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 Institutes of Advanced Technology, CAS, China

Introduction

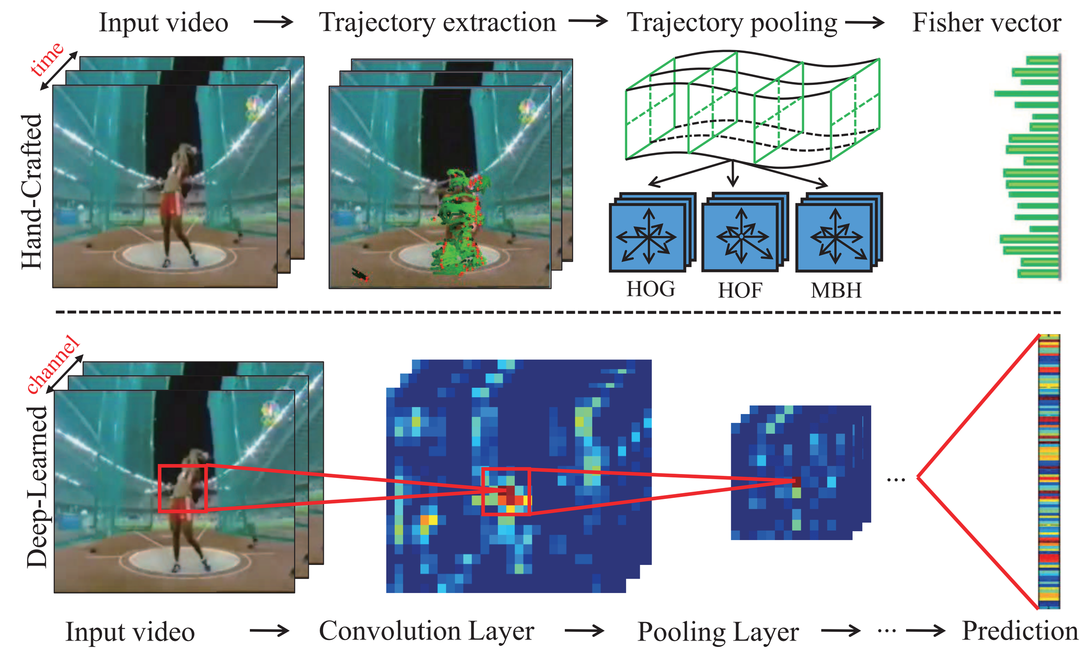


Figure 1: Two types of features in action recognition.

- **Goal**: Design new features sharing merits of both *hand-crafted* and *deep-learned* features for video representation.
- **Existing works**:
  - **Improved trajectories** [1]: (i) Extracting trajectories. (ii) Pooling local features along trajectories (HOG, HOF, MBH).
  - **Two-stream ConvNets** [2]: (i) Stacking frames or optical flow fields. (ii) Learning features for classification with CNNs.
- **Our idea**: *Trajectory-Pooled Deep-Convolutional Descriptors* (TDD):
  - (i) we exploit deep architectures to learn discriminative convolutional feature maps.
  - (ii) we perform trajectory-constrained pooling to aggregate these convolutional feature maps into effective descriptors.
- **Advantages**:
  - TDDs are automatically learned and contain high discriminative capacity compared with those hand-crafted features;
  - TDDs take account of the intrinsic characteristics of temporal dimension and introduce the strategies of trajectory-constrained sampling and pooling.

Improved Trajectories Revisited

- **Improved trajectories**:
  - Densely sampling a set of points and tracking them by media filtering:

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

  - Camera motion estimation: determining a homography matrix by using SURF feature matching and optical flow matching.
  - Camera motion estimation is capable of rectifying the optical flow fileds and removing the trajectories of background.
- **iDTs for TDDs**
  - Given a video V, we obtain a set of trajectories: $\mathbb{T}(V) = \{T_1, T_2, \cdots, T_K\}$
  - T_k denotes the k^{th} trajectory in the original spatial scale:

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

  - where (x_p^k, y_p^k, z_p^k) is the pixel position, and P is the length of trajectory ($P = 15$).

Deep Convolutional Descriptors

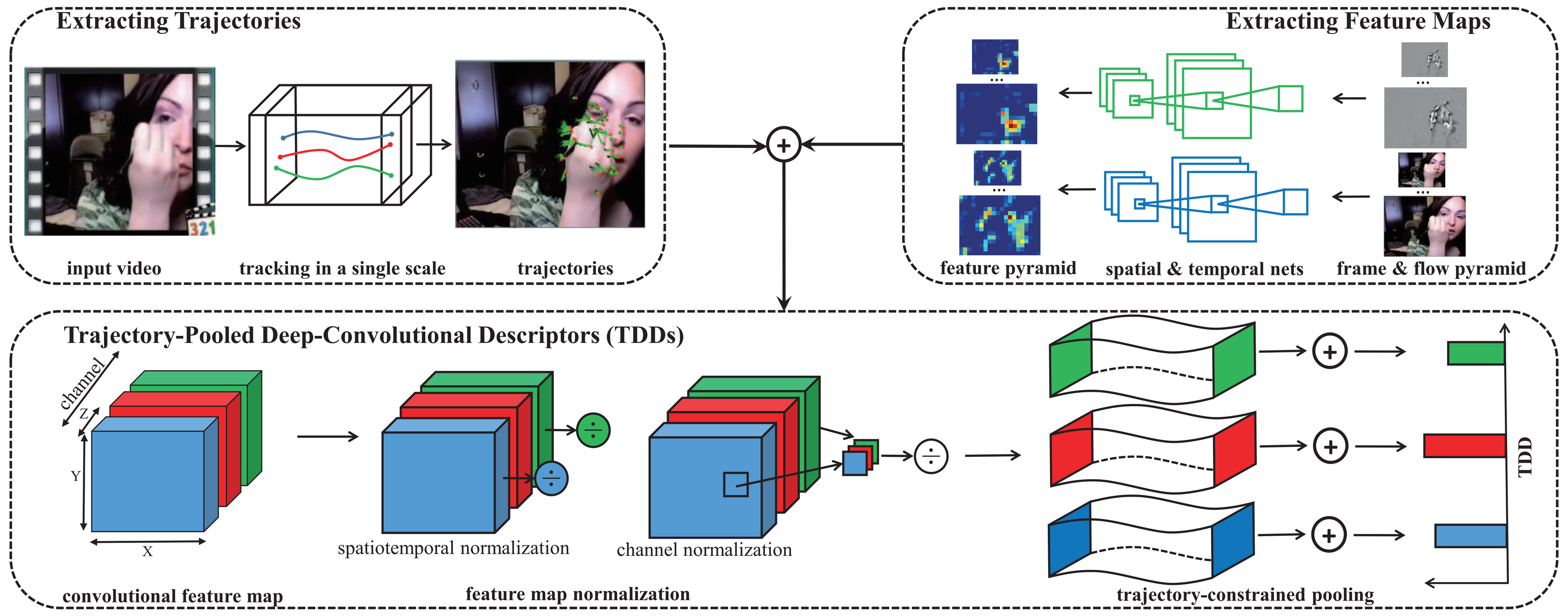


Figure 2: TDD extraction pipeline.

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.

- **Convolutional networks**:
  - We choose the two-stream ConvNets, which is composed of *spatial nets* and *temporal nets*.
  - **Spatial nets** capture static appearance cues and are trained on single frame images ($224\times224\times3$),
  - **Temporal nets** describe the dynamic motion information and are trained on the stacking of optical flow fields ($224\times224\times20$).
- **Convolutional feature maps**:
  - We use two-stream ConvNets as generic feature extractors:

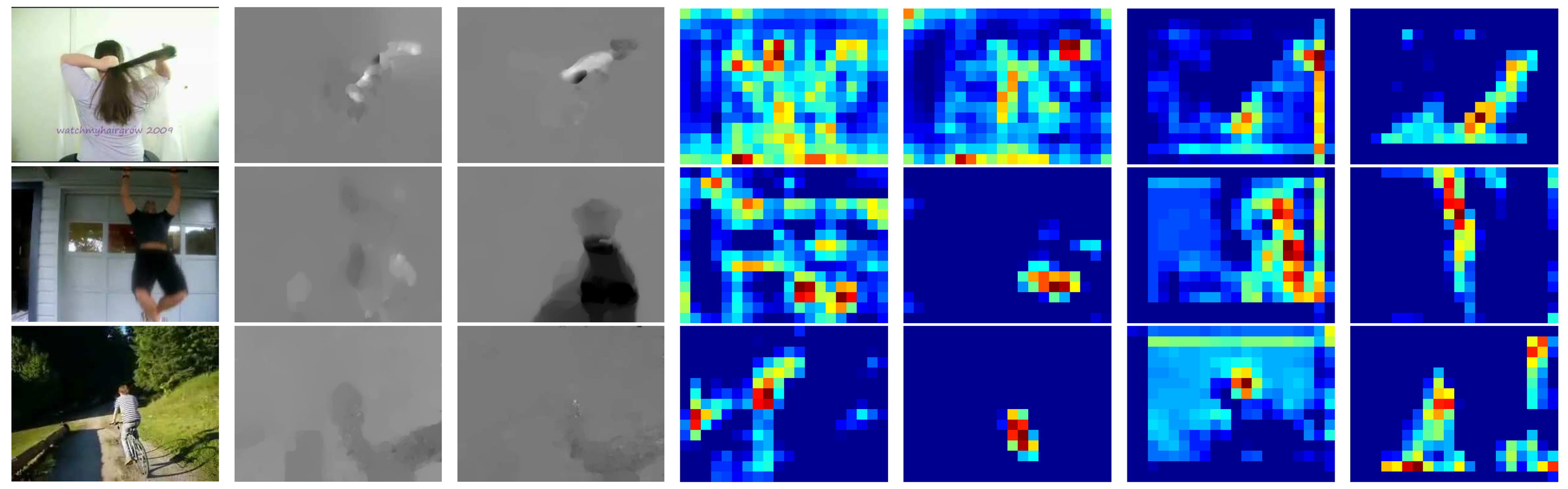
$$\mathbb{C}(V) = \{C_1^s, C_2^s, \cdots, C_M^s, C_1^t, C_2^t, \cdots, C_M^t\},$$

  where $C_m^s \in \mathbb{R}^{H_m\times W_m\times L\times N_m}$ is the m^{th} feature map.
  - We conduct zero padding of the layer's input with size $\lfloor k/2 \rfloor$ before each convolutional or pooling layer, with kernel size k.
  - A point with video coordinates (x_p, y_p, z_p) will be centered on $(r\times x_p, r\times y_p, z_p)$ in convolutional map, where r is map size ratio.
- **Trajectory-pooled descriptors**:
  - Two normalization methods:
    - Spatiotemporal normalization: $\widetilde{C}_{st}(x,y,z,n) = C(x,y,z,n)/\max \mathbf{V}_{st}^n$.
    - Channel normalization: $\widetilde{C}_{ch}(x,y,z,n) = C(x,y,z,n)/\max \mathbf{V}_{ch}^{x,y,z}$.
  - Sum pooling along trajectory:

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

  - *Multi-scale TDD extension*: we construct multi-scale pyramid representations of video frames and optical flow fields, which are transformed into multi-scale convolutional feature maps by ConvNets.

Experimental Results



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

Figure 3: Examples of video frames, optical flow fields, and feature maps.

	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.

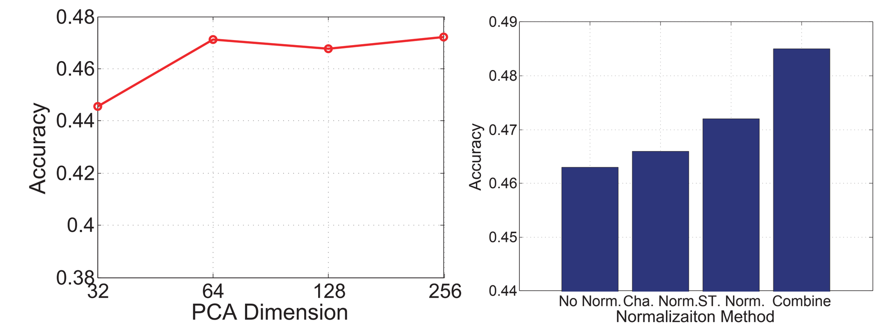


Figure 4: Left: Performance trend with varying PCA reduced dimension. Right: Comparison of different normalization methods.

Algorithm	HMDB51	UCF101	Algorithm	HMDB51	UCF101
HOG [1]	40.2%	72.4%	Spatial conv4	48.5%	81.9%
HOF [1]	48.9%	76.0%	Spatial conv5	47.2%	80.9%
MBH [1]	52.1%	80.8%	Spatial conv4 and conv5	**50.0%**	**82.8%**
HOF+MBH [1]	54.7%	82.2%	Temporal conv3	54.5%	81.7%
iDT [1]	**57.2%**	**84.7%**	Temporal conv4	51.2%	80.1%
Spatial net [2]	40.5%	73.0%	Temporal conv3 and conv4	**54.9%**	**82.2%**
Temporal net [2]	54.6%	83.7%	TDD	63.2%	90.3%
Two-stream ConvNets [2]	**59.4%**	**88.0%**	TDD and iDT	**65.9%**	**91.5%**

Table 3: Comparison of TDD with iDT features [1] and two-stream ConvNets [2].

HMDB51		UCF101	
STIP+BoVW	23.0%	STIP+BoVW	43.9%
Motionlets	42.1%	Deep Net	63.3%
DT+BoVW	46.6%	DT+VLAD	79.9%
DT+MVSV	55.9%	DT+MVSV	83.5%
iDT+FV	57.2%	iDT+FV	85.9%
iDT+HSV	61.1%	iDT+HSV	87.9%
Two Stream	59.4%	Two Stream	88.0%
TDD+FV	63.2%	TDD+FV	90.3%
Our best result	**65.9%**	Our best result	**91.5%**

Table 4: Comparison of TDDs to the state of the art.

Model and code is available at http://wanglimin.github.io/tdd/index.html

References

1. H. Wang and C. Schmid. Action recognition with improved trajectories. In ICCV, 2013.
2. K. Simonyan and A. Zisserman. Two-stream convolutional networks for action recognition in videos. In NIPS, 2014.