely loses the track of the objects.

2 DeepTrack: CNN Architecture

Effective object tracking requires multiple cues, which may include color, image gradients and different pixel-wise filter responses. These cues are weakly correlated yet contain complementary information. Therefore, it is preferable to consider them simultaneously in a scalable neural network. Below, we present the CNN structure for a single cue. Then, we couple multiple CNNs and construct our CNN architecture. We then introduce the structural

learning concept, which takes both patch similarities and their spatial relations into account.

2.1 CNN for a single cue

Our CNN for a single cue consists of two convolutional layers[1], corresponding sigmoid functions as activation neurons and average pooling operators. The dashed gray block in Fig. 1 shows the structure of our single cue network, which can be expressed as $(32\times 32)-(10\times 10\times 9)-(1\times 1\times 18)-(2)$ in conventional neural network notation.

The input is 32×32 image patches, which draws a balance between the representation power and computational load. The first convolutional layer contains 9 kernels each of size 13×13 (an empirical trade-off between overfitting due to a very large number of kernels and discrimination power), followed by a pooling operation that reduces the obtained feature map (filter response) to a lower dimension. The second convolutional layer contains 162 kernels with size 9×9. This leads to a 18 dimensional feature vector in the second layer, after the pooling operation in this layer.

The fully connected layer is a logistic regression operation. It takes the feature vector $\theta\in\mathcal{R}^{18}$ and generates the score vector $\mathbf{s}=[s_1,s_2]^{\mathrm{T}}\in\mathcal{R}^2$, with s_1 and s_2 corresponding to the positive score and negative score respectively. In order to increase the score margin between the positive and negative samples, we calculate the final CNN score of patch n as $S_n=s_1\cdot\exp(s_1-s_2)$.

2.2 CNN for multiple cues

In this work, we use 4 image cues generated from the given gray-scale image, *i.e.*, three locally normalized images with different parameter configurations and a gradient image. The locally normalized image is usually employed in CNNs as the normalization not only alleviates the saturation problem but also makes the CNN robust to illumination changes. On the other hand, the gradient images capture the shape information of the object and is a good complementary feature to the normalized gray-scale images. In specific, two parameters r_μ and r_σ determine a local contrast normalization process and we use three configurations, *i.e.*, $\{r_\mu=3,r_\sigma=1\}$, $\{r_\mu=3,r_\sigma=3\}$ and $\{r_\mu=5,r_\sigma=5\}$, respectively. We let CNN to select the most informative cues in a data driven fashion.

By concatenating the final responses of these 4 cues, we build a fully connected layer to the binary output vector (the green dashed block in Fig. 1).

2.3 Structural loss function

Let $\mathbf{x}_n$ and $\mathbf{l}_n\in\{[0,1]^{\mathrm{T}},[1,0]^{\mathrm{T}}\}$ denote the cue of the input patch and its ground truth label (background or foreground) respectively, and $f(\mathbf{x}_n;\Omega)$ be the predicted score of $\mathbf{x}_n$ with network weights Ω, in the conventional setting of CNN learning, the objective function of N samples in the batch is

$$\mathcal{L}=\frac{1}{N}\sum_{n=1}^{N}\|f(\mathbf{x}_n;\Omega)-\mathbf{l}_n\|_2 \tag{1}$$

when the CNN is trained in the batch-mode. Equation 1 is a commonly used loss function and performs well in binary classification problems. However, for object localization tasks, usually higher performance can be obtained by 'structurizing' the binary classifier. The

[1]Note that here the "convolutional layer" refers to the convolution process, rather than the layers defined in Fig.1

advantage of employing the structural loss is the larger number of available training samples, which is crucial to the CNN training. In the ordinary binary-classification setting, one can only use the training samples with high importance to avoid class ambiguity. In contrast, the structural CNN is learned based upon all the sampled patches.

In our CNN framework, we modify the original CNN's output to $f(\phi\langle\Gamma,\mathbf{y}_n\rangle;\Omega)\in\mathbb{R}$, where Γ is the current frame, $\mathbf{y}_n\in\mathbb{R}^o$ is the motion hypothesis vector of the target object, which determines the object's location in Γ and o is the freedom degree[2] of the transformation. The operation $\phi\langle\Gamma,\mathbf{y}_n\rangle$ suffices to crop the features from Γ using the motion $\mathbf{y}_n$.

The associated structural loss is defined as

$$\mathcal{L}=\frac{1}{N}\sum_{n=1}^{N}\left[\Delta(\mathbf{y}_n,\mathbf{y}^*)\cdot\|f(\phi\langle\Gamma,\mathbf{y}_n\rangle;\Omega)-\mathbf{l}_n\|_2\right], \tag{2}$$

where $\mathbf{y}^*$ is the (estimated) motion state of the target object in the current frame. To define $\Delta(\mathbf{y}_n,\mathbf{y}^*)$ we first calculate the overlapping score $\Theta(\mathbf{y}_n,\mathbf{y}^*)$ [7] as

$$\Theta(\mathbf{y}_n,\mathbf{y}^*)=\frac{\text{area}(r(\mathbf{y}_n)\bigcap r(\mathbf{y}^*))}{\text{area}(r(\mathbf{y}_n)\bigcup r(\mathbf{y}^*))} \tag{3}$$

where $r(\mathbf{y})$ is the region defined by $\mathbf{y}$, $\bigcap$ and $\bigcup$ denotes the intersection and union operations respectively. Finally we have

$$\Delta(\mathbf{y}_n,\mathbf{y}^*)=\left|\frac{2}{1+\exp(-(\Theta(\mathbf{y}_n,\mathbf{y}^*)-0.5))}-1\right|\in[0,1]. \tag{4}$$

And the sample label $\mathbf{l}_n$ is set to $[1,0]$ if $\Theta(\mathbf{y}_n,\mathbf{y}^*)>0.5$ while $[0,1]$ otherwise. From 4 we can see that $\Delta(\mathbf{y}_n,\mathbf{y}^*)$ actually measures the importance of the training patch n. For instance, patches that are very close to object center and reasonably far from it may play more significant roles in training the CNN, while the patches in between are less important.

3 Tracking with DeepTrack

It is possible to train jointly the above CNN architecture (shown in Fig. 1) as a whole using the patches described in Sec. 2.3. However, to make the tracking robust and computationally efficient, we train each cue and the fully-connected layer iteratively. We also maintain a pool of CNN models for modelling changes over time.

3.1 Online learning: a coordinate-descent approach

We used Stochastic Gradient Decent (SGD) for learning the parameters Ω. Using multiple image cues leads to a CNN with higher complexity, which implies a low training speed and a high possibility of overfitting. Notice that each image cue may be weakly independent, thus we train the network iteratively.

In particular, we define the model parameters as $\Omega=\{\mathbf{w}^1_{cov},\cdots,\mathbf{w}^K_{cov},\mathbf{w}^1_{fc},\cdots,\mathbf{w}^K_{fc}\}$, where $\mathbf{w}^k_{cov}$ denotes the filter parameters in cue-k while $\mathbf{w}^k_{fc}$ corresponds to the fully-connected layer. After we complete the training on $\mathbf{w}^k_{cov}$, we evaluate the filter responses from all the

[2] In this paper, we set $o=3$, i.e., the object bounding box changes in its xy location and the scale.

cues in the fully-connected layer and then jointly update $\{\mathbf{w}_{fc}^1, \cdots, \mathbf{w}_{fc}^K\}$ with a small learning rate (see Algorithm 1). This can be regarded as a coordinate-descent variation of SGD. In practice, we found this strategy effectively curbs the overfitting while increases the training speed. An empirical comparison between the traditional SGD and the proposed coordinate-descent variation is also reported in 4.3.

Algorithm 1 Iterative SGD-based training

Inputs: Frame image Γ; CNN model (K cues) $f(\phi\langle\Gamma,\cdot\rangle;\Omega=\{\mathbf{w}_{cov}^1,\cdots,\mathbf{w}_{cov}^K,\mathbf{w}_{fc}^1,\cdots,\mathbf{w}_{fc}^K\})$.
Given/Estimated $\mathbf{y}^*$; Sample labels $\mathbf{l}_1,\mathbf{l}_2,\cdots,\mathbf{l}_N$; Sample motion hypotheses $\mathbf{y}_1,\mathbf{y}_2,\cdots,\mathbf{y}_N$.
Learning rates $\hat{r}=\frac{r}{K}$; minimal loss ε; training step budget $M \gg K$.
for $m \leftarrow 1,\ M$ **do**
  Calculate loss $\mathcal{L}=\frac{1}{N}\sum_{n=1}^{N}[\Delta(\mathbf{y}_n,\mathbf{y}^*)\cdot\|f(\phi\langle\Gamma,\mathbf{y}_n\rangle;\Omega)-\mathbf{l}_n\|_2]$
  If $\mathcal{L}\leq\varepsilon$, **break**;
  $k=\mathrm{mod}(m,K)+1$;
  Update $\mathbf{w}_{cov}^k$ using SGD with learning rate r;
  Jointly update $\{\mathbf{w}_{fc}^1,\cdots,\mathbf{w}_{fc}^K\}$ using Gradient Decent with step length $\hat{r}$;
end for
Outputs: New CNN model.

3.2 Temporal adaptation with a CNN pool

Instead of learning one complicated and powerful CNN model for all the appearance observations in the past, we chose a relatively small number of filters in the CNN within a framework equipped with a temporal adaptation mechanism. Assuming we buffer the models of the CNN during each update, we build a pool of CNN models $\mathcal{F}=\{f_1,f_2,\ldots,f_Q\}$ associated with a prototype set $\mathcal{X}=\{\hat{\mathbf{x}}_1,\hat{\mathbf{x}}_2,\ldots,\hat{\mathbf{x}}_Q\}$, where prototype $\hat{\mathbf{x}}_q$ is a typical positive sample cue[3] when the CNN model f_q is learned and Q is the size limit of the pool[4].

Given a new frame, we randomly generate 1500 motion hypothesises $\mathcal{Y}=\{\mathbf{y}_1,\mathbf{y}_2,\ldots,\mathbf{y}_N\}$ of the object around its previous motion state. For each hypothesis $\mathbf{y}_n$, we select k nearest CNN models in the pool by measuring the similarities between its cue $\mathbf{x}_n$ and all the prototypes in $\mathcal{X}$[5]. Then we perform k nearest CNNs for each $\mathbf{y}_n$ and the hypothesis with the highest maximal score is chosen as the detected result, *i.e.*,

$$\mathbf{y}^*=\arg\max_{\mathbf{y}_n\in\mathcal{Y}}\left(\max_{f_j\in\mathcal{N}_n} f_j(\phi\langle\Gamma,\mathbf{y}_n\rangle;\Omega)\right), \tag{5}$$

where $\mathcal{N}_n$ is a subset of $\mathcal{Y}$ and contains the k nearest CNNs of $\mathbf{y}_n$.

After detection, we update the selected model $f^*=\arg\max_{f_j\in\mathcal{N}_{\mathbf{y}^*}} f_j(\phi\langle\Gamma,\mathbf{y}_n\rangle;\Omega)$ and increase its rank according to its test error on the current frame. The system will then decide whether to update f^* or add a new CNN into the pool, based on the training error. When the pool size reaches the budget Q, the CNN candidates with the lowest rank is replaced. The details of the temporal adaptation mechanism are illustrated in Figure 2. This temporal adaptation strategy has two advantages: 1) we can accommodate as many as possible appearance

[3] In this work we use image gradient as the prototype cue and set $k=5$, $Q=50$.
[4] The idea of using template pool has already shown its superiority for long term visual tracking, see [15]
[5] In this work the similarity is measured in the space obtained via a PCA mapping

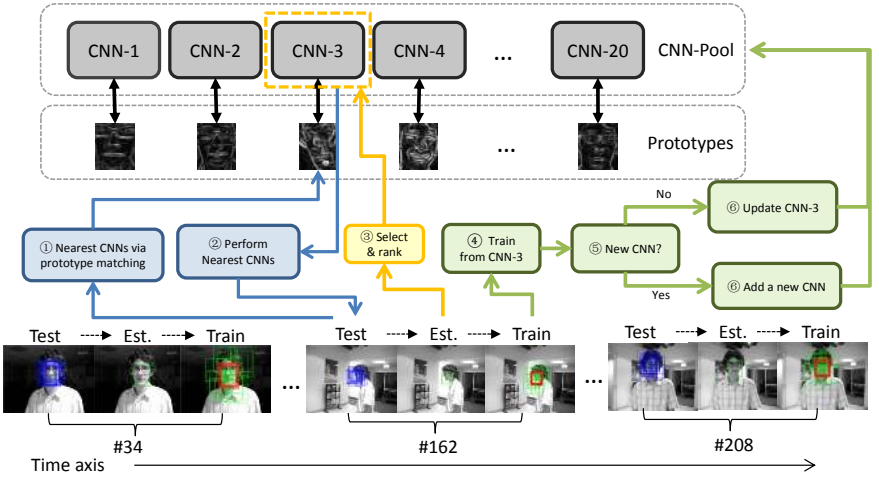

Figure 2: Illustration of the temporal adaptation mechanism. In the middle row, the test, estimation and training operations are marked in blue, yellow and green blocks respectively. The numbers ① to ⑥ indicates the operation order. The bottom row shows the three-stages operations on a frame: test, estimation and training. In the frame images, the green bounding-boxes are the negative samples while the red ones denote the positiv samples.

variations without learning an ensemble of CNNs or a very complicated CNN, which is time consuming, 2) the associated ranking mechanism can explicitly refine the model pool and discard CNNs with low ranking.

3.3 Class-specific tracking

In certain applications, the target object is from a known class of objects such as human faces. Our method can use this prior information to leverage the performance of tracking by training a class-specific detector. In the tracking stage, given the particular instance information, one needs to combine the class-level detector and the instance-level tracker in a certain way, which usually leads to higher model complexity.

The proposed multiple-cue CNN and the coordinate-descent optimization method naturally allow class-specific tracking. We first learn a class-level detector by the CNN. The CNN detector is then used as a special cue in the DeepTrack. In the tracking stage, all the parameters in the class-cue is fixed except the ones in the fully-connected layer. The information from the class-level and the instance-level are naturally merged via update the fully-connected layer of the joint model.

4 Experiments

4.1 Data and comparison methods

We evaluate our method on 16 benchmark video sequences that cover most challenging tracking scenarios such as scale changes, illumination changes, occlusions, cluttered backgrounds and fast motion. The first frames of these sequences are listed in Figure 3. We

compare our method with 9 other state-of-the-art trackers including TLD [12], CXT [6], ASLA [11], Struck [9], SCM [20], and the classical tracking algorithms, *e.g.*, Frag [1], CPF [16], IVT [17] and MIL [2].

Figure 3: The first frames of the selected video sequences. Top row, from left to right: David, Jumping, David2, Trellis, Fish, Car4, CarDark, Singer2. Bottom row, from left to right: Skating1, Shaking, FaceOcc2, FaceOcc1, Singer1, Deer, Dudek, Sylvester.

The tracking results are evaluated via the following two measurements: 1) Tracking Precision (TP), the percentage of the frames whose estimated location is within the given distance-threshold τ_d to the ground truth, and 2) Tracking Success Rate (TSR), the percentage of the frames in which the overlapping score defined in Equation 3 between the estimated location and the ground truth is larger than a given overlapping-threshold τ_o. For a fair comparison, we run our algorithm for 5 times and then report the average TP/TSR scores. The results of other visual trackers are obtained from [19]. We run our algorithm in Matlab with an unoptimized code mixed with CUDA-PTX kernels for the CNN implementation. In average, it takes 1.3 seconds for tracking the target object in one frame on a PC with a 3.2G Hz CPU, 16G memory and a middle-level consumer graphics card (NVIDIA GTX770).

4.2 Results

Firstly, we evaluate using fixed thresholds $\tau_d = 15$, $\tau_o = 0.5$, which is a common setting in tracking evaluations (shown in Table 1). In particular, the score of TP and TSR are shown together in each table block. The average scores are also reported for all the trackers. We can see that, the proposed method achieves better average results compared with other trackers, with the performance gap 10% for TP and 12% for TSR. It is possible to see our DeepTrack is very robust to dramatic appearance changes, e.g. due to motion blurs (Jumping and Deer) or illumination variations (Fish, Trellis and Singer2).

In addition to evaluations with fixed thresholds, we also analyse tracking performance with varying ones. In specific, for TP, we evaluate the trackers with the thresholds $\tau_d = 1, 2, \cdots, 50$ while for TSR, we use the thresholds $\tau_o = 0$ to 1 at the step of 0.05. According to the scores under different criteria, we generate the precision curves and the success-rate curves for each tracking method, which is shown in Figure 4.

From the score plots we can see that, DeepTrack ranks the first (red curves) for both TP and TSR evaluations. Our algorithm is very robust as for $\tau_o < 0.72$ and $\tau_d > 7$ as it outperform all other trackers. DeepTrack rarely misses the target completely. Having mentioned that when the overlap thresholds are tight (*e.g.* $\tau_o > 0.8$ or $\tau_d < 5$), our tracker has similar response to rest of the trackers we tested.

	David	Jumping	David2	Trellis	Fish	Car4	CarDark	Singer2	
TLD	**1.00**/**0.97**	0.98/0.85	**1.00**/0.95	0.48/0.47	0.98/0.96	0.86/0.79	0.61/0.53	0.04/0.10	
CXT	**1.00**/0.83	0.86/0.29	**1.00**/**1.00**	0.88/0.81	**1.00**/**1.00**	0.30/0.30	0.71/0.69	0.05/0.04	
ASLA	**1.00**/0.96	0.29/0.17	1.00/0.95	0.86/0.86	**1.00**/**1.00**	**1.00**/**1.00**	**1.00**/**1.00**	0.03/0.04	
Struck	0.32/0.24	**1.00**/0.80	**1.00**/**1.00**	0.83/0.78	**1.00**/**1.00**	0.97/0.40	**1.00**/**1.00**	0.03/0.04	
SCM	**1.00**/0.91	0.14/0.12	**1.00**/0.91	0.86/0.85	0.84/0.86	0.97/0.97	**1.00**/**1.00**	0.11/0.16	
Frag	0.13/0.12	0.96/0.85	0.31/0.30	0.35/0.36	0.52/0.55	0.18/0.21	0.39/0.25	0.16/0.20	
CPF	0.11/0.03	0.14/0.11	**1.00**/0.46	0.22/0.17	0.09/0.10	0.03/0.02	0.11/0.02	0.08/0.14	
IVT	0.95/0.79	0.18/0.10	**1.00**/0.93	0.32/0.31	**1.00**/**1.00**	**1.00**/**1.00**	0.77/0.70	0.04/0.04	
MIL	0.42/0.23	0.76/0.48	0.69/0.32	0.17/0.24	0.34/0.39	0.35/0.28	0.27/0.18	0.18/0.48	
CNN	0.98/0.95	**1.00**/**0.99**	**1.00**/0.97	**0.99**/**0.97**	0.99/**1.00**	0.97/0.79	**1.00**/**1.00**	**0.80**/**0.91**	
	Skating1	Shaking	FaceOcc2	FaceOcc1	Singer1	Deer	Dudek	Sylvester	Overall
TLD	0.26/0.23	0.33/0.40	0.69/0.83	0.04/0.83	0.98/0.99	0.73/0.73	0.50/0.84	**0.94**/**0.93**	0.65/0.71
CXT	0.20/0.12	0.05/0.11	0.98/0.95	0.19/0.77	0.80/0.32	0.94/0.92	0.73/0.92	0.76/0.75	0.65/0.61
ASLA	**0.76**/**0.69**	0.25/0.38	0.67/0.81	0.13/0.31	**1.00**/**1.00**	0.03/0.03	0.61/0.90	0.78/0.75	0.65/0.68
Struck	0.29/0.37	0.08/0.17	**0.99**/**1.00**	0.24/**1.00**	0.56/0.30	**1.00**/**1.00**	0.78/**0.98**	0.93/**0.93**	0.69/0.69
SCM	0.72/0.42	**0.70**/**0.90**	0.74/0.87	0.65/**1.00**	**1.00**/**1.00**	0.03/0.03	0.80/**0.98**	0.89/0.89	0.72/0.74
Frag	0.14/0.12	0.07/0.07	0.54/0.75	**0.83**/**1.00**	0.23/0.22	0.18/0.21	0.48/0.59	0.68/0.68	0.38/0.40
CPF	0.20/0.19	0.14/0.12	0.29/0.35	0.18/0.52	0.94/0.32	0.04/0.04	0.45/0.69	0.79/0.71	0.30/0.25
IVT	0.08/0.07	0.01/0.01	0.91/0.91	0.34/0.98	0.81/0.48	0.03/0.03	**0.84**/0.97	0.68/0.68	0.56/0.56
MIL	0.12/0.10	0.18/0.23	0.55/0.94	0.15/0.76	0.33/0.28	0.08/0.13	0.57/0.86	0.54/0.55	0.36/0.40
CNN	0.60/0.46	0.39/0.42	0.89/0.79	0.50/0.97	0.76/0.92	0.98/**1.00**	0.43/0.77	0.91/0.90	**0.82**/**0.86**

Table 1: The tracking scores of the proposed method and other visual trackers. The reported results are shown in the order of "TP/TSR".

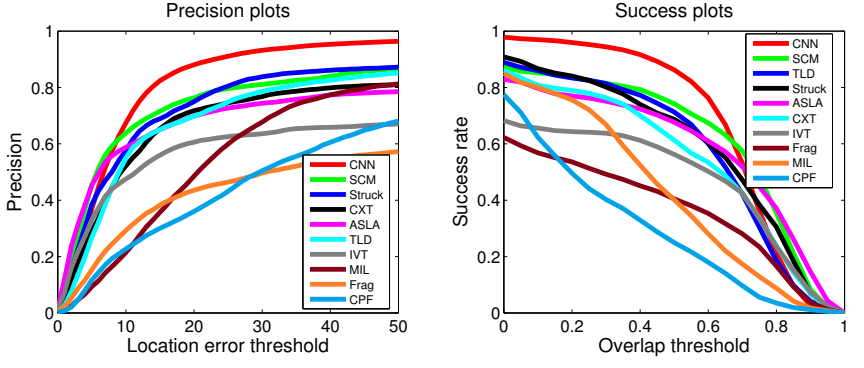

Figure 4: The Precision Plot (left) and the Success Plot (right) of the comparing visual tracking methods. Note that, the color of one curve is determined by the rank of the corresponding tracker, not the tracker's name.

4.3 Class specific tracking

On the 8 video sequences that contain human faces, we analysed the class-specific version of the DeepTrack. We first learn a general face detector using CNN, based on the data set proposed in [10]. Then, the CNN-based face detector is embedded into the DeepTrack to track the faces on the sequences. The performance scores are shown in Table 2, we can see that, with prior knowledge carried by the detector, the class-specific CNN tracker significantly outperforms the original one on most of the sequences. Based on the face videos, we also verify the effectiveness of the proposed iterative-SGD method by reporting the results of the CNN-trackers, which is trained in the ordinary SGD manner.

From Table 2 and Figure 5 we can see that, equipped with the class-specific information, the CNN tracker achieves better performance under both TP and TSR criteria. The class-specific cue significantly improves the performance of DeepTrack on some sequences on which the ordinary DeepTrack performs poorly (Shaking and Dudek). In contrast, without

	David	Jumping	David2	Trellis	Shaking	FaceOcc2	FaceOcc1	Dudek	Overall
CNN	0.98/0.95	**1.00**/**0.99**	**1.00**/**0.97**	0.99/0.97	0.39/0.42	**0.89**/0.79	0.50/**0.97**	0.43/0.77	0.77/0.84
Class-CNN	**1.00**/**0.98**	0.99/0.78	1.00/0.73	**1.00**/**1.00**	**0.60**/**0.65**	0.84/**0.92**	**0.56**/0.94	**0.73**/**0.95**	**0.84**/**0.87**
Normal-SGD	0.52/0.50	0.76/0.68	0.97/0.72	0.65/0.62	0.04/0.05	0.80/0.81	0.20/0.95	0.30/0.68	0.53/0.63

Table 2: The comparison between the DeepTrack and the class-specific DeepTrack. The reported results are shown in the order of "TP/TSR". For each video sequence, the best result is shown in bold font.

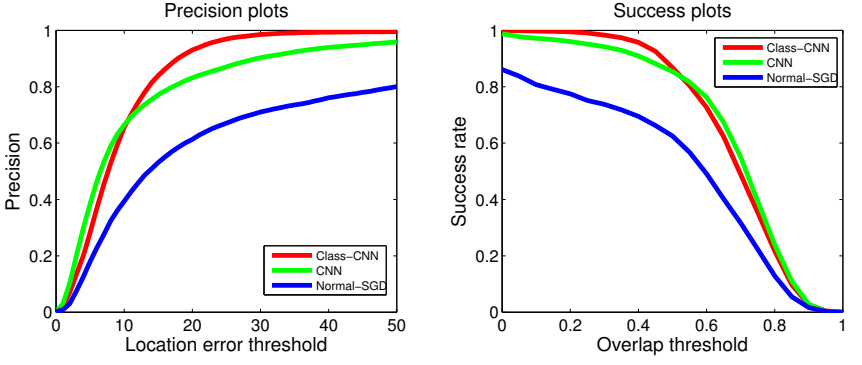

Figure 5: The Precision Plot (left) and the Success Plot (right) of the comparing visual tracking methods.

the proposed iterative SGD method, the average performance of the CNN tracker drops by around 20%.

5 Conclusion

We introduced DeepTrack, a CNN based online object tracker. We employed a CNN architecture and a structural loss function that handles multiple input cues and class-specific tracking. We also proposed an iterative procedure, which speeds up the training process significantly. Together with the CNN pool, our experiments demonstrate that DeepTrack performs very well on 16 sequences.

References

[1] Amit Adam, Ehud Rivlin, and Ilan Shimshoni. Robust fragments-based tracking using the integral histogram. In *CVPR 2006*, volume 1.

[2] Boris Babenko, Ming-Hsuan Yang, and Serge Belongie. Visual tracking with online multiple instance learning. *Transactions on Pattern Analysis and Machine Intelligence*, August 2011.

[3] Y. Bengio, A. Courville, and P. Vincent. Representation learning: A review and new perspectives. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 35(8): 1798–1828, 2013.

[4] Dan Claudiu Ciresan, Ueli Meier, and Jürgen Schmidhuber. Multi-column deep neural networks for image classification. In *CVPR 2012*.

[5] Robert T. Collins, Yanxi Liu, and Marius Leordeanu. Online selection of discriminative tracking features. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27 (10):1631–1643, 2005. ISSN 0162-8828.

[6] Thang Ba Dinh, Nam Vo, and Gérard Medioni. Context tracker: Exploring supporters and distracters in unconstrained environments. In *CVPR 2011*, pages 1177–1184. IEEE.

[7] Mark Everingham, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. The pascal visual object classes (voc) challenge. *Intl J. of Comp. Vis.*, 88 (2):303–338, 2010.

[8] Jialue Fan, Wei Xu, Ying Wu, and Yihong Gong. Human tracking using convolutional neural networks. *Trans. Neur. Netw.*, 21(10):1610–1623, October 2010. ISSN 1045-9227.

[9] Sam Hare, Amir Saffari, and Philip HS Torr. Struck: Structured output tracking with kernels. In *ICCV 2011*, pages 263–270. IEEE.

[10] Vidit Jain and Erik Learned-Miller. Fddb: A benchmark for face detection in unconstrained settings. Technical Report UM-CS-2010-009, University of Massachusetts, Amherst, 2010.

[11] Xu Jia, Huchuan Lu, and Ming-Hsuan Yang. Visual tracking via adaptive structural local sparse appearance model. In *CVPR 2012*, pages 1822–1829. IEEE.

[12] Zdenek Kalal, Jiri Matas, and Krystian Mikolajczyk. Pn learning: Bootstrapping binary classifiers by structural constraints. In *CVPR 2010*, pages 49–56. IEEE.

[13] Koray Kavukcuoglu, Pierre Sermanet, Y-Lan Boureau, Karol Gregor, Michaël Mathieu, and Yann LeCun. Learning convolutional feature hierachies for visual recognition. In *NIPS 2010*.

[14] Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS 2012*.

[15] Karel Lebeda, Simon Hadfield, Jiri Matas, and Richard Bowden. Long-term tracking through failure cases. In *Computer Vision Workshops (ICCVW), 2013 IEEE International Conference on*, pages 153–160. IEEE, 2013.

[16] Patrick Pérez, Carine Hue, Jaco Vermaak, and Michel Gangnet. Color-based probabilistic tracking. In *ECCV 2002*.

[17] David A. Ross, Jongwoo Lim, Ruei-Sung Lin, and Ming-Hsuan Yang. Incremental learning for robust visual tracking. *Intl. J. Comp. Vis*, 77(1-3):125–141, May 2008. ISSN 0920-5691.

[18] Naiyan Wang and Dit-Yan Yeung. Learning a deep compact image representation for visual tracking. In *NIPS 2013*.

[19] Yi Wu, Jongwoo Lim, and Ming-Hsuan Yang. Online object tracking: A benchmark. *CVPR 2013*.

[20] Wei Zhong, Huchuan Lu, and Ming-Hsuan Yang. Robust object tracking via sparsity-based collaborative model. In *CVPR 2012*, pages 1838–1845. IEEE.