Action Recognition with Trajectory-Pooled Deep-Convolutional Descriptors

Limin Wang[1,2] Yu Qiao[2] Xiaoou Tang[1,2]

[1]Department of Information Engineering, The Chinese University of Hong Kong

[2]Shenzhen key lab of Comp. Vis. & Pat. Rec., Shenzhen Institutes of Advanced Technology, CAS, China

07wanglimin@gmail.com, yu.qiao@siat.ac.cn, xtang@ie.cuhk.edu.hk

Abstract

Visual features are of vital importance for human action understanding in videos. This paper presents a new video representation, called trajectory-pooled deep-convolutional descriptor (TDD)*, which shares the merits of both hand-crafted features [31] and deep-learned features [24]. Specifically, we utilize deep architectures to learn discriminative convolutional feature maps, and conduct trajectory-constrained pooling to aggregate these convolutional features into effective descriptors. To enhance the robustness of TDDs, we design two normalization methods to transform convolutional feature maps, namely spatiotemporal normalization and channel normalization. The advantages of our features come from (i) TDDs are automatically learned and contain high discriminative capacity compared with those hand-crafted features; (ii) TDDs take account of the intrinsic characteristics of temporal dimension and introduce the strategies of trajectory-constrained sampling and pooling for aggregating deep-learned features. We conduct experiments on two challenging datasets: HMDB51 and UCF101. Experimental results show that TDDs outperform previous hand-crafted features [31] and deep-learned features [24]. Our method also achieves superior performance to the state of the art on these datasets* [1].

1. Introduction

Human action recognition [1, 24, 31, 35, 36] in videos attracts increasing research interests in computer vision community due to its potential applications in video surveillance, human computer interaction, and video content analysis. However, action recognition remains as a difficult problem when focusing on realistic datasets collected from movies [17], web videos [15, 26], and TV shows [20]. There are large intra-class variations in the same action class, which may be caused by background clutter, viewpoint change, and various motion speeds and styles. Meanwhile, the high dimension and low resolution of video further increases the difficulty to design efficient and robust recognition method. *Visual representations* from action videos are crucial for dealing with these issues and designing effective recognition systems. Currently, there are mainly two types of video features available for action recognition, as illustrated in Figure 1.

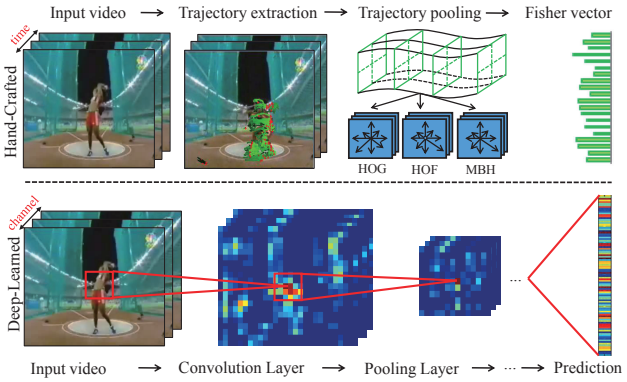

Figure 1. There are mainly two types of features in action recognition: *hand-crafted* features and *deep-learned* features. For hand-crafted features, improved trajectories [31] combined with Fisher vector are most successful. For deep-learned features, Convolutional Networks (ConvNets) [18] are popular deep architectures, which contain a sequence of convolutional and pooling layers. They aims to automatically learn features with a deep discriminatively trained neural network.

The first type of representations are the *hand-crafted* local features, and typical local features include Space Time Interest Points [16], Cuboids [7], Dense Trajectories [30], and Improved Trajectories [31]. Calculation of these local features can be usually decomposed into two phrases: *detector*, which aims to discover the salient and informative regions for action understanding, and *descriptor*, whose goal is to describe the visual patterns of extracted regions. Among these local features, improved trajectories with rich descriptors of HOG, HOF, MBH have shown to be successful on a number of challenging datasets (e.g. HMDB51 [15], UCF101 [26]) and contests (e.g. THUMOS

[1]The TDD code and learned two-stream ConvNet models are available at https://wanglimin.github.io/tdd/index.html

[11]). Improved trajectories include several important ingredients in their extraction process. Firstly, these extracted trajectories are mainly located at regions with high motion salience, which contain rich and discriminative information for action recognition. Secondly, these local descriptors of the corresponding regions in several successive frames, are aligned and pooled along the trajectories. This trajectory-constrained sampling strategy also takes account of the temporal continuity of human action, and is effective to deal with the variations of motion speed. However, these hand-crafted descriptors are not optimized for visual representation and may lack discriminative capacity for action recognition.

The second type of representations are the *deep-learned* features, and typical methods include Convolutional RBMs [29], 3D ConvNets [9], Deep ConvNets [12], and Two-Stream ConvNets [24]. These deep learning methods aim to automatically learn the semantic representation from raw video by using a deep neural network discriminatively trained from a large number of labeled data. Two-Stream ConvNets [24] are probably the most successful architecture at present, and they match the state-of-the-art performance of improved trajectories [31, 32] on UCF101 and HMDB51. They are composed of two neural networks, namely spatial nets and temporal nets. Spatial nets mainly capture the discriminative appearance features for action understanding, while temporal nets aim to learn the effective motion features. However, unlike image classification tasks [14], these deep learning based methods fail to outperform previous hand-crafted features. One problem of deep learning methods is that they require a large number of labeled videos for training, while most available datasets are relatively small. Meanwhile, most of current deep learning based action recognition methods largely ignore the intrinsic difference between temporal domain and spatial domain, and just treat temporal dimension as feature channels when adapting the architectures of ConvNets to model videos.

Motivated by the above analysis, this paper proposes a new kind of video feature, called *trajectory-pooled deep-convolutional descriptor* (TDD). The design of TDD aims to combine the benefits of both hand-crafted and deep-learned features. To achieve this goal, our approach integrates the key factors from two successful video representations, namely improved trajectories [31] and two-stream ConvNets [24]. We utilize deep architecture to learn multi-scale convolutional feature maps, and introduce the strategies of trajectory-constrained sampling and pooling to encode deep features into effective descriptors.

Specifically, we first train two-stream ConvNets on a relatively large dataset, while more labeled action videos will make ConvNet training more stable and robust. Then, we treat the learned two-stream ConvNets as generic feature extractors, and use them to obtain multi-scale convolutional feature maps for each video. Meanwhile, we detect a set of point trajectories with the method of improved trajectories. Based on convolutional feature maps and improved trajectories, we pool the local ConvNet responses over the spatiotemporal tubes centered at the trajectories, where the resulting descriptor is called TDD. Finally, we choose Fisher vector representation to aggregate these local TDDs over the whole video into a global super vector, and use linear SVM as the classifier to perform action recognition. We conduct experiments on two public action datasets: the HMDB51 dataset [15] and the UCF101 dataset [26]. We show that our TDDs obtain the state-of-the-art performance for action recognition on these challenging datasets. Meanwhile, our results demonstrate that our TDDs are complementary to those hand-crafted features (HOG, HOF, and MBH) and the fusion of them is able to further boost the recognition performance.

2. Related Works

Hand-crafted features. Local features [7, 16, 33, 39] have become popular and effective representations in action recognition, as these local features do not require algorithms to detect human body and are robust to background clutter, illumination changes, and video noise. Space Time Interest Points [16] proposed Harris3D detector to extract informative regions, while Cuboid [7] detector relied on temporal Gabor filters. Willems *et al.* [39] proposed a Hessian detector, which is a spatio-temporal extension of Hessian saliency measure used for blob detection in images. Meanwhile several local descriptors have been proposed to represent the 3D volumes extracted around these interest points, such as Histogram of Gradient (HOG), Histogram of Optical Flow (HOF) [17], 3D Histogram of Gradient (HOG3D) [13], and Extended SURF (ESURF) [39]. Recent works made use of point trajectories [30, 31] to extract and align 3D volumes, and resorted to more rich low level descriptors for constructing effective video representations, including HOG, HOF, and Motion Boundary Histogram (MBH).

One limitation of these local features is that they lack semantics and discriminative capacity. To overcome this issue, several mid-level and high-level video representations have been proposed such as Action Bank [22], Dynamic-Poselets [37], Motionlets [35], Motion Atoms and Phrases [34], and Actons [42]. They usually resorted to some heuristic mining methods to select discriminative visual elements as feature units. Instead, this paper takes a different view of this problem and replace these local hand-crafted descriptors with deep-learned representations. Our deep representations deliver high level semantic information, and are learned automatically from training data without using these heuristic rules.

Deep-learned features. Deep learning techniques have

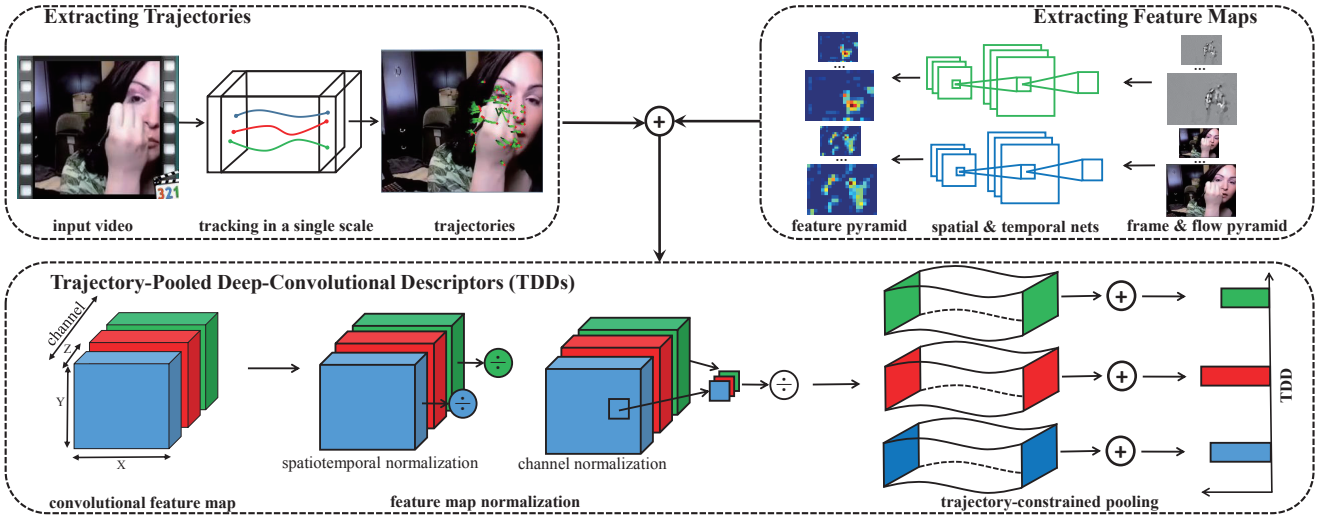

Figure 2. **Pipeline of TDD.** The whole process of extracting TDD is composed of three steps: (i) extracting trajectories, (ii) extracting multi-scale convolutional feature maps, and (iii) calculating TDD. We effectively exploit two available state-of-the-art video representations, namely improved trajectories and two-stream ConvNets. Grounded on them, we conduct trajectory-constrained sampling and pooling over convolutional feature maps to obtain trajectory-pooled deep convolutional descriptors.

achieved great success in image based tasks [14, 25, 28, 41] and there have been a number of attempts to develop deep architectures for video action recognition [9, 12, 24, 29]. Taylor *et al.* [29] used Gated Restricted Boltzmann Machines (GRBMs) to learn the motion features in an unsupervised manner and then resorted to convolutional learning to fine tune the parameters. Ji *et al.* [9] extended 2D ConvNet to video domain for action recognition on relatively small datasets, and recently Karpathy *et al.* [12] tested ConvNets with deep structures on a large dataset, called Sports-1M. However, these deep models achieved lower performance compared with shallow hand-crafted representation [31], which might be ascribed to two facts: firstly, available action datasets are relatively small for deep learning; secondly, learning complex motion patterns is more challenging. Simonyan *et al.* [24] designed two-stream ConvNets containing spatial and temporal net by exploiting large ImageNet dataset for pre-training and explicitly calculating optical flow for capturing motion information, and finally it matched the state-of-the-art performance.

However, these deep models lacked considerations of temporal characteristics of video data and relied on large training datasets. We incorporate video temporal characteristics into deep architectures by using strategy of trajectory-constrained sampling and pooling, and propose a new descriptor. Meanwhile, our descriptors can be easily adapted to the datasets of smaller size.

3. Improved Trajectories Revisited

As shown in Figure 2, our proposed representation (TDD) is based on low level trajectory extraction and we choose improved trajectories [31]. In this section, we briefly review the extraction process of improved trajectories. It is worth noting that our TDD is independent of the method of extracting trajectories, and we use improved trajectories due to its good performance.

Improved trajectories are extended from dense trajectories [30]. To compute dense trajectories, the first step is to densely sample a set of points on 8 spatial scales on a grid with step size of 5 pixels. Points in homogeneous areas are eliminated by setting a threshold for the smaller eigenvalue of their autocorrelation matrices. Then these sampled points are tracked by media filtering of dense flow field.

$$P_{t+1} = (x_{t+1}, y_{t+1}) = (x_t, y_t) + (\mathcal{M} * \omega_t)|_{(\overline{x}_t, \overline{y}_t)}, \quad (1)$$

where $\mathcal{M}$ is the median filter kernel, $*$ is convolutional operation, $\omega_t = (u_t, v_t)$ is the dense optical flow field of the t^{th} frame, and $(\overline{x}_t, \overline{y}_t)$ is the rounded position of (x_t, y_t). To avoid the drifting problem of tracking, the maximum length of trajectory is set as 15-frame. Finally, those static trajectories are removed as they lack motion information, and other trajectories with suddenly large displacement are also ignored, since they are obviously incorrect due to inaccurate optical flow.

Improved trajectories boost the recognition performance of dense trajectories by taking camera motion into account. It assumes that the background motion of two consecutive frames can be characterized by a homography matrix. To estimate the homography matrix, the first step is to find the correspondence between two consecutive frames. They resort to SURF [2] feature matching and optical flow based matching, as these two kinds of matching scheme are complementary to each other. Then, they use the

RANSAC [8] algorithm to estimate homography matrix. Based on the homography, they rectify the frame image to remove the camera motion and re-calculate the optical flow, called *warped flow*. Warped flow brings advantages to the descriptors calculated from optical flows, in particular for HOF, and trajectories corresponding to camera motion can be removed too.

We adopt improved trajectories for the task of TDD extraction, but make a modification. Unlike dense trajectories or improved trajectories, we only track points on its original spatial scale, and extract multi-scale TDDs around the extracted trajectories (see Section 4). We observe that tracking on a single scale is fast for implementation. In summary, given a video V, we obtain a set of trajectories

$$\mathbb{T}(V) = \{T_1, T_2, \cdots, T_K\}, \tag{2}$$

where K is the number of trajectories, and T_k denotes the k^{th} trajectory in the original spatial scale:

$$T_k = \{(x_1^k, y_1^k, z_1^k), (x_2^k, y_2^k, z_2^k), \cdots, (x_P^k, y_P^k, z_P^k)\}, \tag{3}$$

where (x_p^k, y_p^k, z_p^k) is the pixel position of the p^{th} point in trajectory T_k, and P is the length of trajectory ($P = 15$). These trajectories will be used for trajectory-constrained sampling and pooling in the process of TDD extraction, as described in the next section.

4. Deep Convolutional Descriptors

In this section, we describe a new video representation, called *trajectory-pooled deep-convolutional descriptor* (TDD), which shares the benefits of both hand-crafted and deep-learned features. We first introduce the architectures of convolutional networks (ConvNets) we used. Then, we show how to adapt the ConvNets trained on large datasets to extract multi-scale convolutional feature maps. Finally, based on improved trajectories and convolutional feature maps, we describe the details of how to calculate TDDs.

4.1. Convolutional networks

Our TDD starts with designing deep ConvNets for extracting convolutional feature maps. In principle, any kind of ConvNet architecture can be adopted for TDD extraction. In our implementation, we choose the two-stream ConvNets[24] due to their good performance on the datasets of UCF101 and HMDB51.

The two-stream ConvNets contain two separate ConvNets, namely spatial nets and temporal nets. Spatial nets are designed for capturing static appearance cues, which are trained on single frame images ($224 \times 224 \times 3$), while temporal nets aim to describe the dynamic motion information, whose input are volumes of stacking optical flow fields ($224 \times 224 \times 2F$, F is the number of stacking flows). Meanwhile, decoupling the spatial and temporal nets also allows to exploit the available images by pre-training spatial nets on the ImageNet challenge dataset [6], and explicitly handle motion information with optical flow algorithms for temporal nets

The details about ConvNets are shown in Table 1. This ConvNet architecture is original from the Clarifai networks [41] and adapted to the task of action recognition with less filters in conv4 layer and lower-dimensional full7 layer. But we make a small modification. We use the same network architecture for both spatial and temporal net in addition to the input data layer, while the original two-stream ConvNets [24] ignore the second local response normalized (LRN) layer in the temporal net due to memory consumption problem. The implementation and training details can be found in Section 5.

4.2. Convolutional feature maps

Once the training of two-stream ConvNets is complete, we treat them as generic feature extractors to obtain the convolutional feature maps of videos. In general, for each video, we obtain these feature maps of spatial and temporal net in a frame-by-frame and volume-by-volume manner, respectively. In order to make the feature maps with equal temporal duration with input video, we pad the optical flow fields at the beginning with $F - 1$ copies of the optical flow field from the first frame, where F is the number of stacking optical flow.

For each frame or volume, we take it as the input for spatial or temporal nets. We make two modifications about the spatial and temporal nets. The first one is that we remove the layers after the target layer for feature extraction. For example, to extract feature maps of conv4, we will remove the layers from conv5 to full8. Therefore, the output of spatial and temporal net will be the convolutional feature maps, which will be used for extracting TDD in the next subsection.

The second modification is that before each convolutional or pooling layer, with kernel size k, we conduct zero padding of the layer's input with size $\lfloor k/2 \rfloor$. This padding allows the input and output maps of these layers to have the same spatial extent. With this padding, it will be straightforward to map the positions of trajectory points in video to the coordinates of convolutional feature maps. A trajectory point with video coordinates (x_p, y_p, z_p) in Equation (3) will be centered on $(r \times x_p, r \times y_p, z_p)$ in convolutional map, where r is map size ratio with respective to input size, as listed in Table 1.

ConvNets are bottom-up architectures with a sequence of alternating convolutional and pooling layers. Different layers of ConvNets have various receptive fields as shown in Table 1, ranging from 7×7 to 171×171. As described in paper [41], these different layers capture patterns from simple visual elements such as edges, to complex visual

Layer	conv1	pool1	conv2	pool2	conv3	conv4	conv5	pool5	full6	full7	full8
size	7×7	3×3	5×5	3×3	3×3	3×3	3×3	3×3	-	-	-
stride	2	2	2	2	1	1	1	2	-	-	-
channel	96	96	256	256	512	512	512	512	4096	2048	101
map size ratio	1/2	1/4	1/8	1/16	1/16	1/16	1/16	1/32	-	-	-
receptive field	7×7	11×11	27×27	43×43	75×75	107×107	139×139	171×171	-	-	-

Table 1. **ConvNet Architectures.** We use similar architectures to two-stream ConvNets [24], which are adapted to the task of action recognition from the Clarifai networks [41], with less filters in conv4 layer (512 vs. 1024) and lower-dimensional full7 layer (2048 vs. 4096). For layers of conv1 and conv2, local response normalized (LRN) is applied with parameters settings: $n = 5, \alpha = 5 \times 10^{-4}, \beta = 0.75$. The layers of full6 and full7 are regularised by using dropout and the full8 layer acts as a soft-max classifier. The activation function for all weight layers is the rectification linear unit (RELU). The size ratios of feature maps with respect to input data range from $1/2$ to $1/32$, and the feature receptive fields vary from 7×7 to 171×171, for different convolutional and pooling layers (conv1 to pool5).

concepts such as parts and objects. The higher layers have larger receptive fields and obtain more invariant and discriminative features. Intuitively, these different layers describe the visual content at different levels, each of which is complementary to each other for the task of recognition. We will exploit this complimentary property of different layers during the extraction of TDD. Given a video V, we obtain a set of convolutional feature maps:

$$\mathbb{C}(V) = \{C_1^s, C_2^s, \cdots, C_M^s, C_1^t, C_2^t, \cdots, C_M^t\}, \quad (4)$$

where $C_m^s \in \mathbb{R}^{H_m \times W_m \times L \times N_m}$ is the m^{th} feature map of spatial net, H_m is its height, W_m is its width, L is the video duration, and N_m is the number of channels. $C_m^t \in \mathbb{R}^{H_m \times W_m \times L \times N_m}$ is the m^{th} feature map of temporal net, M is the number of layers for extracting TDD.

4.3. Trajectory-pooled descriptors

We will describe the method for extracting trajectory-pooled deep-convolutional descriptors (TDDs) from a set of improved trajectories $\mathbb{T}(V)$ and convolutional feature maps $\mathbb{C}(V)$ for a given video V. In essence, TDD is a kind of local trajectory-aligned descriptor computed in a 3D volume around the trajectory. TDDs from the spatial and temporal nets capture the appearance and motion information of this 3D volume, respectively. The size of the volume is $N \times N$ pixels and P frames, where N is the receptive field size and P is the trajectory length. The extraction of TDD is composed of two steps: *feature map normalization* and *trajectory pooling*.

Normalization proves to be an effective strategy in designing features partially because it can reduce the influence of illumination. It has been widely exploited in local descriptors such as SIFT [19], HOG [5], and HOF [17], and in deep learning such as local response normalization [14]. We apply the normalization strategy to the convolutional feature maps of two-stream ConvNets to suppress the activation burstiness of some neurons. We design two kinds of normalization methods:

- *Spatiotemporal Normalization.* For spatiotemporal normalization, we normalize the feature map for each channel independently across the video spatiotemporal extent. Given a feature map $C \in \mathbb{R}^{H \times W \times L \times N}$ of Equation (4), we normalize the convolutional feature value as follows:

$$\widetilde{C}_{st}(x, y, z, n) = C(x, y, z, n) / \mathrm{maxV}_{st}^n, \quad (5)$$

where maxV_{st}^n is the maximum value of n^{th} feature maps over the whole video spatiotemporal extent, which means $\mathrm{maxV}_{st}^n = \max_{x,y,z} C(x, y, z, n)$. The spatiotemporal normalization method ensures that each convolutional feature channel ranges in the same interval, and thus contributes equally to final TDD recognition performance.

- *Channel Normalization.* For channel normalization, we normalize the feature map for each pixel independently across the feature channels. We conduct channel normalization for feature map $C \in \mathbb{R}^{H \times W \times L \times N}$ as follows:

$$\widetilde{C}_{ch}(x, y, z, n) = C(x, y, z, n) / \mathrm{maxV}_{ch}^{x,y,z}, \quad (6)$$

where $\mathrm{maxV}_{ch}^{x,y,z}$ is the maximum value of different feature channels at pixel position (x, y, z), that is $\mathrm{maxV}_{ch}^{x,y,z} = \max_n C(x, y, z, n)$. This channel normalization is able to make sure that the feature value of each pixel range in the same interval, and let each pixel make the equal contribution in the final representation.

After the step of feature normalization, we will extract TDDs based on trajectories and normalized convolutional feature maps by using trajectory pooling. Specifically, given a trajectory T_k and a normalized feature map $\widetilde{C}_m^a$, which is the m^{th}-layer feature map after either spatiotemporal normalization or channel normalization from spatial net or temporal net ($a \in \{s, t\}$), we conduct sum-pooling of the normalized feature maps over the 3D volume centered at the trajectory as follows:

$$D(T_k, \widetilde{C}_m^a) = \sum_{p=1}^{P} \widetilde{C}_m^a(\overline{(r_m \times x_p^k)}, \overline{(r_m \times y_p^k)}, z_p^k), \quad (7)$$

where (x_p^k, y_p^k, z_p^k) is the p^{th} point position of video coordinates in trajectory T_k, r_m is the m^{th}-layer map size ratio with respective to input size as listed in Table 1, $\overline{(\cdot)}$ is the rounding operation. $D(T_k, \widetilde{C}_m^a)$ is called *trajectory-pooled deep convolutional descriptor*, and is a new kind of feature combing the merits of both improved dense trajectories and two-stream ConvNets.

Multi-scale TDD extension. The above description on TDD extraction is about the single scale, we will present the multi-scale extension of TDD. For improved trajectory, it samples points and tracks them on multi-scale videos, while fixes the spatial extent of HOG, HOF, and MBH descriptors as 32×32. The original method needs to conduct point tracking and descriptor calculation in multi-scale settings. In our implementation, we try a more efficient multi-scale strategy. Specifically, we calculate optical flow and track point in a single scale. Then we construct multi-scale pyramid representations of video frames and optical flow fields. These pyramid representations are fed into the two stream ConvNets and transformed into multi-scale convolutional feature maps as shown in Figure 2. Based on multi-scale convolutional maps and single-scale improved trajectories, we are able to compute multi-scale TDDs efficiently, by applying trajectory pooling to multi-scale convolutional feature maps as described above. The only modification to different scales is to replace feature map size ratio r_m in Equation (7) with $r_m \times s$, where s is the scale of current feature map. In practice, compared with improved trajectories, we use less scales with $s = 1/2, 1/\sqrt{2}, 1, \sqrt{2}, 2$.

5. Experiments

In this section, we first present the details of datasets and their evaluation scheme. Then, we describe the details of our method. Finally, we give the experimental results and compare TDD with the state of the art.

5.1. Datasets

In order to verify the effectiveness of TDDs, we conduct experiments on two public large datasets, namely HMDB51 [15] and UCF101 [26]. The HMDB51 dataset is a large collection of realistic videos from various sources, including movies and web videos. The dataset is composed of $6,766$ video clips from 51 action categories, with each category containing at least 100 clips. Our experiments follow the original evaluation scheme using three different training/testing splits. In each split, each action class has 70 clips for training and 30 clips for testing. The average accuracy over these three splits is used to measure the final performance.

The UCF101 dataset contains 101 action classes and there are at least 100 video clips for each class. The whole dataset contains $13,320$ video clips, which are divided into 25 groups for each action category. We follow the evaluation scheme of the THUMOS13 challenge [11] and adopt the three training/testing splits for evaluation. As UCF101 is larger than HMDB51, we use the UCF101 dataset to train two-stream ConvNets initially, and transfer this learned model for TDD extraction on the HMDB51 dataset.

5.2. Implementation details

Two-stream ConvNets training. Training deep ConvNets is more challenging for action recognition as action is more complex than object and the available dataset is extremely small compared with the ImageNet dataset [6]. We choose the training dataset of UCF101 split1 for learning two-stream ConvNets as it is probably the largest public available dataset. We use the Caffe toolbox [10] for ConvNet implementation. The network weights are learnt using the mini-batch (set to 256) stochastic gradient descent with momentum (set to 0.9). For spatial net, we first resize the frame to make the smaller side as 256, and then randomly crop a 224×224 region from the frame. It then undergoes random horizontal flipping. We pre-train the network with the public available model [4]. Finally, we fine tune the model parameters on the UCF101 dataset, where the learning rate is set as 10^{-2}, decreased to 10^{-3} after 14K iterations, and training stopped at $20K$ iterations.

For temporal net, its input is 3D volume of stacking optical flows fields. We choose the TVL1 optical flow algorithm [40] and use the OpenCV implementation, due to its balance between accuracy and efficiency. For fast computation, we discretize the values of optical flow fields into integers and set their range as 0-255 just like images. Specifically, we choose to stack 10 frames of optical flow fields to keep a balance between performance and efficiency. We train temporal net on UCF101 from scratch. As the dataset is relatively small, we use high dropout ratio to improve the generalization capacity of trained model. We set dropout 0.9 for full6 layer and dropout 0.8 for full7 layer. The training procedure of temporal net is similar to spatial net and a $224 \times 224 \times 20$ sub-volume is randomly cropped and flipped from training video. The learning rate is initially set as 10^{-2} and decreases to 10^{-3} after 50K iterations. It is then reduced to 10^{-4} after 70K iterations and training is stopped at 90K iterations.

Results of two-stream ConvNets. To evaluate the trained model, as in [24], we select 25 frames for each video clip and obtain 10 crops for each frame. The final recognition result is the average across these crops and frames. We obtain 71.2% recognition accuracy with spatial net and 80.1% with temporal net. The performance of our implemented two-stream ConvNets is 84.7%, which is similar to that of two-stream ConvNets [24] (85.6%).

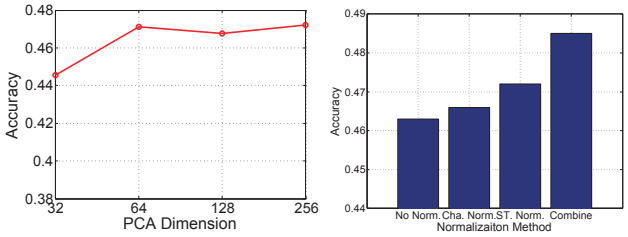

Figure 3. Exploration of different settings in TDD on the HMDB51 dataset. Left: Performance trend with varying PCA reduced dimension. Right: Comparison of different normalization methods. "Combine" means the fusion of spatiotemporal normalization and channel normalization.

However, obtaining ConvNets with high performance is not the final goal of this paper, and we aim to verify the effectiveness of TDDs.

Feature encoding. We choose Fisher vector [23] to encode the TDDs of a video clip into high dimensional representation as its effectiveness for action recognition has been verified in previous works [38, 27], and then use a linear SVM as the classifer ($C = 100$). In order to train GMMs, we first de-correlate TDD with PCA and reduce its dimension to D. Then, we train a GMM with K ($K = 256$) mixtures, and finally the video is represented with a $2KD$-dimensional vector.

5.3. Exploration experiments

Dimension reduction. To specify the PCA dimension of TDD for GMM training and Fisher vector encoding, we first explore different dimensions reduced by PCA on the HMDB51 dataset, with conv4 descriptors from spatial net. In this exploration experiment, we use the spatiotemporal normalization method for TDD and the results are shown in the left of Figure 3. We vary the dimension from 32 to 256 and the results show that dimension 64 achieves the high performance, and higher dimension may cause performance degradation. Thus, we fix the dimension as 64 for TDDs in the remainder of this section.

Normalization method. Another important component in TDD design is the normalization method and we have presented two normalization methods: spatiotemporal normalization (ST. Norm.) and channel normalization (Cha. Norm.) in Section 4.3 . We conduct experiments to investigate the effectiveness of normalization methods by using conv4 descriptors from spatial net on the HMDB51 dataset, and the results are shown in the right of Figure 3. We see that normalization is important for improving performance and spatiotemporal normalization is the best choice. We also explore the complementary property of these two normalization methods by fusing the Fisher vectors of them, and observe that it can further improve the performance. Therefore, in the remainder of this section, we will use the combined representation obtained from these two normalization methods for TDDs.

Algorithm	HMDB51	UCF101
HOG [31, 32]	40.2%	72.4%
HOF [31, 32]	48.9%	76.0%
MBH [31, 32]	52.1%	80.8%
HOF+MBH [31, 32]	54.7%	82.2%
iDT [31, 32]	**57.2%**	**84.7%**
Spatial net [24]	40.5%	73.0%
Temporal net [24]	54.6%	83.7%
Two-stream ConvNets [24]	**59.4%**	**88.0%**
Spatial conv4	48.5%	81.9%
Spatial conv5	47.2%	80.9%
Spatial conv4 and conv5	**50.0%**	**82.8%**
Temporal conv3	54.5%	81.7%
Temporal conv4	51.2%	80.1%
Temporal conv3 and conv4	**54.9%**	**82.2%**
TDD	63.2%	90.3%
TDD and iDT	**65.9%**	**91.5%**

Table 3. Performance of TDD on the HMDB51 dataset and UCF101 dataset. We compare our proposed TDD with iDT features [31] and two-stream ConvNets [24]. We also explore the complementary properties TDD features and iDT features. The combination of them can further boost the performance.

Different layers. Finally we investigate the performance of TDDs from different layers of spatial and temporal nets on the HMDB51 dataset, and the results are summarized in Table 2. For layers of conv5, conv4, and conv3, we use the outputs of RELU activations, and for layers of conv2 and conv1, we choose the outputs of max pooling layers after convolution operations. We see that descriptors of layers conv4 and conv5 obtain highest recognition performance for spatial net, while the ones of layers conv3 and conv4 are top performers for temporal net. Therefore, in the following evaluation of TDD, we choose the descriptors from conv4 and conv5 layers for spatial nets, and conv3 and conv4 layers for temporal nets.

5.4. Evaluation of TDDs

In this section, we evaluate the performance of our proposed TDDs on the HMDB51 and UCF101 dataset, and the experimental results are summarized in Table 3. We first compare the performance of TDDs with that of improved trajectories. The convolutional descriptors of spatial net are much better than HOG descriptors, which indicates that deep-learned features contains more discriminative capacity than hand-crafted features. For convolutional descriptors of temporal net, they are better than or comparable to the descriptors of HOF and MBH, but the improvement is not so evident as spatial convolutional descriptors. The reason may be that HOF and MBH calculation is based on warped optical flow instead of original optical flow, which has been proved to be pretty effective for HOF descriptor [31]. We

	Spatial ConvNets					Temporal ConvNets				
Convolutional layer	conv1	conv2	conv3	conv4	conv5	conv1	conv2	conv3	conv4	conv5
Recognition accuracy	24.1%	33.9%	41.9%	**48.5%**	**47.2%**	39.2%	50.7%	**54.5%**	**51.2%**	46.1%

Table 2. The performance of different layers of spatial nets and temporal nets on the HMDB51 dataset.

(a) RGB (b) Flow-x (c) Flow-y (d) S-conv4 (e) S-conv5 (f) T-conv3 (g) T-conv4

Figure 4. Examples of video frames, optical flow fields, and their corresponding feature maps of spatial nets and temporal nets.

consider using warped flow for TDDs extraction in the future.

We also compare the performance of TDDs with the two-stream ConvNets. Although our trained two-stream ConvNets obtain slightly lower performance than theirs, we see that our spatial TDDs outperform spatial nets by a large margin and temporal TDD is comparable to their temporal net. These results indicate the fact that trajectory-constrained sampling and pooling is an effective strategy for improving recognition performance, in particular for spatial TDDs. We also notice that the combined TDDs from spatial and temporal nets outperform two-stream ConvNets by around 4% and 2% on the two datasets, respectively. We also show some examples of video frames, optical flow fields, and their corresponding feature maps in Figure 4. From these examples, we see that the convolutional feature maps are relatively sparse and exhibit high correlation with the action areas.

Finally, we explore a practical way to improve the recognition performance of action recognition system by combining TDDs with iDTs, using early fusion of Fisher vector representation. The recognition results are shown in Table 3, and the fusion of them can further boost the performance. This further improvement indicates our TDDs are complementary to those low-level local features.

Computational costs. Compared with iDT, we only track points on a single scale and extract original flow instead of warped flow. The ConvNets are implemented by Cuda and computing is very efficient.

5.5. Comparison to the state of the art

Table 4 compares our recognition results with several recently published methods on the dataset of HMDB51 and UCF101. The performance of TDDs outperforms previous methods on both datasets. On the HMDB51 dataset, our best result outperforms other methods by 4.8%, and on the UCF101 dataset, our best result outperforms by 3.5%. This superior performance of TDDs indicates the effectiveness of introducing trajectory-constrained sampling and pooling into deep-learned features.

HMDB51		UCF101	
STIP+BoVW [15]	23.0%	STIP+BoVW [26]	43.9%
Motionlets [35]	42.1%	Deep Net [12]	63.3%
DT+BoVW [30]	46.6%	DT+VLAD [3]	79.9%
DT+MVSV [3]	55.9%	DT+MVSV [3]	83.5%
iDT+FV [31]	57.2%	iDT+FV [32]	85.9%
iDT+HSV [21]	61.1%	iDT+HSV [21]	87.9%
Two Stream [24]	59.4%	Two Stream [24]	88.0%
TDD+FV	63.2%	TDD+FV	90.3%
Our best result	**65.9%**	Our best result	**91.5%**

Table 4. Comparison of TDD to the state of the art. We separately present the results of TDDs and our best results obtained with early fusion of TDDs and iDTs.

6. Conclusions

This paper has proposed an effective video presentation, called trajectory-pooled deep-convolutional descriptor (TDD), which integrates the advantages of hand-crafted and deep-learned features. Deep architectures are utilized to learn discriminative convolutional feature maps, and then the strategies of trajectory-constrained sampling and pooling are adopted to aggregate these convolutional features into TDDs. Our features achieve superior performance on two datasets for action recognition, as evidenced by comparison with the state-of-the-art methods.

Acknowledgement

This work is supported by a donation of Tesla K40 GPU from NVIDIA Corporation. Limin Wang is supported by Hong Kong PhD Fellowship. Yu Qiao is the corresponding author and supported by National Natural Science Foundation of China (91320101, 61472410), Shenzhen Basic Research Program (JCYJ20120903092050890, JCYJ20120617114614438, JCYJ20130402113127496), 100 Talents Program of CAS, and Guangdong Innovative Research Team Program (No.201001D0104648280).

References

[1] J. K. Aggarwal and M. S. Ryoo. Human activity analysis: A review. *ACM Comput. Surv.*, 43(3):16, 2011. 1

[2] H. Bay, T. Tuytelaars, and L. J. V. Gool. SURF: speeded up robust features. In *ECCV*, 2006. 3

[3] Z. Cai, L. Wang, X. Peng, and Y. Qiao. Multi-view super vector for action recognition. In *CVPR*, 2014. 8

[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. 6

[5] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *CVPR*, 2005. 5

[6] J. Deng, W. Dong, R. Socher, L. Li, K. Li, and F. Li. ImageNet: A large-scale hierarchical image database. In *CVPR*, 2009. 4, 6

[7] P. Dollár, V. Rabaud, G. Cottrell, and S. Belongie. Behavior recognition via sparse spatio-temporal features. In *VS-PETS*, 2005. 1, 2

[8] M. A. Fischler and R. C. Bolles. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM*, 24(6), 1981. 4

[9] S. Ji, W. Xu, M. Yang, and K. Yu. 3D convolutional neural networks for human action recognition. *TPAMI*, 35(1), 2013. 2, 3

[10] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. B. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. *CoRR*, abs/1408.5093. 6

[11] Y.-G. Jiang, J. Liu, A. Roshan Zamir, I. Laptev, M. Piccardi, M. Shah, and R. Sukthankar. THUMOS challenge: Action recognition with a large number of classes, 2013. 2, 6

[12] A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar, and L. Fei-Fei. Large-scale video classification with convolutional neural networks. In *CVPR*, 2014. 2, 3, 8

[13] A. Kläser, M. Marszalek, and C. Schmid. A spatio-temporal descriptor based on 3D-gradients. In *BMVC*, 2008. 2

[14] A. Krizhevsky, I. Sutskever, and G. E. Hinton. ImageNet classification with deep convolutional neural networks. In *NIPS*, 2012. 2, 3, 5

[15] H. Kuehne, H. Jhuang, E. Garrote, T. Poggio, and T. Serre. HMDB: A large video database for human motion recognition. In *ICCV*, 2011. 1, 2, 6, 8

[16] I. Laptev. On space-time interest points. *IJCV*, 64(2-3), 2005. 1, 2

[17] I. Laptev, M. Marszalek, C. Schmid, and B. Rozenfeld. Learning realistic human actions from movies. In *CVPR*, 2008. 1, 2, 5

[18] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. In *ISP*. IEEE Press, 2001. 1

[19] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *IJCV*, 60(2), 2004. 5

[20] A. Patron-Perez, M. Marszalek, I. Reid, and A. Zisserman. Structured learning of human interactions in TV shows. *TPAMI*, 34(12), 2012. 1

[21] X. Peng, L. Wang, X. Wang, and Y. Qiao. Bag of visual words and fusion methods for action recognition: Comprehensive study and good practice. *CoRR*, abs/1405.4506, 2014. 8

[22] S. Sadanand and J. J. Corso. Action bank: A high-level representation of activity in video. In *CVPR*, 2012. 2

[23] J. Sánchez, F. Perronnin, T. Mensink, and J. J. Verbeek. Image classification with the Fisher vector: Theory and practice. *IJCV*, 105(3), 2013. 7

[24] K. Simonyan and A. Zisserman. Two-stream convolutional networks for action recognition in videos. *NIPS*, 2014. 1, 2, 3, 4, 5, 6, 7, 8

[25] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014. 3

[26] K. Soomro, A. R. Zamir, and M. Shah. UCF101: A dataset of 101 human actions classes from videos in the wild. *CoRR*, abs/1212.0402, 2012. 1, 2, 6, 8

[27] C. Sun and R. Nevatia. Large-scale web video event classification by use of Fisher vectors. In *WACV*, 2013. 7

[28] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. *CoRR*, abs/1409.4842, 2014. 3

[29] G. W. Taylor, R. Fergus, Y. LeCun, and C. Bregler. Convolutional learning of spatio-temporal features. In *ECCV*, 2010. 2, 3

[30] H. Wang, A. Kläser, C. Schmid, and C.-L. Liu. Dense trajectories and motion boundary descriptors for action recognition. *IJCV*, 103(1), 2013. 1, 2, 3, 8

[31] H. Wang and C. Schmid. Action recognition with improved trajectories. In *ICCV*, 2013. 1, 2, 3, 7, 8

[32] H. Wang and C. Schmid. LEAR-INRIA submission for the thumos workshop. In *ICCV Workshop on THUMOS Challenge*, 2013. 2, 7, 8

[33] H. Wang, M. M. Ullah, A. Kläser, I. Laptev, and C. Schmid. Evaluation of local spatio-temporal features for action recognition. In *BMVC*, 2009. 2

[34] L. Wang, Y. Qiao, and X. Tang. Mining motion atoms and phrases for complex action recognition. In *ICCV*, 2013. 2

[35] L. Wang, Y. Qiao, and X. Tang. Motionlets: Mid-level 3D parts for human motion recognition. In *CVPR*, 2013. 1, 2, 8

[36] L. Wang, Y. Qiao, and X. Tang. Latent hierarchical model of temporal structure for complex activity classification. *TIP*, 23(2), 2014. 1

[37] L. Wang, Y. Qiao, and X. Tang. Video action detection with relational dynamic-poselets. In *ECCV*, 2014. 2

[38] X. Wang, L. Wang, and Y. Qiao. A comparative study of encoding, pooling and normalization methods for action recognition. In *ACCV*, 2012. 7

[39] G. Willems, T. Tuytelaars, and L. J. V. Gool. An efficient dense and scale-invariant spatio-temporal interest point detector. In *ECCV*, 2008. 2

[40] C. Zach, T. Pock, and H. Bischof. A duality based approach for realtime tv-L^1 optical flow. In *29th DAGM Symposium on Pattern Recognition*, 2007. 6

[41] M. D. Zeiler and R. Fergus. Visualizing and understanding convolutional networks. In *ECCV*, 2014. 3, 4, 5

[42] J. Zhu, B. Wang, X. Yang, W. Zhang, and Z. Tu. Action recognition with actons. In *ICCV*, 2013. 2