Eigen-Evolution Dense Trajectory Descriptors

Yang Wang, Vinh Tran, and Minh Hoai
Stony Brook University, Stony Brook, NY 11794-2424, USA
{wang33, tquangvinh, minhhoai}@cs.stonybrook.edu

***Abstract*— Trajectory-pooled Deep-learning Descriptors have been the state-of-the-art feature descriptors for human action recognition in video on many datasets. This paper improves their performance by applying the proposed eigen-evolution pooling to each trajectory, encoding the temporal evolution of deep learning features computed along the trajectory. This leads to Eigen-Evolution Trajectory (EET) descriptors, a novel type of video descriptor that significantly outperforms Trajectory-pooled Deep-learning Descriptors. EET descriptors are defined based on dense trajectories, and they provide complimentary benefits to video descriptors that are not based on trajectories. Empirically, we observe that the combination of EET descriptors and VideoDarwin outperforms the state-of-the-art methods on the Hollywood2 dataset, and its performance on the UCF101 dataset is close to the state-of-the-art.**

I. INTRODUCTION

Recognizing human actions in realistic videos is difficult due to various challenges such as background clutter, self occlusion, and viewpoint variation. An effective and efficient approach to handle these challenges is to use local visual space-time descriptors, an approach that does not require non-trivial pre-processing steps such as recognition and segmentation. In particular, one of the best types of video descriptor is Dense Trajectories [27] (or improved Dense Trajectories [28]), which remain competitive even in the recent surge of deep-learning descriptors [22], [25], [29]. In fact, most recent human action recognition methods [1], [8], [9], [14], [19], [29], [31] combine Dense Trajectories with deep learning features to obtain better performance.

Human actions, by definition, are caused by the motion of humans, and Dense Trajectories exploit this fact. The power and success of Dense Trajectories can be attributed to their ability to extract local descriptors along humans' motion. The key idea is to densely sample feature points and track them in the video based on optical flow. Typically, the feature points are tracked for 15 frames, leading to trajectories with a temporal span of 15 frames. The improved Dense Trajectories method [28] explicitly estimates the camera motion and prunes away trajectories from the background; this method only retains trajectories from humans or objects of interest. For each trajectory, multiple descriptors are computed along the trajectory to capture shape, appearance, and motion information. In particular, each trajectory leads to four feature vectors: Trajectory, Histogram of Oriented Gradients [4], Histogram of Optical Flow (HOF), and Motion Boundary Histogram (MBH). Recently, Wang *et al*. [29] proposed to improve trajectory features by aggregating deep-learning descriptors (based on Two-stream CNN [22]) along each trajectory. This led to the current state-of-the-art descriptors called Trajectory-pooled Deep-learning Descriptors (TDD).

However, TDD ignores the temporal evolution of features along the trajectories, which turns out to be important for classifying human actions. For each trajectory, TDD first computes a sequence of feature vectors (based on Two-stream CNN [22]), one vector at each time step of the trajectory. Subsequently, the feature vectors are averaged to represent the trajectory, and this is the main problem of TDD, because averaging is a summary operator that throws away much information, including the evolution of the features along the trajectory. In this work, we propose an alternative way to look at a sequence of feature vectors: it can be seen as an ordered set of one-dimensional functions. Each function corresponds to the evolution of a feature over time. Instead of using the average value to summarize a function, we propose to represent it as a linear combination of basis functions. The basis functions can be optimally determined using Principle Component Analysis (PCA) to find the principle directions of feature evolution. Finally, the trajectory is represented as one or several vectors of PCA coefficients. We refer to this process as Eigen-Evolution pooling. This process yields Eigen-Evolution Trajectory (EET) descriptors as illustrated in Figure 1.

Eigen-Evolution pooling is as efficient as average pooling. The basis functions for feature evolution can be precomputed, and computing an EET descriptor is equivalent to a single matrix vector multiplication. This process is still efficient even when multiple EET descriptors are needed, i.e., computing EET1, EET2, etc. The main computational bottleneck is not due to the pooling operation, but the process of computing CNN feature vectors along densely populated trajectories. To speed up this procedure, we first compute the video-wide convolution feature maps. We then use each trajectory's coordinates to access its corresponding deep-learning features.

EETs are local descriptors and they can be used in any standard recognition pipeline for action recognition. In this paper, we use EET descriptors with Fisher Vector encoding [21] and Support Vector Machines [26]. We evaluate EET descriptors on Hollywood2 [20] and UCF101 [24] datasets and find that they outperform the current trajectories-based descriptors. When using EET descriptors in conjunction with a video pooling method, the combined method outperforms

This project is partially supported by the National Science Foundation Award IIS-1566248 and Samsung Global Research Outreach.

978-1-5386-2335-0/18/$31.00

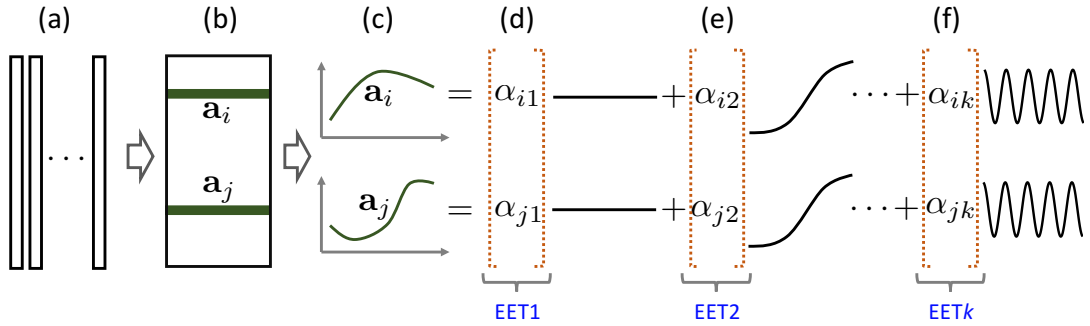


Fig. 1. **Illustration of the Eigen-Evolution Trajectory (EET) Descriptors.** (a): given a trajectory, we consider the spatiotemporal volume encapsulating the trajectory and extract a sequence of deep-learning feature vectors for the volume. (b, c): the sequence of feature vectors can be viewed as an ordered set of one-dimensional functions. Each function can be decomposed as a linear combination of basis functions. The set of coefficients that correspond to a basis function defines an EET descriptor. We refer to them as EET1, EET2, $\cdots$, based on the order of the basis functions (subfigures d, e, f).

the current state-of-the-art recognition performance on the Hollywood2 dataset.

II. Related Works

Hand-crafted descriptors. Feature descriptors are important for human action recognition, and much previous work has been devoted to the design of local features [5], [15], [18], [28], [34], which are often robust to background clutter, self occlusion, and viewpoint variation in realistic video clips. There are various ways to extract informative regions from a video clip. For instance, Space Time Interest Points [18] use 3D Harris corner detector, while Cuboid [5] uses temporal Gabor filters. In this paper, we choose the improved Dense Trajectories (iDT) [28] due to its good performance. Each trajectory leads to four feature vectors: Trajectory, Histograms of Oriented Gradient [4], Histograms of Optical Flow, and Motion Boundary Histograms. These 4 descriptors are computed along the trajectory to capture information about shape, appearance, motion, and the change of motion, respectively.

Deep-learning descriptors. Deep learning has achieved great success in image based visual recognition tasks [11], [12], [16], [23]. There has been some attempts to develop deep architectures for action recognition [22], [25], [30] as well. Wang *et al*. [29] propose to improve dense trajectories descriptors by aggregating deep-learning descriptors (based on Two-stream CNN [22]) along each trajectory. This leads to the current state-of-the-art descriptors called Trajectory-pooled Deep-learning Descriptors (TDD).

Pooling methods. We propose a method for pooling information along a trajectory. Our method encodes the temporal evolution along the trajectory, addressing the problem of the averaging pooling. We are not the first to realize the problem of average pooling. Hoai & Zisserman [13] proposed to use the orderly weighted averaging instead of averaging. This approach has also been applied to spatial pooling [32], [33].

There is a connection between the proposed eigen-evolution pooling with rank pooling [10]. Rank pooling can be applied to any temporal sequence. A popular method is VideoDarwin, which applies rank pooling to the sequence of video frames. Rank pooling has been applied to another extreme level of image pixels to produce dynamic images focusing on humans and objects of motion [1].

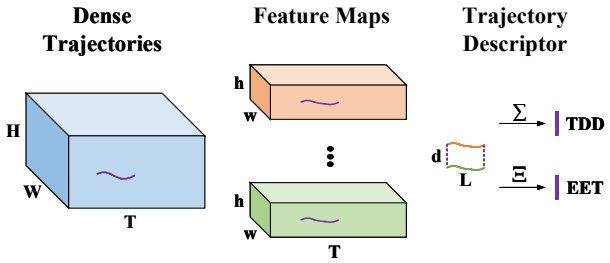


Fig. 2. **Illustration of computing deep-learning descriptors for trajectories.** Given a video, We extract the dense trajectories and the convolutional feature maps. Each trajectory is associated with a sequence of feature vectors $\mathbf{f}_1, \cdots, \mathbf{f}_L \in \mathbb{R}^d$, with d being the number of CNN feature maps and L the length of the trajectory. TDD applies Average Pooling to represent a trajectory. We propose to use Eigen-Evolution Pooling instead.

III. Eigen-Evolution Descriptors

In this section, we describe Eigen-Evolution Trajectory (EET) descriptors, which are founded on dense trajectories, deep-learning features, and the proposed eigen-evolution pooling.

A. Improved Dense Trajectories (iDT)

EET descriptors are defined based on dense trajectories. To extract dense trajectories, we use the improved implementation of [28] instead of the original dense trajectories [27]. We customize the implementation of [28] with two modifications: (i) videos are normalized to have the height of 360 pixels, and (ii) frames are extracted at 25 fps. These modifications are added to standardize the feature extraction procedure across videos and datasets. They did not significantly alter the performance of the action recognition system. Note that each trajectory is specified as a sequence of $L = 16$ points in the video space, corresponding to a temporal span of 15 frames. The coordinates of these points on the trajectory are used to pool deep convolutional features.

B. Eigen-Evolution Pooling

The pipeline of computing deep-learning descriptors for trajectories is illustrated in Figure 2. Given a video, we first

extract thousands of trajectories as local interest points in the video space, and densely apply a CNN to extract the convolutional feature maps. Subsequently, for each trajectory, we consider a spatiotemporal volume encapsulating that trajectory. The length of the volume is the same as the length of the trajectory, and the width and and height of the volume are defined by the scale of the trajectory. Each trajectory-based volume is associated with a sequence of feature vectors $\mathbf{f}_1, \cdots, \mathbf{f}_L \in \mathbb{R}^d$, with d being the number of CNN feature maps and L the length of the trajectory. The current state-of-the-art trajectory-based method, TDD [29], uses the average of $\mathbf{f}_1, \cdots, \mathbf{f}_L$ to represent the trajectory, i.e., $\mathbf{u}_0 = \frac{1}{L}\sum_{t=1}^{L} \mathbf{f}_t$. Average pooling is a simple approach, but it throws away the temporal progression of the feature vectors along the trajectory. In order to capture the temporal evolution of the feature vectors, we propose eigen-evolution pooling to replace average pooling.

Let $\mathbf{F} = [\mathbf{f}_1, \cdots, \mathbf{f}_L] \in \mathbb{R}^{d \times L}$ represent a sequence of feature vectors. Instead of considering $\mathbf{F}$ as a collection of columns, we propose to look at $\mathbf{F}$ as a list of rows. Let $\mathbf{a}_i$ denote the row of $\mathbf{F}$, i.e., $\mathbf{F} = [\mathbf{a}_1, \cdots, \mathbf{a}_d]^T$. Each row $\mathbf{a}_i \in \mathbb{R}^L$ is a one-dimensional function that corresponds to the evolution of a feature over time. Instead of using the average value to summarize a function, we propose to represent it as a linear combination of basis functions.

We propose to use a set of basis functions that minimizes the reconstruction errors. Suppose we have a set of orthonormal basis functions $\mathbf{G} = [\mathbf{g}_1, \cdots, \mathbf{g}_k] \in \mathbb{R}^{L \times k}, \mathbf{G}^T\mathbf{G} = \mathbf{I}_k$ with a coupled encoder $\mathbf{W} \in \mathbb{R}^{k \times L}$, such that a function $\mathbf{a}$ can be decomposed into a linear combination of basis functions $\mathbf{Gc}, \mathbf{c} \in \mathbb{R}^k$, and the coefficient vector $\mathbf{c}$ can be obtained as the product between the input function and the encoder, i.e., $\mathbf{c} = \mathbf{Wa}$. Note that if k (the number of basis functions) is small, the reconstructed function $\mathbf{Gc} = \mathbf{GWa}$ might not be exactly the same as the input function $\mathbf{a}$. In order to keep as much information as possible, we propose to find the optimal set of basis functions $\mathbf{G}$, by minimizing the reconstruction error.

$$\mathbf{G}^* = \underset{\mathbf{G}^T\mathbf{G}=I}{\operatorname{argmin}} \sum_{\mathbf{F}} \sum_{i} \|\mathbf{GWa}_i - \mathbf{a}_i\|^2 \tag{1}$$

In the above, the first summation $\sum_{\mathbf{F}}$ refers to the enumeration over multiple trajectories; each trajectory leads to a feature matrix $\mathbf{F}$. The second summation $\sum_i$ enumerates through the rows of $\mathbf{F}$. By setting the gradient of the loss with respect to the encoder $\frac{\partial L}{\partial \mathbf{W}} = 2(\mathbf{W} - \mathbf{G}^T)\sum_{\mathbf{F}}\sum_i \mathbf{a}_i\mathbf{a}_i^T = 0$, we have $\mathbf{W} = \mathbf{G}^T$, i.e., the encoder should be exactly the transpose of the basis functions. Then the Eq. (1) can be simplified to:

$$\mathbf{G}^* = \underset{\mathbf{G}^T\mathbf{G}=I}{\operatorname{argmax}} \sum_{j=1}^{k} \mathbf{g}_j^T \mathbf{B} \mathbf{g}_j, \tag{2}$$

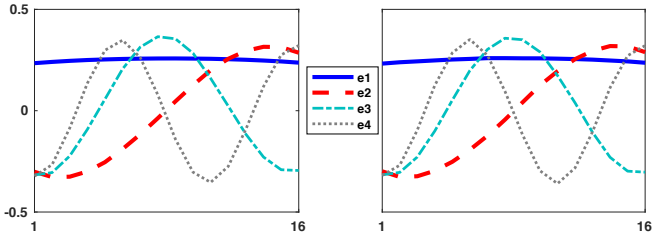
$$\text{where } \mathbf{B} = \sum_{\mathbf{F}} \sum_{i} \mathbf{a}_i\mathbf{a}_i^T = \sum_{\mathbf{F}} \mathbf{F}^T\mathbf{F} \tag{3}$$

The matrix $\mathbf{B}$ is essentially a covariance matrix. It is the covariance matrix between time steps, not the covariance matrix between features. The optimal set of basis functions $\mathbf{G}^*$ can be found using eigen decomposition:

$$\mathbf{B} = \sum_{i=1}^{L} \lambda_i \mathbf{e}_i \mathbf{e}_i^T, \quad \lambda_1 \geq \cdots \geq \lambda_L. \tag{4}$$

where $\mathbf{e}_1, \cdots, \mathbf{e}_L$ are the eigen vectors with corresponding eigen values $\lambda_1, \cdots, \lambda_L$. Since $\mathbf{B}$ is the covariance of features over times, we refer to $\mathbf{e}_1, \cdots, \mathbf{e}_L$ as eigen evolution functions or simply eigen evolutions. For smallest possible reconstruction error, we must have $\mathbf{g}_1 = \mathbf{e}_1, \cdots, \mathbf{g}_k = \mathbf{e}_k$.

For a basis function $\mathbf{g}$ and a feature sequence $\mathbf{F}$, $\mathbf{Fg}$ is the vector of coefficients corresponding to the basis function $\mathbf{g}$. This has the same dimension as the feature vectors in $\mathbf{F}$ and we refer to it as an Eigen-Evolution Trajectory (EET) descriptor because $\mathbf{g}$ is an eigen-evolution function. With different basis functions $\mathbf{g}$'s, we capture behaviors at different evolution directions. When $\mathbf{e}_1, \mathbf{e}_2, \cdots, \mathbf{e}_k$ are used as the basis function, we obtain k different descriptors, which will be referred to as EET1, EET2, ..., EETk respectively.



(a) all features (b) single feature

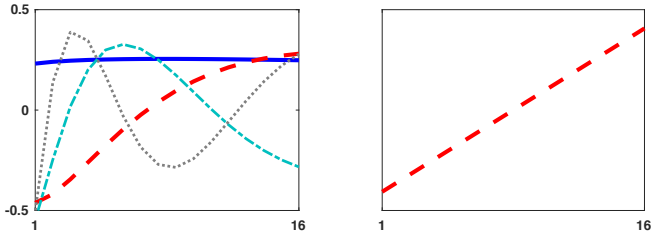
Fig. 3. **Visualizing Eigen-evolution functions.** (a): top four eigen-evolution functions for all features combined; the first eigen-evolution function is similar to an average function. (b): top four eigen-evolution functions of a single feature (chosen randomly from a list of 512 deep learning features). Eigen-evolution functions for a single feature and all features are similar.

C. Eigen-evolution functions

Figure 3(a) shows the top four eigen-evolution functions obtained by the procedure described above for the combination of all features extracted using the spatial-stream VGG-M model [3]. Not surprisingly, the first function is similar to the average function; the function is almost flat with a slight upward curve in the middle.

It is possible to use different set of basis functions for different features for optimal reduction of reconstruction errors, but we found no benefit of doing so. Figure 3(b) shows the top four eigen-evolutions obtained for a single feature that is chosen randomly from the list of all features in the deep learning feature vector associated with TDD. Comparing Figure 3(a) and Figure 3(b), we observe little difference between the two sets of basis functions. Therefore, for simplicity and efficiency, we propose to use a single set of eigen-evolution functions for all features.

We also find a surprising connection between the proposed eigen-evolution pooling and rank pooling [10]. Rank pooling, originally known as VideoDarwin [10], is a method to aggregate relevant information over time and encode the



(a): Eigen-evolution functions (b): Rank pooling weights

Fig. 4. **Eigen-evolution pooling and Rank pooling.** (a): top four eigen-evolution functions for L_2-normalized accumulated features; the same set of features used by rank pooling. (b): pooling weights of the rank pooling method, it is similar to the second eigen-evolution function in (a).

temporal progression. The idea is to find a direction in the feature space that preserves the temporal order of a sequence of feature vectors. Rank pooling, however, is computationally intensive, requiring solving a Rank SVM formulation every time we need to encode a sequence of feature vectors. To overcome the efficiency issue, Bilen *et al*. [1] proposed an approximate formulation for rank pooling. Given a sequence of feature vectors $\mathbf{f}_1, \cdots, \mathbf{f}_L$, they first compute the L_2-normalized accumulated feature vectors: $\mathbf{F}_t = \left\| \sum_{i=1}^{t} \mathbf{f}_i \right\|_2$. Approximate rank pooling is essentially the sum pooling of feature differences $(\mathbf{F}_j - \mathbf{F}_i)$ over all time intervals $1 \leq i < j \leq L$:

$$u_1 = \sum_{1 \leq i < j \leq L} (\mathbf{F}_j - \mathbf{F}_i) = \sum_{t=1}^{L} (2t - L - 1)\mathbf{F}_t. \quad (5)$$

The pooling weights of rank pooling is plotted in Figure 4(b). This function is similar to the second eigen-evolution function obtained for sequences of L_2-normalized accumulated feature vectors $\{\mathbf{F}_t\}_{t=1}^{L}$. Note that the eigen-evolution functions are learned from data and they are the principle evolution directions. In this case, it is interesting that the second principle direction is similar to the direction that preserves temporal progression. An advantage of eigen-evolution over rank pooling is that eigen-evolution is more general. Eigen-evolution can be applied to any sequence of feature vectors, not just L_2-normalized accumulated feature vectors. Furthermore, rank pooling only provides a single descriptor that corresponds to the second principle evolution direction, while eigen-pooling provides a principle approach to obtain multiple descriptors with complementary benefits.

D. Convolutional Features

Eigen evolution is a general pooling method that is not restricted to deep learning features. However, in this paper, we confine our experiments to CNN feature vectors. The procedure to compute CNN feature maps is flexible with a variety of options. We can choose which CNN model and which convolutional layer to be used to extract the feature maps. The spatial resolution of the input video can also vary, because the convolutional layers do not require a specific size for the input as in the case of a fully-connected layer. There are also several different methods for normalizing the feature maps. In the following, we discuss these options in detail.

1) Feature Map Extraction: We first discuss some options for extracting convolutional feature maps.

• Two-stream CNN. In principle, any kind of CNN architecture can be used to extract convolutional feature maps. In our implementation, we follow the original TDD paper and use a Two-stream CNN provided by [29]. The model is trained on UCF-101 dataset and it consists of a spatial and a temporal CNN. The spatial CNN is based on VGG-M-2048 model and fine-tuned with individual RGB frames (224×224×3); the temporal CNN has a similar structure, but its input is a volume of stacked optical flows (224×224×2F, where $F = 10$ is the number of flow maps).

To extract convolutional feature maps, we make two modifications to the original model. First, we make the model fully-convolutional by removing all the fully-connected layers. Second, we pad zeros to each convolution layer's input with the size of the pad being half the size of the convolutional kernels. This is to avoid the boundary-shrinking effect of when applying convolution.

• Input Scale. The size of the input RGB frames and optical flow images can vary, because we only use convolutional layers and convolutional layers do not require a fixed input size. Having a larger input size means each neuron will look at a smaller neighborhood (receptive field) and produce features at a finer scale. Combining descriptors at different input scales can improve the classification results.

• Spatial Feature Maps. We extract frames from videos at 25fps and resize them into a predefined scale ($360p \times 480p$). Subsequently we feed individual frames into the pre-trained spatial CNN and take the output of the 'conv4' or 'conv5' layer as the spatial feature maps for each video, as suggested in the original TDD method [29].

• Temporal Feature Maps. The process of computing temporal feature maps is similar to that of computing spatial feature maps, except we feed into the pre-trained temporal CNN with 10 consecutive flow images (with horizontal and vertical motion channels). The flow images are computed at 25fps using a GPU version of TVL1 algorithm, and subsequently resized into a predefined scale as well. For computing the temporal CNN descriptors, we used the 'conv3' or 'conv4' layer's output as the temporal feature maps.

2) Feature Map Normalization: After the feature maps are extracted, they are normalized. Traditionally, normalization has been widely used for hand-crafted local descriptors such as HOG, HOF, and MBH. Here we consider two normalization methods for convolutional feature maps: *in-channel* and *in-voxel*.

• In-channel normalization. Figure 5 (left) illustrates the in-channel normalization method, which is L_∞ normalization of each individual feature channel across the video's spatiotemporal volume. This normalization is to ensure each feature channel ranges in the same interval, and thus contributes equally to final representation.

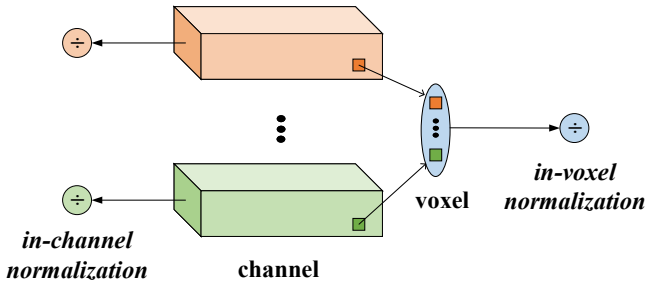


Fig. 5. **Illustration of Feature Map Normalization.** Left: *in-channel* normalization is performed within each individual feature channel; Right: *in-voxel* normalization is performed within each individual voxel.

• In-voxel normalization. Figure 5 (right) illustrates the in-voxel normalization method. It is L_∞ normalization of each individual voxel across the feature channels. This ensures each voxel ranges in the same interval, and consequently makes equal contribution to the final representation.

E. Fisher Vector encoding

After computing the local descriptors, we use Fisher Vector encoding [21] to represent an entire video. A Fisher Vector encodes both first and second order statistics of the feature descriptors (using a Gaussian Mixture Model (GMM)). Fisher Vector outerformed the bag-of-word approach for action classification [28].

For each type of descriptors, we first sample a subset of 1,000,000 data points. We use PCA to de-correlate the descriptors and reduce the dimension to D ($D = 64$ for CNN-based descriptors; D is half of the original dimension for iDT descriptors), and then train a GMM with $K = 256$ mixtures. Finally, each video is represented with a $2KD$ dimensional Fisher Vector, which is subsequently power normalized ($\alpha = 0.5$) and L_2 normalized, as in [21], [28].

After Fisher Vector encoding, we train one-vs-all SVMs [26] for action classification. After learning the classifiers, given a test video, we compute the probability of each action by normalizing the scores across actions with a softmax function [31]. To combine multiple descriptor types, we compute the average kernel, which is equivalent to concatenating the corresponding Fisher Vectors.

IV. Experiments

A. Datasets

We evaluate the performance of EET descriptors on two datasets: Hollywood2 [20] and UCF101 [24]. The Hollywood2 dataset has 12 action classes and contains 1707 video clips collected from 69 different Hollywood movies. The videos are split into a training set of 823 videos and a testing set of 884 videos. The training and testing videos come from different movies. We augment the training set with horizontally flipped training videos, and report the mean Average Precision as the performance measure on Hollywood2. The UCF101 dataset contains 101 action categories with 13,320 action videos collected from YouTube. Each category has at least 100 video clips. The dataset has three different training/test splits. We use top-1 accuracy as the evaluation metric for each training/test split in UCF101.

TABLE I

Chosen configuration for EET descriptors

Component	**Used Configuration**
model	Two-stream VGG-M-2048
scale	$360p \times 480p$ (UCF101) $180p \times 240p$ (Hollywood2)
layer	spatial: conv4 temporal: conv3
normalization	in-voxel

TABLE II

EET with different types of feature sequences

Sequence	Preprocess	EET1	EET2	EET3
$\mathbf{f}_t$	none	77.5	81.8	80.5
$\mathbf{F}_t$ (Eq.5)	accum + L_2	78.0	82.3	81.7

Performance of EET descriptors with two different types of feature sequences. $\{\mathbf{f}_t\}$: original feature sequence. $\{\mathbf{F}_t\}$: the L_2-normalized accumulative sequence (Eq. (5)).

B. Configuration of Feature Maps

The feature extraction process as four configurable components: *model*, *layer*, *scale*, *normalization*. For *model* and *layer*, we adopt the same models and best layers used in the TDD paper [29]. For *scale*, we compare three input sizes: $360p \times 480p$, $240p \times 320p$, $180p \times 240p$ and find that the EET descriptors perform best at scale $360p \times 480p$ on UCF101 and $180p \times 240p$ on Hollywood2. For *normalization*, we find that in-voxel normalization is slightly better than in-channel normalization for EET descriptors. The reason might be, with in-voxel normalization, each voxel is equally represented so eigen-evolution pooling is more effective in discerning the changes among adjacent voxels. In the end, the setting given in Table I is used for the evaluation of EET descriptors.

C. Evaluation of EET Descriptors

• EET with sequence preprocessing. As explained before, eigen-evolution pooling can be applied to the original feature sequence $\{\mathbf{f}_t\}$ or the L_2-normalized accumulative sequence $\{\mathbf{F}_t\}$ (Eq. (5)). Table II compares the performance of EET on these two sequences. As can be observed, the performance of EET(1,2,3) is consistently better when applied on the sequence $\{\mathbf{F}_t\}$. Hereafter, we will use the $\{\mathbf{F}_t\}$ sequence to compute EET descriptors in our experiments.

• Eigen-Evolution Pooling & Rank Pooling. We have discussed a surprising connection between the eigen-evolution pooling and the rank pooling. Table III compares the performance of EET with the rank-pooled descriptors. When being used alone, EET2 outperforms EET1 and EET3, and EET2 achieves similar performance with rank pooling method. When combined, the EET descriptors outperform the rank-pooled descriptor, owing to the additional information which

TABLE III

EET DESCRIPTORS VERSUS RANK-POOLED DESCRIPTORS

Rank-pooled	EET1	EET2	EET3	EET1+2	EET2+3	EET1+2+3
82.4	78.0	82.3	81.7	82.8	83.4	83.8

TABLE IV

COMPARING EET AND TDD – TRAJECTORY-LEVEL POOLING

Dataset	FeatureMap	TDD	EET	Improvement
Hollywood2	spatial	43.5	54.4	10.9
	temporal	63.1	66.0	2.9
	2-stream	64.7	68.7	4.0
UCF101 (split 1)	spatial	77.5	84.4	6.9
	temporal	77.9	81.0	3.1
	2-stream	86.1	88.8	2.7

EET descriptors significantly outperform TDD on both the spatial and temporal feature maps, indicating the importance of preserving the temporal evolution of appearance and motion along the trajectories.

is ignored by rank-pooling. To balance between the performance and efficiency, we propose to combine EET2 and EET3 as the default EET descriptor.

• EET & TDD. We evaluate the performance of the proposed EET descriptors (computed with only one layer and one scale per stream, as specified in Table I) on both Hollywood2 and UCF101 datasets. The experimental results are summarized in Table IV. Note that we can apply eigen-evolution pooling in both forward and backward directions (also for rank pooling in [10], [9]). The forward and backward EET descriptors are then fused by concatenating their corresponding Fisher Vectors. As can be observed, compared to the TDD descriptors, EET descriptors significantly improve the recognition performance on both the spatial and temporal streams, indicating the importance of preserving the temporal evolution of appearance and motion along the trajectories. The advantage is most evident on the spatial stream, where EET outperforms TDD by a large margin on both datasets (from 43.5% to 54.5% on Hollywood2; from 77.5% to 84.4% on UCF101).

D. Video-level pooling methods for EET descriptors

EET descriptors are local; each descriptor is confined to the spatio-temporal volume of a trajectory. For a video, EET descriptors can be densely computed, and they must be aggregated to produce the feature vector representation for the video. We refer to this procedure as video-level pooling, and there are different ways for doing so. In this section, we compare the performance of different video-level pooling methods for EET descriptors.

Each trajectory has a temporal span of 15 frames, and we assign each trajectory to its middle frame (the 8^{th} frame). Each frame is therefore associated with a set of trajectories, and we can compute an unnormalized Fisher Vector for each frame. Subsequently, a video can be represented as a sequence of frame-wise unnormalized Fisher Vectors $\phi_1, \cdots, \phi_T$. To compress the information from the sequence into a single feature vector, we can apply different video-level pooling methods. Until now, we have been using average pooling to represent an entire video as $\phi = norm(\sum_{t=1}^{T} \phi_t)$. As aforementioned, we set the $norm$ function to be power normalization ($\alpha = 0.5$) followed by L_2 normalization. We can also apply video-level rank pooling, i.e., VideoDarwin [10]). Aside from average pooling (AP) and rank pooling (RP), we also propose a simple and effective pooling method called hierarchical average pooling (HAP). To obtain the final feature vector using hierarchical average pooling, we run a sliding window (size: 20, stride: 1) and perform average pooling and normalization within each window, and subsequently perform average pooling over the entire video again. We suggest to use HAP instead of AP, because HAP consistently outperforms AP in our experiments.

TABLE V

COMPARISON OF VIDEO-LEVEL POOLING METHODS.

Dataset	Descriptor	AP	HAP	RP	HAP+RP
Holly.	EET-spatial	54.4	54.5	50.7	54.4
	EET-temporal	66.0	67.2	62.5	66.6
	EET-2-stream	68.7	69.9	65.0	69.2
	iDT	69.4	71.6	73.5	73.8
	EET-2-stream (HAP) + iDT (HAP + RP)				76.8
UCF101 (split 1)	EET-spatial	84.4	84.8	83.5	84.4
	EET-temporal	81.0	82.8	81.9	82.8
	EET-2-stream	88.8	89.6	88.2	89.2
	iDT	84.0	85.4	85.0	85.7
	EET-2-stream (HAP) + iDT (HAP + RP)				89.6

For EET, Hierarchical average pooling (HAP) outperforms both rank pooling (RP) and average pooling (AP). On Hollywood2, the fusion of EET and iDT significantly improves the performance.

Table V compares the performance different video-level pooling methods (for EET descriptors and iDT descriptors) on Hollywood2 and UCF101 datasets. As can be observed, hierarchical average pooling (HAP) outperforms both rank pooling (RP) and average pooling (AP) for EET descriptors. For iDT, the combination of HAP and RP achieves the best performance. Interestingly, the improvement of rank pooling over average pooling at the video level is more evident for the temporal stream, whereas at the trajectory level the improvement is higher for the spatial stream. On the Hollywood2 dataset, the fusion between EET and iDT descriptors achieves significant improvement. On the UCF101 dataset, however, the combination of EET with iDT does not lead to much better performance.

E. Comparison to the state-of-the-art

To achieve the best recognition performance, we compute EET descriptors with multiple convolutional layers (spatial: 'conv4'+'conv5'; temporal: 'conv3'+'conv4') and input scales ($180p \times 240p$, $240p \times 320p$, $360p \times 480p$). We also fuse EET with iDT descriptors the same way as did in Table V. Table VI and Table VII compare the proposed method with the state-of-the-art methods. On the Hollywood2 dataset, our method outperforms previous state-of-the-art by 2%. On the UCF101 dataset, we achieve performance on par with previous state-of-the-art methods [2], [6], [7], [8], [30], which used a much deeper architecture (Inception, ResNet)

TABLE VI

COMPARISON WITH STATE-OF-THE-ART ON HOLLYWOOD2.

Method	mAP(%)
2-stream TSN (pretrained) [30]	*62.6
iDT [28]	64.7
Non-Action [31]	71.0
SSD + RCS [13]	73.6
VideoDarwin [10]	73.7
HRP + iDT [9]	76.7
TDD [29]	*68.4
TDD + iDT [29]	*73.0
EET (Proposed)	74.5
EET + iDT (Proposed)	78.7

* denotes results from our implementation.

TABLE VII

COMPARISON WITH STATE-OF-THE-ART ON UCF101.

Method	Accuracy(%)
iDT [28]	85.9
Two Stream CNN [22]	88.0
Multi-skip Feature Stacking [17]	89.1
DIN + iDT [1]	89.1
C3D + iDT [25]	90.4
HRP + iDT [9]	91.4
RNN-FV + iDT [19]	94.1
3-stream TSN [30]	94.2
ST-ResNet + iDT [6]	94.6
ST-Multiplier + iDT [7]	94.9
I3D [2]	98.0
TDD [29]	90.3
TDD + iDT [29]	91.5
EET (Proposed)	91.8
EET + iDT (Proposed)	92.2
EET + iDT + 2-stream TSN (Proposed)	94.5

and additional training data (Kinectics). In principle, the proposed method can be used with these newer CNN models for further improvement. The fairest comparison is between EET, TDD, and 2-Stream CNN, all of which used the same architecture (VGG-M) and training data.

V. CONCLUSIONS

In this paper, we have proposed Eigen-Evolution Trajectory (EET) descriptors, which integrate the benefits of dense trajectories, deep-learning features, and eigen-evolution pooling. Deep architectures are utilized to extract discriminative feature maps, and eigen-evolution pooling is applied to each trajectory, capturing the temporal evolution of appearance and motion along the trajectory. The EET descriptors significantly outperform the previous state-of-the-art trajectory-based descriptors. Combining EET descriptors that preserve short-term evolution and VideoDarwin that captures long-term dynamics, we are able to advance the state-of-the-art performance on the Hollywood2 dataset.

REFERENCES

[1] H. Bilen, B. Fernando, E. Gavves, A. Vedaldi, and S. Gould. Dynamic image networks for action recognition. In *Proc. CVPR*, 2016.

[2] J. Carreira and A. Zisserman. Quo vadis, action recognition? a new model and the kinetics dataset. In *Proc. CVPR*. IEEE, 2017.

[3] K. Chatfield, K. Simonyan, A. Vedaldi, and A. Zisserman. Return of the devil in the details: Delving deep into convolutional nets. In *Proc. BMVC.*, 2014.

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

[5] P. Dollár, V. Rabaud, G. Cottrell, and S. Belongie. Behavior recognition via sparse spatio-temporal features. In *ICCV Workshop on Visual Surveillance & Performance Evaluation of Tracking and Surveillance*, 2005.

[6] C. Feichtenhofer, A. Pinz, and R. Wildes. Spatiotemporal residual networks for video action recognition. In *NIPS*, 2016.

[7] C. Feichtenhofer, A. Pinz, and R. P. Wildes. Spatiotemporal multiplier networks for video action recognition. In *Proc. CVPR*. IEEE, 2017.

[8] C. Feichtenhofer, A. Pinz, and A. Zisserman. Convolutional two-stream network fusion for video action recognition. In *Proc. CVPR*, 2016.

[9] B. Fernando, P. Anderson, M. Hutter, and S. Gould. Discriminative hierarchical rank pooling for activity recognition. In *Proc. CVPR*, 2016.

[10] B. Fernando, E. Gavves, J. O. M., A. Ghodrati, and T. Tuytelaars. Modeling video evolution for action recognition. In *Proc. CVPR*, 2015.

[11] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proc. CVPR*, 2014.

[12] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *Proc. CVPR*, 2016.

[13] M. Hoai and A. Zisserman. Improving human action recognition using score distribution and ranking. In *Proc. ACCV*, 2014.

[14] M. Jain, J. C. van Gemert, and C. G. Snoek. What do 15,000 object categories tell us about classifying and localizing actions? In *Proc. CVPR*, 2015.

[15] Y. Kong, B. Satarboroujeni, and Y. Fu. Hierarchical 3d kernel descriptors for action recognition using depth sequences. In *Proc. Int. Conf. Autom. Face and Gesture Recog.* IEEE, 2015.

[16] A. Krizhevsky, I. Sutskever, and G. Hinton. ImageNet classification with deep convolutional neural networks. In *NIPS*, 2012.

[17] Z. Lan, M. Lin, X. Li, A. G. Hauptmann, and B. Raj. Beyond gaussian pyramid: Multi-skip feature stacking for action recognition. In *Proc. CVPR*, 2015.

[18] I. Laptev. On space-time interest points. *IJCV*, 64(2–3):107–123, 2005.

[19] G. Lev, G. Sadeh, B. Klein, and L. Wolf. Rnn fisher vectors for action recognition and image annotation. In *Proc. ECCV*, 2016.

[20] M. Marszalek, I. Laptev, and C. Schmid. Actions in context. In *Proc. CVPR*, 2009.

[21] F. Perronnin, J. Sánchez, and T. Mensink. Improving the fisher kernel for large-scale image classification. In *Proc. ECCV*, 2010.

[22] K. Simonyan and A. Zisserman. Two-stream convolutional networks for action recognition in videos. In *NIPS*, 2014.

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

[24] K. Soomro, A. R. Zamir, and M. Shah. UCF101: A dataset of 101 human action classes from videos in the wild. Technical Report CRCV-TR-12-01, University of Central Florida, 2012.

[25] D. Tran, L. Bourdev, R. Fergus, L. Torresani, and M. Paluri. Learning spatiotemporal features with 3d convolutional networks. In *Proc. ICCV*. IEEE, 2015.

[26] V. Vapnik. *Statistical Learning Theory*. Wiley, New York, NY, 1998.

[27] H. Wang, A. Klaser, C. Schmid, and C.-L. Liu. Action recognition by dense trajectories. In *Proc. CVPR*, 2011.

[28] H. Wang and C. Schmid. Action recognition with improved trajectories. In *Proc. ICCV*, 2013.

[29] L. Wang, Y. Qiao, and X. Tang. Action recognition with trajectory-pooled deep-convolutional descriptors. In *Proc. CVPR*, 2015.

[30] L. Wang, Y. Xiong, Z. Wang, Y. Qiao, D. Lin, X. Tang, and L. Van Gool. Temporal segment networks: Towards good practices for deep action recognition. In *Proc. ECCV*. Springer, 2016.

[31] Y. Wang and M. Hoai. Improving human action recognition by non-action classification. In *Proc. CVPR*, 2016.

[32] Z. Wei, H. Adeli, M. Hoai, G. Zelinsky, and D. Samaras. Learned region sparsity and diversity also predict visual attention. In *NIPS*, 2016.

[33] Z. Wei and M. Hoai. Region ranking SVMs for image classification. In *Proc. CVPR*, 2016.

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