Towards Efficient Action Recognition: Principal Backpropagation for Training Two-Stream Networks

Wenbing Huang*, Lijie Fan*, Mehrtash Harandi, Lin Ma, Huaping Liu, Wei Liu, Chuang Gan

***Abstract*—In this paper, we propose a novel Principal Backpropagation Networks (PBNets) to revisit the backpropagation algorithms commonly used in training two-stream networks for video action recognition. We content that existing approaches always take all the frames/snippets for the backpropagation not optimal for video recognition since the desired actions only occur in a short period within a video. To remedy these drawbacks, we design a Watch-and-Choose mechanism. In particular, the watching stage exploits a dense snippet-wise temporal pooling strategy to discover the global characteristic for each input video, while the choosing phase only backpropagates a small number of representative snippets that are selected with two novel strategies, i.e. Max-rule and KL-rule. We prove that with the proposed selection strategies, performing the backpropagation on the selected subset is capable of decreasing the loss of the whole snippets as well. The proposed PBNets are evaluated on two standard video action recognition benchmarks UCF101 and HMDB51, where it surpasses the state-of-the-arts consistently, but requiring less memory and computation to achieve high performance.**

***Index Terms*—Action recognition, Two-stream networks, Principal Backpropagation Networks, temporal pooling strategy.**

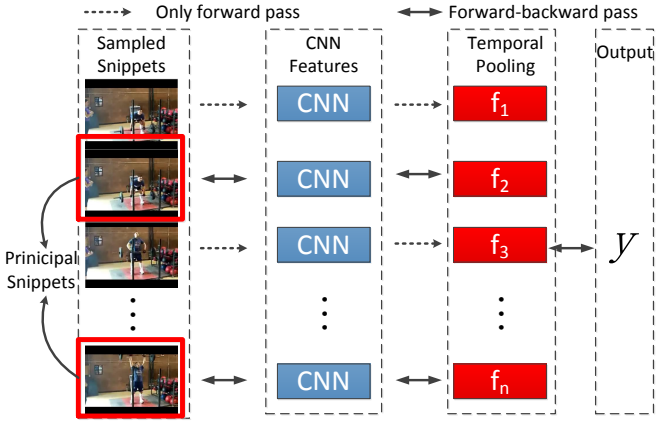


Fig. 1. We propose the Principal Backpropagation Networks (PBNets) for action recognition with long temporal awareness. PBNets perform the forward pass for all sampled snippets but conduct a complete forward-backward pass for only the selected snippets.

I. Introduction

Action recognition from videos is a highly active line of research due to its broad applications (*e.g.*, video surveillance [1] and human behavior analysis [2]). In the past five years or so, deep architectures and in particular Convolutional Neural Networks (CNNs) have matured at a meteoric speed when image classification is taken into account. However, the same cannot be said when *spatiotemporal data and in particular the problem of action recognition* is considered. This, by no means, implying that "no progress has been made", but merely reflecting the difficulty associated with analyzing spatiotemporal data. In fact, applying deep solutions for action recognition demands a particular network design for modeling the motion patterns.

State-of-the-art methods for action recognition analyze spatiotemporal video data through two-stream networks [3], [4], [5], [6], [7]. The two-stream network [3] is a clever design to benefit from spatiotemporal information in recognizing actions. Noting that learning appearance and motion clues jointly could become overwhelming, Simonyan and Zisserman propose to simplify the learning procedure [3] via two separate networks. In two-stream models, the spatial network targets appearance of actions while the temporal network is explicitly fed with optical flow data to learn the motion clues.

The original structure of the two-stream networks has a short temporal span (10 frames to be specific). This in turn makes the network vulnerable for tasks with long-term dependencies. Several studies suggest pooling the features along or between the streams to circumvent this shortcoming [4], [8], [6]. Equipping the two-stream network with a recurrent network is also considered in the literature to address the same problem [4], [9]. Very recently, Wang *et al*. propose Temporal-Segment-Networks (TSNs) where a sparse sampling method is employed to train the two-stream networks [7]. To be brief, a video is first partitioned into several segments. From each segment, a snippet, specifically an RGB frame for the spatial stream and an optical-flow stack for the temporal stream, is chosen randomly and used consequently to train the two-stream networks

TSN is proved to be exceptionally successful in training two-stream networks, leading to state-of-the-art performances [7]. This can be attributed to the fact that the snippets extracted from the video segments are aggregated together,

∗ denote equal contributions for this paper.

Wenbing Huang, Lin Ma, and Wei Liu are with Tencent AI Lab; Lijie Fan is with Computer Science and Artificial Intelligence Lab, Massachusetts Institute of Technology; Mehrtash Harandi is in the department of Electrical and Computer Systems Engineering at Monash University; Chuang Gan is with MIT-Watson Lab; Huaping Liu is with Department of Computer Science and Technology, Tsinghua University.

Contacting Email: ganchuang1990@gmail.com.

thus leading to a better representation of temporal evolution over the appearance and optical-flow for the entire video. In practice, the TSN splits each video into three segments. This raises a natural question, is it enough to have only three segments to model the evolution of appearance/optical-flow? Training with three segments may sound like a design-choice in TSNs. On one hand, complex actions such as playing basketball, usually contains multiple motion components (*i.e*., bouncing, holding and shooting), and it demands more snippets to cover the entire story. However, as we will shortly show, having more segments in TSNs will not only incur heavy computations during backpropagation but also increase the memory footprint significantly. On the other hand, recognizing actions based on videos is indeed a weakly supervised learning problem where only the video-level label is available. Intuitively, not all of the snippets within a video are related to the action itself. For better classification, we need to find out the key snippets and then perform aggregation over them.

In this paper, we introduce the Principal Backpropagation Networks (PBNets), a variant of the two-stream networks [3] with long temporal awareness. Compared to previous studies, our solution benefits from denser temporal sampling which in turn leads to learning longer-temporal information. This however comes at a very economical cost in terms of back-propagation and memory requirement. Our contributions in this work can be summarized as follows:

- We design a *Watch and Choose* mechanism to train the two-stream networks efficiently. In the *Watch* phase, we perform the forward pass for all sampled snippets in CNNs to obtain the full temporal information. In the *Choose* phase, we select a representative subset of snippets to perform a complete forward-backward computations for each input video.
- We propose two selection rules, namely the Max-rule(§ III-B1) and the KL-rule(§ III-B2). The Max-rule attempts to find the "worse" snippets to bound the optimization loss, while the KL-rule aims at choosing the "nearest" ones to the full video under the Kullback-Leibler divergence. We theoretically prove that both rules are guaranteed to decrease the loss of all sampled snippets.
- The proposed method provides superior results than other counterparts on two popular action recognition benchmarks, *i.e*. 95.4% on UCF101 [10] and 72.5% on HMDB51 [11].

II. Related Work

Traditionally, actions were described using 3D primitive shapes, silhouettes and motion signatures to name a few. Later on, methods based on aggregating hand-crafted local features (*e.g*., motion boundary histograms) were shown to deliver better performances [12], [13], [14], [15], [16]. See [17] and references therein for a recent review.

The current trend, following the success of hand-crafted local features, is to learn video descriptors directly from data using the concept of deep learning [2], [8], [4], [17]. Naturally, a deep net should capture both appearance and motion clues towards recognizing actions. This is because some actions can be merely identified using their appearances (*e.g*., playing flute) while for fine-grained actions motion information is required.

To address this need, two strategies have shaped the literature lately. Some studies opt for learning the appearance and motion information jointly by extending 2D convolutional layers to their 3D counterparts [2]. On the other hand, the idea of two-stream networks [3] demonstrates that separating the learning procedure (at least partially) is beneficial. This is achieved by training two networks, one using the appearance (*i.e*., RGB) data and one using motion information (*i.e*., optical flow). In defense of the two-stream networks, we note that modeling motion information through 3D convolutional filters not only is computationally expensive but also attains limited gains empirically [4]. As such and inline with our work, here we chiefly focus on recent developments in two-stream solutions.

The original two-stream network, proposed by Simonyan and Zisserman, uses two separate networks on RGB and optical flow data along a layer that combines the softmax scores of the streams into final decisions [3]. Questioning the short-term span of two-stream networks, Ng *et al*. proposed various ways of pooling across each stream to increase the temporal-awareness of the networks [4]. Replacing feature pooling with recurrent networks was also studied in [4], though empirically pooling showed to be superior. A relevant study is the work of Wang *et al*. [7] where temporal segment networks are introduced to again circumvent the drawback of short temporal span. The induced randomness and sparsity along the explicit use of segments are proved to be exceptionally successful in training two-stream networks, leading to state-of-the-art performances. The authors in [18] also proposed to mine key frames for better classification accuracies. However, the method only processes one snippet for one video and never performs aggregation over multiple snippets to consider long-range temporal dependency. There are also a line of researches focusing on making use of the correlations between two streams to boost the recognition performance. For instance, the work by [5] proposed to use residual connections, the spatial-temporal pooling between streams was developed in [6], the Spatiotemporal Multiplier Network [19] studied the multiplicative gating functions and the spatiotemporal pyramid network by [20] proposed to fuse the spatial and temporal features in a pyramid structure. A more recent work is the TVNet proposed by [21] that further facilitates the training of the two-stream models. By replacing the TV-L1 method with a trainable deep neural network, TVNet enables well initialization and end-to-end training of optical flows.

Before concluding this part, we acknowledge the work of Shrivastava *et al*. where a similar idea to ours was proposed, albeit for the task of object detection [22]. We note that for object detection, the proposals are considered as different data points and their losses are independent. On the contrary, for the task of video recognition, the sampled snippets together represent the whole video, and the selection of the principal snippets depends on their correlations with the global video information.

III. Our Approach

We start this section by briefly reviewing the temporal pooling strategy developed for the two-stream networks and follow it up by presenting our proposed model, *i.e.* the PBNets.

A. Temporal Pooling Strategy

State-of-the-art two-stream networks benefit from long-range temporal structures by temporal pooling (*e.g.*, [4], [7]). In short, snippets sampled from the video are processed by each stream and the results are combined by pooling layers. A snippet corresponds to an individual frame for the spatial stream and a stack of optical-flow frames for the temporal stream. To establish the ideas behind the PBNets, we need to detail out the forward and backward passes in two-stream networks.

Forward pass. Let $S_1, S_2, \cdots, S_m$ denote m snippets sampled from a video. All the snippets are fed into the CNN and their output features are computed as,

$$\boldsymbol{f}_i = \text{Forward}(S_i; \boldsymbol{W}), \quad i \in \{1, \cdots, m\}\ , \tag{1}$$

where $\boldsymbol{W}$ is the parameters of the CNN. The features associated to the snippets are pooled as,

$$\boldsymbol{z} = \text{Pooling}(\boldsymbol{f}_1, \boldsymbol{f}_2, \cdots, \boldsymbol{f}_m). \tag{2}$$

The pooling function in Eq. (2) can be max, average or even a weighted linear form. Following [7], we use average pooling in our work. The result of pooling is sent to a softmax layer to compute class likelihoods as,

$$\boldsymbol{p} = \text{SoftMax}(\boldsymbol{z}). \tag{3}$$

With the likelihoods at our disposal, we can obtain the cross-entropy loss

$$L(\boldsymbol{p}, y) = \text{CrossEntropy}(\boldsymbol{p}, y), \tag{4}$$

where y is the label for the input video.

Backward pass. In the backward pass, the gradients of the loss with respect to the features $\boldsymbol{f}_1, \boldsymbol{f}_2, \cdots, \boldsymbol{f}_m$ are first obtained. Then the CNN parameters are updated by backpropagating the gradients through the network as

$$\frac{\partial L(\boldsymbol{p}, y)}{\partial \boldsymbol{W}} = \sum_{i=1}^{m} \frac{\partial L(\boldsymbol{p}, y)}{\partial \boldsymbol{f}_i} \frac{\partial \boldsymbol{f}_i}{\partial \boldsymbol{W}}. \tag{5}$$

Obviously, the above temporal pooling strategy will become overwhelming if one wants to discover richer temporal patterns by simply processing more snippets, *i.e.*, increasing the value of m for each video. This is because both the forward pass (Eq. (1)) and the backpropagation (Eq. (5)) need to be performed m times. Furthermore, the activations of all the CNN units for all the m snippets should be stored in the memory for backward computations; thus, a larger m will inevitably cost more memory resource. In the next section, we show how our idea enables us to benefit from richer temporal information while avoiding extensive computations.

B. Principal Backpropagation Networks

The PBNet consists of two processes: the **full forward pass** and the **principal backward pass**, as illustrated in Figure 2.

For the full forward pass, as the name implies, we feed all sampled snippets to the CNN to obtain their features by Eq. (1). For the backward pass, only a small number of principle snippets are chosen for backpropagation. To be specific, we denote the selected snippets as $S_{q_1}, S_{q_2}, \cdots, S_{q_n}$, where $n < m$ and the indexes $q_i \in \{1, 2, \cdots, m\}$. We first perform the pooling operation on these snippets as

$$\boldsymbol{z}^* = \text{Pooling}(\boldsymbol{f}_{q_1}, \boldsymbol{f}_{q_2}, \cdots, \boldsymbol{f}_{q_n}). \tag{6}$$

Then we obtain the cross-entropy loss as $L(\boldsymbol{p}^*, y)$ by using Eq. (3) and Eq. (4). To compute the gradients with respect to the CNN parameters, we carry out the backpropagation as

$$\frac{\partial L(\boldsymbol{p}^*, y)}{\partial \boldsymbol{W}} = \sum_{i=1}^{n} \frac{\partial L(\boldsymbol{p}^*, y)}{\partial \boldsymbol{f}_{q_i}} \frac{\partial \boldsymbol{f}_{q_i}}{\partial \boldsymbol{W}}. \tag{7}$$

Compared to the full backpropagation in Eq. (5), the complexity here has been decreased from $O(m)$ to $O(n)$. With a small number of snippets, ensuring that the performance is similar to that of the full backpropagation is critical. Therefore, it naturally arises a new question *how the principal snippets should be chosen?*

Intuitively, the selected snippets should have some underling correlations with the whole set. If we train the CNN with the selected inputs, we must make sure that the loss of the whole snippets is also minimized to some extent. Moreover, an ideal selection scheme should efficiently filer the redundant or irrelevant information for better classification. To achieve this, we propose two selection rules: the **Max-rule** and the **KL-rule**.

Before presenting the main results, we define a few notations. Let $\{\mathbb{F}_1, \mathbb{F}_2, \cdots, \mathbb{F}_g\}$ be a partition of $\mathbb{F} = \{\boldsymbol{f}_1, \boldsymbol{f}_2, \cdots, \boldsymbol{f}_n\}$, *i.e.*, $\mathbb{F} = \bigcup_{i=1}^{g} \mathbb{F}_i$ and $\mathbb{F}_i \bigcap \mathbb{F}_j = \emptyset$, for $i \neq j$. We denote the elements of $\mathbb{F}_i$ by $\{\boldsymbol{f}_{q_{i,1}}, \boldsymbol{f}_{q_{i,2}}, \cdots, \boldsymbol{f}_{q_{i,n}}\}$. We note that the partitions (*i.e.*, $\mathbb{F}_i$) are equal-size (this can be easily achieved by sampling enough snippets in practice).

1) Max-Rule: Let $\boldsymbol{z}$ and $\boldsymbol{z}_j$ denote the pooling results of $\mathbb{F}$ and its j-th partition (*i.e.*, $\mathbb{F}_j$), respectively; $\boldsymbol{p}$ and $\boldsymbol{p}_j$ are the associated softmax outputs. The following theorem establishes the relationships between their cross-entropy losses.

Theorem 1. *If the snippets are pooled using average pooling (see Eq. (2))*[1]*, then we have*

$$L(\boldsymbol{p}, y) \leq \frac{1}{g} \sum_{j}^{g} L(\boldsymbol{p}_j, y) \leq \max_{1 \leq j \leq g} L(\boldsymbol{p}_j, y), \tag{8}$$

where $L(\cdot, \cdot)$ is given by Eq. (4).

Proof. See the supplementary material. □

For simplicity, hereafter, we name the loss generated by the whole snippets (*i.e.* $L(\boldsymbol{p}, y)$) and its j-th subset (*i.e.* $L(\boldsymbol{p}_j, y)$) as the full and partial losses, respectively. Theorem 1 states that

[1] Although we only consider the average pooling in Theorem 1, the result is also applicable to the weighted pooling, as presented in the supplemenary material.

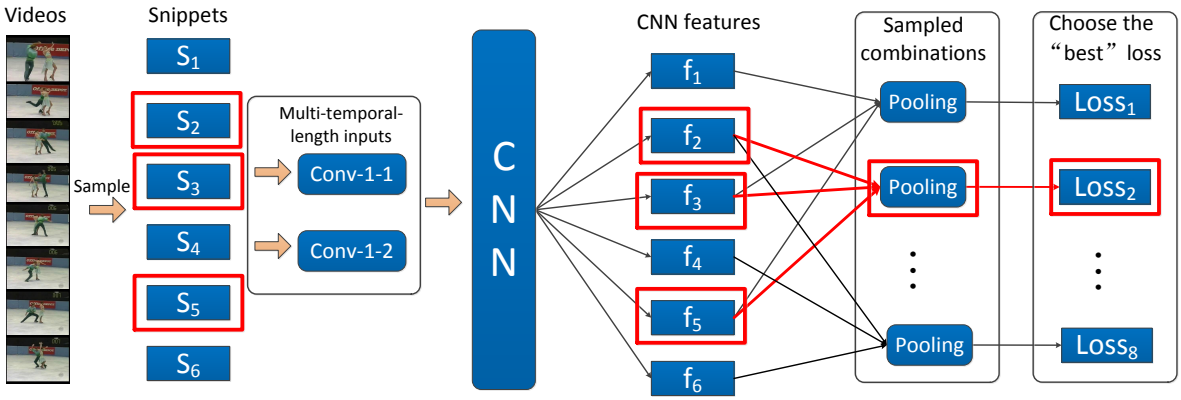


Fig. 2. Illustration of the PBNet for the temporal stream. We sample the snippets with multiple temporal lengths. We perform the forward computations for all snippets and only choose a small number (denoted with red squares) for the complete forward-backward pass via the methods presented in § III-B.

the full loss is bounded by the maximum partial loss. As such, by minimizing the maximum partial loss, we minimize an upper-bound of the full loss. Our empirical evaluations show that this min-max optimization strategy leads to a significant improvement over the previous practice where minimizing the full loss was deemed. One can also relate the Max-rule to the concept of hard example mining (*e.g.* [22] for object detection), where the learner is trained by focusing on the most difficult cases during training.

2) KL-Rule:

Theorem 2. *The full and partial losses are related as*

$$L(\boldsymbol{p}, y) = L(\boldsymbol{p}_j, y) - D_{\mathrm{KL}}(\boldsymbol{p} \| \boldsymbol{p}_j) - E_{\boldsymbol{p}}(\overline{\boldsymbol{z}_j} - \overline{\boldsymbol{z}}), \tag{9}$$

where $D_{KL}(\cdot,\cdot)$ computes the Kullback-Leibler (KL) divergence, $E_{\boldsymbol{p}}(x)$ is the expected value of x over the distribution $\boldsymbol{p}$, $\overline{\boldsymbol{z}} = \boldsymbol{z} - z^{(y)}$ and $\overline{\boldsymbol{z}_j} = \boldsymbol{z}_j - z_j^{(y)}$ with $z^{(y)}$ and $z_j^{(y)}$ being the elements of the y-th class in $\boldsymbol{z}$ and $\boldsymbol{z}_j$, respectively.

Proof. See the supplementary material. □

Eq. (9) suggests that by choosing the subset that minimizes the loss-difference $L_{diff} = |D_{\mathrm{KL}}(\boldsymbol{p}\|\boldsymbol{p}_j) + E_{\boldsymbol{p}}(\overline{\boldsymbol{z}_j} - \overline{\boldsymbol{z}})|$, one can hope that the partial loss $L(\boldsymbol{p}_j, y)$ gets closer to the full loss $L(\boldsymbol{p}, y)$. In other words, minimizing $L(\boldsymbol{p}_j, y)$ should result in decreasing $L(\boldsymbol{p}, y)$.

To make use of Theorem 2, we need to consider a subtle point. The term L_{diff} only measures the differences with respect to class y (see the supplementary material). Thus, selecting the snippets based on L_{diff} may suffer from bias as it does not take the predictions of other classes into account. As such, we suggest selecting the principal snippets by minimizing the KL divergence $D_{\mathrm{KL}}(\boldsymbol{z}\|\boldsymbol{z}_j)$ instead. In other words, the subset that has the closest distribution to the whole should be chosen.

For further explanations of the Max-rule and KL-rule, we have the following two remarks.

Remark 3. *From an optimization perspective, the idea of replacing the full loss with the partial loss is similar to constructing a surrogate loss for training. Specifically, the Max-rule attempts to find the upperbound surrogate, while the KL-rule aims at choosing the nearest surrogate to the original loss.*

Remark 4. *An interesting property of the KL-rule is its robustness to noisy snippets. To put this into perspective, we note that the full loss is more robust to outliers/noisy snippets as a result of the averaging operation. Since the KL-rule chooses the most similar subset, one expects that the noisy snippets are avoided.*

Note that in addition to the Max or KL-rules, one can randomly select the principal snippets without any heuristic consideration for back-propagation. However, this random rule can actually degenerate to the full back-propagation case discussed in § III-A by simply setting $m = n$ in Eq. (7). Our experiments will thoroughly compare the proposed selection rules with the full back-propagation scheme.

3) Selection from sampled partitions: Selecting the principal subset according to the max or KL-rule relies on the way the whole set is partitioned. An enhanced performance is expected if we select the snippets across different partitions. However, this leads to the complexity of $O(m \times (m-1) \cdots \times (m-n+1)/n!)$ for the comparisons over all possible partitions, which will cause a heavy computational burden for a large m. From a practical view, considering all possible combinations of the snippets is not necessary as most snippets within a video are similar to each other. Therefore, we propose a trade-off method by stochastically sampling a certain number of combinations. The principal subset is then chosen among those sampled combinations. In our experiments, we set the number of the sampled combinations to be 100 and find it sufficient to obtain the desired performance for the datasets we applied.

4) Multi-Temporal-Length Snippets: For the flow stream, each snippet captures the local motion component of an action. In this section, we propose to process the snippets with variable temporal lengths (*i.e.*, the number of the optical frames). The motivation behind this is two-fold. First, local motions naturally have different lengths. Even for the very same action, the duration of a local motion is influenced

by environmental factors. Second, inclusion of snippets with different temporal lengths increases the diversity among the training samples, which in turn helps training deep networks with large number of parameters.

In the original two stream network [3], the authors fed the optical frames as the channels of the first convolutional layer. To have multiple temporal lengths, we resort to convolutions prior to the first layer (see Fig. 2). For example, in our experiments, we have made use of two convolutional layers, namely Conv-1-1 with 5 channels to process 5-frame inputs along Conv-1-2 with 10 channels to handle longer inputs. Since the output size of Conv-1-1 and Conv-1-2 is equal (by design), their outputs can be concatenated and fed into the follow-up layers leading to having the same flow stream afterwards.

IV. Practical Considerations

In [7], the inception model with batch-normalization (BN-Inception) [23] is claimed to be superior to some other choices like the VGG-16 net [24]. To make fair comparisons, we use the BN-Inception net as the CNN for the two-stream model. Pre-training is crucial for the network initialization when the training samples are not sufficient. Therefore, for the spatial net, we use the BN-Inception net trained on the ImageNet data [25]. For the temporal net, we resorted to the cross-modality skill introduced in [26] for initialization. To prevent overfitting, we also carry out the corner cropping and scale jittering.

A. Training the Network

For each video, we generate m snippets by first dividing the whole video into m segments and then randomly sampling one snippet from each segment. During training the temporal net, we consider two types of snippets, namely snippets with temporal length of 5 and 10. In doing so, half of a mini-batch is dedicated to snippets with the length of 5 and the other half to the ones with the length of 10.

We use the mini-batch Stochastic Gradient Descent (SGD) algorithm to train the networks. Our training process is illustrated in Figure 3. Each iteration comprises two phases: the full forward pass and the principal forward-backward pass. In particular, the full forward pass processes the m snippets for each input video, while the principal forward-backward pass only handles the selected n snippets. In the forward pass, it is not necessary to propagate the whole batch of videos at one time since we do not need to have all hidden activations any more. To save memory, we divide the original batch-size (denoted as B here) by m/n. The new batch-size becomes $b = Bn/m$ and is much smaller than B (the number of snippets is Bn). The full forward pass will process the new batch-size m/n times. During each forward pass, we choose n snippets for each video according to the method in § III-B and store them in the memory. When all forward passes are done, the total number of the selected snippets is Bn (which is the same as the number of snippets for each forward pass). We perform a complete forward-backward computation for the selected snippets. Our implementation enables the efficient training in terms of memory cost and running time.

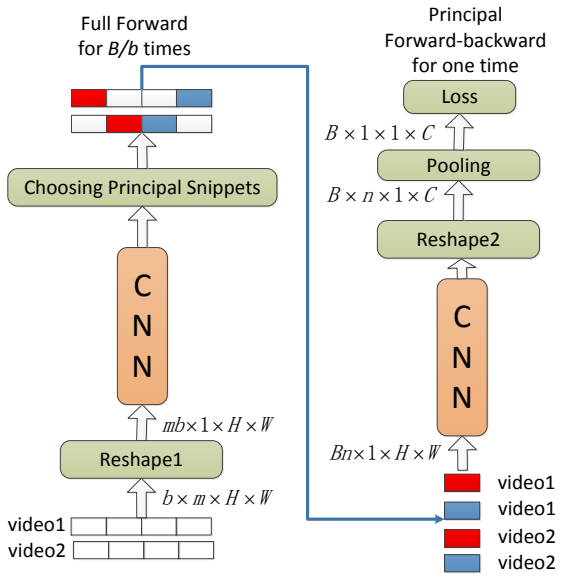


Fig. 3. A visualization of the training procedure for PBNets. Here, H and W are the height and width of each frame, respectively; C is the number of the classes. We sample m snippets from each video and picks n for backpropagation. We perform the full forward pass for $b = \frac{Bn}{m}$ videos per time, where B is the min-batch size. At each forward pass, we select the principal snippets by loss comparisons and store them in the memory. Once the forward computations of the B videos are finished, we begin the complete forward-backward pass for the selected snippets. As the CNN is shared between the forward and backward passes, we apply the "reshape layer" (defined in caffe [27]) to reshape the inputs to have the same shape.

B. Testing Scheme

For the spatial stream and following [3], snippets are extracted from the center and four corners of a video. We sample 25 snippets from each location (*i.e.*, center and corners), followed by flipping them horizontally to enlarge the testing samples. All the sampled snippets (250 in total) are fed to CNN and their outputs are averaged for prediction.

For the temporal net, due to its multi-temporal-length, we sample 25 snippets of length 5 and another 25 snippets with length 10, again from the center and the corners, leading to a total number of 500 snippets after flipping. To fuse the streams, we take a weighted combination of the spatial and temporal networks as done in [7]. We average the prediction scores before the softmax layer as this approach achieves better results than averaging the softmax scores [5]. We follow the suggestion by [7], and set the weights of the spatial and temporal streams to be 2 and 3, respectively.

V. Evaluations

We start this section by introducing benchmark datasets and implementation details used in our experiments. Later, we will compare and contrast the proposed selection rules (see § III-B) and follow it up by studying the performance of the proposed multi-temporal-length technique. We will also compare PBNets against the full back-propagation (full-bp) networks. Finally, we will contrast our models against the

state-of-the-arts. Hereafter, we show our model by PBNet-m-n, meaning n out of m snippets were chosen from each video during training. We also denote the full-bp models by Full-m to show m snippets were sampled and backpropagated.

A. Datasets and Implementation Details

Our experiments are conducted on two popular action recognition datasets, namely the UCF101 [10] and the HMDB51 [11] datasets. The UCF101 dataset contains 13320 videos of 101 action classes. The HMDB51 dataset consists of 6766 videos from 51 action categories. The videos in the HMDB51 dataset are collected from various sources, such as movies and web videos. For both datasets, we follow the provided evaluation protocols and report the average accuracy over the three training/testing splits.

To train the models, we set the mini-batch size to 128 and the momentum to 0.9. For the spatial stream, the learning rate was initialized to 0.001 and decreased by a factor of 10 every 1500 iterations. The spatial stream was trained for 5000 iterations on the UCF101 dataset and 3000 iterations on the HMDB51 dataset. For the temporal stream, the learning rate was initialized to 0.005. The maximum number of iterations for the UCF101 and the HMDB51 datasets was chosen as 18000 and 7000, respectively. Again we decreased the learning rates by a factor of 10 after the 10000th and 16000th iterations for the UCF101 experiment, and after 4000th and 6000th iterations for the HMDB51 case.

We computed the optical flow [28] before training and stored the flow fields as JPEG images by linear compression. We implemented our networks using Caffe [27]. Our experiments were performed on 8 Tesla M40 GPUs.

B. Which Selection Rule?

We propose two selection rules, *i.e.*, the Max-rule and the KL-rule, to choose the principal snippets from each video in § III-B. To better understand their behaviors, we performed an experiment using PBNet-6-3 on the first split of the UCF101 dataset. Here, we denote the *full loss* to be the loss on all sampled snippets, and the *selected partial loss* to be the loss on the selected snippets by the Max-rule or KL-rule. These two losses are recoded every 20 iterations during training.

Figure 4 and Figure 5 display the training losses for the temporal and spatial streams, respectively. We first observe that the full losses produced by the Max-rule is always smaller than the selected partial losses for both streams, which is consistent to our statement in Theorem 1.

For the temporal stream (Figure 4), since the Max-rule chooses the hard examples for training, it naturally generates higher selected partial losses compared to the KL-rule. However, the full loss (which we really care about) converges faster for the Max-rule, suggesting that working with the hard samples can lead to speed-ups for the optimization on overall examples.

For the spatial stream (Figure 5), both full and selected partial losses fluctuate more severely for the Max-rule compared to the KL-rule. The snippets of the spatial stream corresponds to individual RGB frames. However, a single RGB frame can reflect incomplete or even erroneous information about the video, thus selecting the RGB frames with Max-rule can sometimes confuse the classifier. In contrast, the KL-rule does not suffer from such an issue as it always attempts to choose the RGB frames that best represent the whole video.

We summarize the classification accuracies of the Max-rule and the KL-rule for the flow and RGB modalities in Table I. The Max-rule achieves better result than the KL-rule for the temporal net, while the KL-rule outperforms the Max-rule for the spatial net. We also combine the two stream nets under 4 different possible fusion schemes and report the results in Table II. Not surprising, the fusion of RGB with the KL-rule and flow with the Max-rule produces the best accuracy. Based on this experiment, hereafter, we apply the Max-rule in the temporal stream and the KL-rule in the spatial stream.

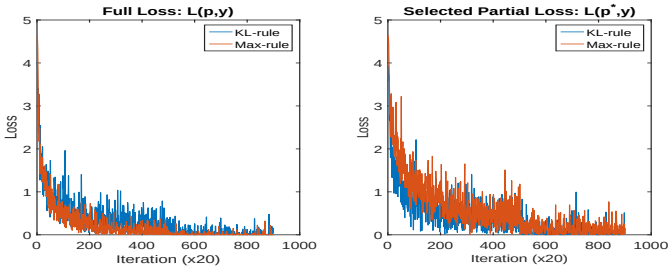


Fig. 4. The training losses for the temporal stream of PBNet-6-3 on the UCF101 dataset (split 1).

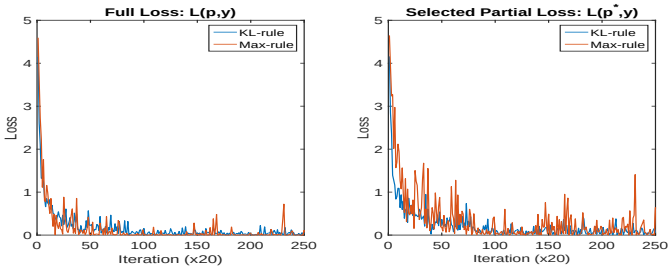


Fig. 5. The training losses for the spatial stream of PBNet-6-3 on the UCF101 dataset (split 1).

TABLE I
THE CLASSIFICATION ACCURACY OBTAINED BY TWO SELECTION RULES ON THE UCF101 DATASET (SPLIT 1).

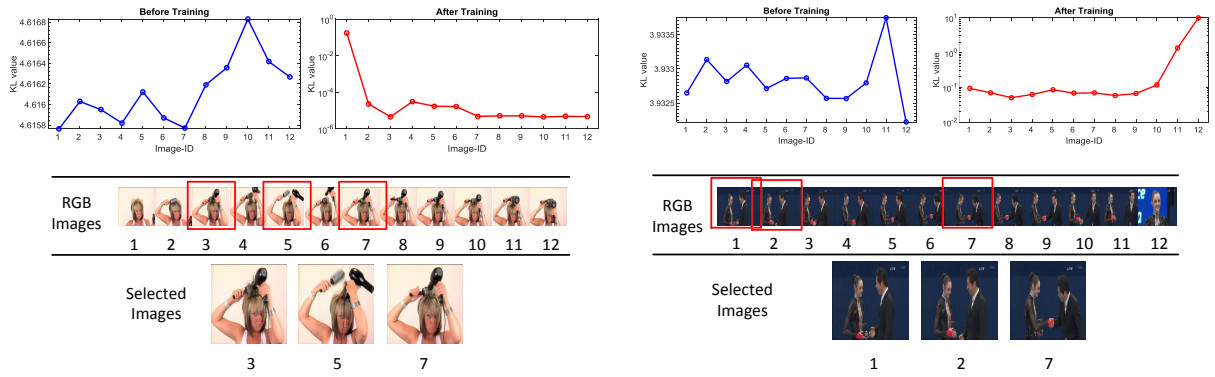
Modality	Max-rule	KL-rule
Flow	**88.1%**	87.5%
RGB	85.3%	**86.0%**

TABLE II
THE CLASSIFICATION ACCURACY OBTAINED BY DIFFERENT FUSION SCHEMES ON UCF101 (SPLIT 1).

Fusion	RGB + Max-rule	RGB + KL-rule
Flow + Max-rule	93.7%	**94.4%**
Flow + KL-rule	93.5%	93.7%

C. Illustration on the KL-rule

Here, we provide more experimental results to illustrate the benefits of the proposed KL-rule (§ 3.2.2). For this purpose, we conducted two additional experiments on the UCF and HMDB



(a) UCF (b) HMDB

Fig. 6. Illustration on the testing example from the (a) UCF and (b) HMDB datasets. We sampled 12 RGB images from the testing video with equal stride and then computed the KL divergences between the class likelihoods (Eq.(3)) of each sampled image and those of the whole set. To evaluate the importance of the training by PBNets, we report the results by using the net before training and the one after training. We also report the 3 selected images whose pooled feature has a minimal KL distance to the global distribution among all possible selections out of 12 sampled images.

datasets, respectively. Particularly, we sampled 12 images with equal stride given a testing video and then computed the KL divergences between the class likelihoods (Eq. (3)) of the CNN features for each image and those for the whole set. To illustrate the importance of the training by our methods, we report the results by using the net before training and the one after training (we applied PBNet-6-3 for the training).

Figure 6 and Figure 8 display the results on UCF and HMDB, respectively. It is observed that the initial nets generate large KL distances which in turn demonstrates the incorrect relevancy between each image and the global video. After training, the KL values decrease significantly. More importantly, the networks are able to detect the outliers which have a relatively larger KL values (*i.e.*, the 1-st image in Figure 6 and the 12-th image in Figure 8). We also report the 3 selected images with minimum KL distance after pooling (note that they are not necessarily corresponded to the 3 minimal KL values). It is interesting to see that the selected images reflect the different stages of the action and hence summarizing the entire story.

D. Evaluation on Multi-Temporal-Length Networks

In § III, we explore the method to process multi-temporal-length snippets for the temporal net. To evaluate the improvement by this extension, we contrast the performance of the PBNets operating on the multi-length inputs (*i.e.*, 5-length and 10-length snippets) to the networks operating only on single-length inputs. Table III summarizes the classification accuracies by PBNet-6-3 on the first split of the UCF101 dataset. It is shown that considering multi-temporal-length inputs is able to slightly boost the classification performance.

TABLE III
EVALUATIONS OF THE MULTI-TEMPORAL-LENGTH INPUTS ON THE UCF101 DATASET (SPLIT 1).

Method	5-length	10-length	Multi-length
PBNet-6-3	87.8%	87.5%%	**88.1%**

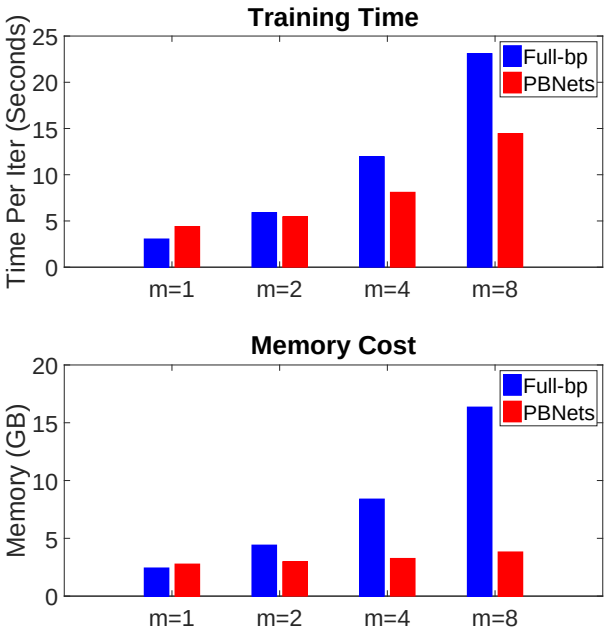


Fig. 7. Efficiency comparisons between Full-m and PBNet-m-1 with m varying from 1 to 8. For fair comparisons, the experiments are carried out on one GPU. The iter_size for SGD is set to be 8.

TABLE V
EFFICIENCY COMPARISONS BETWEEN FULL-m AND PBNET-6-3. FOR FAIR COMPARISONS, THE EXPERIMENTS ARE CARRIED OUT ON ONE GPU. THE ITER_SIZE FOR SGD IS SET TO BE 8.

Modality	Full-3	Full-4	Full-6	PBNet-6-3
TimePerIter	10.37s	12.12s	13.88s	11.30s
Memory	7.63GB	9.66GB	13.86GB	8.14GB

E. Comparisons with Full-bp Models

In this experiment, we contrast the proposed PBNets against the full-bp models. We increase the number of the sampled

TABLE IV
CLASSIFICATION PERFORMANCE COMPARISON BETWEEN PBNETS AND FULL-BP MODELS ON THE UCF101 DATASET (SPLIT 1).

Modality	Full-1	Full-2	Full-3	Full-4	Full-5	Full-6	PBNet-6-3	PBNet-8-4
Flow	85.3%	86.9%	87.4%	87.5%	87.5%	87.4%	88.1%	88.5%
RGB	85.0%	85.2%	85.4%	85.2%	85.5%	85.5%	86.0%	86.2%

snippets (*i.e.*, m) from 1 to 6 in full-bp models, and report the classification accuracies on the first split of the UCF101 dataset for both streams in Table IV. We first note that employing more snippets can almost increase the performance, although the improvement becomes small when $m > 3$.

In addition to the full-bp models, we also summarize the results of PBNet-6-3 and PBNet-8-4 in Table IV. PBNet-6-3 and PBNet-8-4 outperform Full-3 and Full-4, respectively, implying that involving longer temporal information into the training of PBNets can promote the accuracies consistently. Our model PBNet-6-3 even achieves better accuracies than the Full-6 model. By selecting the principal snippets, our PBNet-6-3 not only reduces the back-propagation time but also removes the snippets that are not so related to the task (e.g. the background items), which potentially enables our model to outperform the Full-6 model. Overall, PBNet-8-4 outperforms all full-bp models. As presented in § III-B2, the model Full-3 is equivalent to the case where 3 principal snippets are selected by random. Our PBNet-6-3 outperforms Full-3 thus verifying the benefit of the proposed selection rules to the naive random selection method.

We compare the training times of full-bp models and PBNets in Figure 7. For this experiment, we increased the value of m from 1 to 8 for full-bp models and PBNets, and fix the number of selected snippets to 1 for PBNets. For fair comparisons, we set all the caffe parameters (*e.g.*, iter_size) similarly in all models, and carried out the experiments on one GPU. Figure 7 shows that PBNet-1-1 is slightly slower than Full-1, even though PBNet-1-1 is actually equivalent to Full-1. This is because in our implementation, PBNet-1-1 needs an additional forward pass in the forward-backward stage. Increasing m incurs more computations for full-bp models; while for PBNets, the selected backpropagation scheme remarkably reduces the running times (even for a large m).

We also plot the memory footprints in Figure 7. The full-bp models require more memory when the value of m is increased. On the contrary, our PBNets almost have the same memory footprint even when m is large. As such, training with limited memory is possible with our model.

In addition, we compare the training time and memory cost of PBNet-6-3 with the Full-bp models in Table V. Clearly, our PBNet-6-3 costs slightly more time and memory than Full-3, but is more efficient than other Full-bps (especially for Full-6).

F. Comparisons with The State-of-The-Arts

We compare the classification accuracy of PBNets with the state-of-the-art approaches over all three splits of the UCF101 and the HMDB51 dataset in Table VI. The improvements achieved by PBNets are quite substantial, at least being 6.5% on the UCF101 and 9.5% on the HMDB51 compared to the original two-stream method [3]. Such significant gains are achieved as a result of employing better models (*i.e.*, BN-Inception net) and also considering long-range temporal patterns. Our solution comfortably outperforms the Spatial-Temporal-Residue-Network (ST-ResNet) which benefits from a very deep network (*i.e.*, 50-layer ResNet) along cross-stream residual connections [5]. This clearly demonstrates the importance of our proposal in training two-stream models. The TSN model [7] also benefits from long-range temporal structure (using temporal pooling) and BN-Inception net. Nevertheless, extending TSN to work with longer temporal information seems to be difficult due to its high computation load and memory requirements. Furthermore, we note that our solution is superior to TSN on both datasets.

As shown in [29], [30], [31], [5], combining CNN models with trajectory-based hand-crafted IDT features [32] can improve the final performances. Hence, we performed a late fusion of hand-crafted IDT features with our approach. For this purpose, we averaged the L2-normalized SVM scores of FV-encoded IDT features (*i.e.*, HOG, HOF and MBH) with the L2-normalized video predictions (before the loss layer) of our PBNets. Table VI summarizes the results and indicates that there is still room for improvement. Overall, our 95.4% on the UCF101 and 72.5% on the HMDB51 are superior than other counterparts on these two action recognition benchmarks.

TABLE VI
MEAN CLASSIFICATION ACCURACY OF THE STATE-OF-THE-ARTS ON HMDB51 AND UCF101.

Method	UCF101	HMDB51
Two-Stream Model [3]	88.0%	59.4%
Two-Stream+LSTM [4]	88.6%	-
Two-Stream (VGG16) [33]	91.4%	58.5%
Two-Stream Fusion [6]	92.5%	65.4%
KVMF [18]	93.1%	63.3%
TSN [7]	94.0%	68.5%
ST-ResNet [5]	93.4%	66.4%
TDD + IDT [29]	91.5%	65.9%
Dynamic Image Networks + IDT [31]	89.1%	65.2%
Two-Stream Fusion + IDT [6]	93.5%	69.2%
ST-ResNet + IDT [5]	94.6%	70.3%
Full-6	93.7%	68.7%
Full-6 + IDT	94.6%	70.8%
PBNet-6-3	**94.5%**	**68.9%**
PBNet-8-4	**94.9%**	**69.3%**
PBNet-6-3 + IDT	**95.2%**	**72.2%**
PBNet-8-4 + IDT	**95.4%**	**72.5%**

G. Evaluations on Untrimmed Videos

To further evaluate the effectiveness of our method, we also conduct experiments of untrimmed videos on the THUMOS14 [34]. This dataset contains four subsets: training set,

validation set, testing set and background set. We use the training set (*i.e*. all videos in UCF101) and the validation set for training, and the testing set containing 1575 videos for testing. The training settings are the same as that of UCF101 excluding that we enlarge the numbers of the training iterations to be 25000 and 8000 for the temporal and spatial streams, respectively. Besides during training, we regard the same video with the principal and secondary labels as two different videos: one with the principal label and the other with the secondary label. We only apply the single temporal-length input (*i.e*. 5-length) for the flow stream. For the testing, we uniformly divide each video into 100 segments, and extract the center and four corners from each segment followed by horizontal flipping. We first obtain total 1000 snippets for each testing video, then average the output scores of them, and finally compute the Mean Average Precision (mAP) by the THUMOS toolkit. We report the MAPs of Full-3 and PBNet-6-3 in Table VII. Obviously, our PBNet-6-3 outperforms Full-3 remarkably on the untrimmed videos.

TABLE VII
THE CLASSIFICATION ACCURACIES ON THUMOS14.

Methods	RGB	Flow	Two-stream
Full-3	64.14%	63.07%	72.7%
PBNet-6-3	69.09%	64.34%	76.5%

VI. CONCLUSION

We proposed the Principal Backpropagation Networks (PBNets) to efficiently train the two-stream network models for video action recognition with long-range temporal awareness. Specifically, our approach performs the forward computation for all sampled snippets to obtain the global temporal dependencies, but only backpropagates a small number of snippets selected by two new strategies, namely the Max-rule and the KL-rule. We showed that with these two rules, one can minimize the pooling loss over all the snippets. In our experiments, we demonstrated that the proposed PBNet achieves the state-of-the-art performance on two popular action recognition datasets.

REFERENCES

[1] R. Poppe, "A survey on vision-based human action recognition," *Image and Vision Computing (IVC)*, vol. 28, no. 6, pp. 976–990, 2010.

[2] S. Ji, W. Xu, M. Yang, and K. Yu, "3d convolutional neural networks for human action recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 1, pp. 221–231, Jan 2013.

[3] K. Simonyan and A. Zisserman, "Two-stream convolutional networks for action recognition in videos," in *Proc. Advances in Neural Information Processing Systems (NIPS)*, 2014, pp. 568–576.

[4] J. Y.-H. Ng, M. Hausknecht, S. Vijayanarasimhan, O. Vinyals, R. Monga, and G. Toderici, "Beyond short snippets: Deep networks for video classification," in *Proc. Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015, pp. 4694–4702.

[5] C. Feichtenhofer, A. Pinz, and R. Wildes, "Spatiotemporal residual networks for video action recognition," in *Proc. Advances in Neural Information Processing Systems (NIPS)*, 2016, pp. 3468–3476.

[6] C. Feichtenhofer, A. Pinz, and A. Zisserman, "Convolutional two-stream network fusion for video action recognition," in *Proc. Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 1933–1941.

[7] 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. European Conference on Computer Vision (ECCV)*. Springer, 2016, pp. 20–36.

[8] A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar, and L. Fei-Fei, "Large-scale video classification with convolutional neural networks," in *Proc. Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2014, pp. 1725–1732.

[9] J. Donahue, L. Anne Hendricks, S. Guadarrama, M. Rohrbach, S. Venugopalan, K. Saenko, and T. Darrell, "Long-term recurrent convolutional networks for visual recognition and description," in *Proc. Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015, pp. 2625–2634.

[10] K. Soomro, A. R. Zamir, and M. Shah, "Ucf101: A dataset of 101 human actions classes from videos in the wild," *arXiv preprint arXiv:1212.0402*, 2012.

[11] H. Kuehne, H. Jhuang, E. Garrote, T. Poggio, and T. Serre, "Hmdb: a large video database for human motion recognition," in *Proc. Int. Conference on Computer Vision (ICCV)*. IEEE, 2011, pp. 2556–2563.

[12] I. Laptev, M. Marszalek, C. Schmid, and B. Rozenfeld, "Learning realistic human actions from movies," in *Proc. Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2008, pp. 1–8.

[13] Y.-G. Jiang, Q. Dai, W. Liu, X. Xue, and C.-W. Ngo, "Human action recognition in unconstrained videos by explicit motion modeling," *IEEE Transactions on Image Processing*, vol. 24, no. 11, pp. 3781–3795, 2015.

[14] Y.-G. Jiang, Q. Dai, X. Xue, W. Liu, and C.-W. Ngo, "Trajectory-based modeling of human actions with motion reference points," in *European Conference on Computer Vision*. Springer, 2012, pp. 425–438.

[15] J. Liu, C. Chen, Y. Zhu, W. Liu, and D. N. Metaxas, "Video classification via weakly supervised sequence modeling," *Computer Vision and Image Understanding*, vol. 152, pp. 79–87, 2016.

[16] H. Wang, A. Kläser, C. Schmid, and C.-L. Liu, "Dense trajectories and motion boundary descriptors for action recognition," *Int. Journal of Computer Vision (IJCV)*, vol. 103, no. 1, pp. 60–79, 2013.

[17] S. Herath, M. Harandi, and F. Porikli, "Going deeper into action recognition: A survey," *Image and Vision Computing (IVC)*, pp. –, 2017.

[18] W. Zhu, J. Hu, G. Sun, X. Cao, and Y. Qiao, "A key volume mining deep framework for action recognition," in *Proc. Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2016, pp. 1991–1999.

[19] C. Feichtenhofer, A. Pinz, and R. Wildes, "Spatiotemporal multiplier networks for video action recognition," in *Proc. Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 7445–7454.

[20] W. Yunbo, L. Mingsheng, W. Jianmin, and P. S. Yu, "Spatiotemporal pyramid network for video action recognition," in *Proc. Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[21] L. Fan, W. Huang, C. Gan, S. Ermon, B. Gong, and J. Huang, "End-to-end learning of motion representation for video understanding," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 6016–6025.

[22] A. Shrivastava, A. Gupta, and R. Girshick, "Training region-based object detectors with online hard example mining," in *Proc. Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 761–769.

[23] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proc. Int. Conference on Machine Learning (ICML)*, 2015, pp. 448–456.

[24] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[25] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *Proc. Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2009, pp. 248–255.

[26] L. Wang, Y. Xiong, Z. Wang, and Y. Qiao, "Towards good practices for very deep two-stream convnets," *arXiv preprint arXiv:1507.02159*, 2015.

[27] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," in *ACM international conference on Multimedia (ACM MM)*. ACM, 2014, pp. 675–678.

[28] T. Brox, A. Bruhn, N. Papenberg, and J. Weickert, "High accuracy optical flow estimation based on a theory for warping," in *Proc. European Conference on Computer Vision (ECCV)*. Springer, 2004, pp. 25–36.

[29] L. Wang, Y. Qiao, and X. Tang, "Action recognition with trajectory-pooled deep-convolutional descriptors," in *Proc. Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015, pp. 4305–4314.

[30] D. Tran, L. Bourdev, R. Fergus, L. Torresani, and M. Paluri, "Learning spatiotemporal features with 3d convolutional networks," in *Proc. Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015, pp. 4489–4497.

[31] H. Bilen, B. Fernando, E. Gavves, A. Vedaldi, and S. Gould, "Dynamic image networks for action recognition," in *Proc. Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 3034–3042.

[32] H. Wang and C. Schmid, "Action recognition with improved trajectories," in *Proc. Conference on Computer Vision and Pattern Recognition (CVPR)*, 2013, pp. 3551–3558.

[33] N. Ballas, L. Yao, C. Pal, and A. Courville, "Delving deeper into convolutional networks for learning video representations," *arXiv preprint arXiv:1511.06432*, 2015.

[34] Y. Jiang, J. Liu, A. R. Zamir, G. Toderici, I. Laptev, M. Shah, and R. Sukthankar, "Thumos challenge: Action recognition with a large number of classes," 2014.

APPENDIX A
PROOFS

In the paper, we first introduce Theorem 1 and then present Theorem 2. Actually, the proof of Theorem 1 is based on the result of Theorem 2. Therefore, in this supplementary material, we switch the order of presentation to facilitate readability. Let $\{\mathbb{F}_1, \mathbb{F}_2, \cdots, \mathbb{F}_g\}$ be a partition of $\mathbb{F} = \{\boldsymbol{f}_1, \boldsymbol{f}_2, \cdots, \boldsymbol{f}_m\}$, *i.e.*, $\mathbb{F} = \bigcup_{i=1}^{g} \mathbb{F}_i$ and $\mathbb{F}_i \bigcap \mathbb{F}_j = \emptyset$, for $i \neq j$. We denote the elements of $\mathbb{F}_j$ by $\{\boldsymbol{f}_{q_{j,1}}, \boldsymbol{f}_{q_{j,2}}, \cdots, \boldsymbol{f}_{q_{j,n}}\}$. We note that the partitions (*i.e.*, $\mathbb{F}_j$) are equal-size (this can be easily achieved by sampling enough snippets in practise). Let $\boldsymbol{z}$ and $\boldsymbol{z}_j$ denote the pooling results of $\mathbb{F}$ and its j-th partition (*i.e.*, $\mathbb{F}_j$), respectively; $\boldsymbol{p}$ and $\boldsymbol{p}_j$ are the associated softmax outputs.

Theorem 2. *The full and partial losses are related as*

$$L(\boldsymbol{p}, y) = L(\boldsymbol{p}_j, y) - D_{\mathrm{KL}}(\boldsymbol{p} \| \boldsymbol{p}_j) - E_{\boldsymbol{p}}(\overline{\boldsymbol{z}_j} - \overline{\boldsymbol{z}}), \tag{10}$$

where $D_{KL}(\cdot,\cdot)$ computes the Kullback-Leibler (KL) divergence, $E_{\boldsymbol{p}}(\cdot)$ is the expected value over the distribution $\boldsymbol{p}$, $\overline{\boldsymbol{z}} = \boldsymbol{z} - z^{(y)}$ and $\overline{\boldsymbol{z}_j} = \boldsymbol{z}_j - z_j^{(y)}$ with $z^{(y)}$ and $z_j^{(y)}$ being the elements of the y-th class in $\boldsymbol{z}$ and $\boldsymbol{z}_j$, respectively.

Proof. The softmax losses of $L(\boldsymbol{z}, y)$ and $L(\boldsymbol{z}_j, y)$ take the form as

$$\begin{aligned} L(\boldsymbol{p}, y) &= \log(\sum_{c=1}^{C} \exp(z^{(c)})) - z^{(y)}, & (11)\\ L(\boldsymbol{p}_j, y) &= \log(\sum_{c=1}^{C} \exp(z_j^{(c)})) - z_j^{(y)}, & (12) \end{aligned}$$

where $z^{(c)}$ and $z_j^{(c)}$ denote the c-th columns of $\boldsymbol{z}$ and $\boldsymbol{z}_j$, respectively; C is the number of the classes. Then, the loss difference is computed by

$$\begin{aligned} L_{diff} &= L(\boldsymbol{p}_j, y) - L(\boldsymbol{p}, y),\\ &= \log(\frac{\sum_{c=1}^{C} \exp(z_j^{(c)})}{\sum_{c=1}^{C} \exp(z^{(c)})}) - (z_j^{(y)} - z^{(y)}). \end{aligned} \tag{13}$$

As for the KL-divergence $D_{KL}(\boldsymbol{p} \| \boldsymbol{p}_j)$, we have

$$\begin{aligned} & D_{KL}(\boldsymbol{p} \| \boldsymbol{p}_j)\\ =& \sum_{c=1}^{C} p^{(c)} \log\left(\frac{p^{(c)}}{p_j^{(c)}}\right),\\ =& \sum_{c=1}^{C} p^{(c)} \log(\frac{\sum_{k=1}^{C} \exp(z_j^{(k)})}{\sum_{k=1}^{C} \exp(z^{(k)})} \cdot \frac{\exp(z^{(c)})}{\exp(z_j^{(c)})}),\\ =& \sum_{c=1}^{C} p^{(c)} \log(\frac{\sum_{k=1}^{C} \exp(z_j^{(k)})}{\sum_{k=1}^{C} \exp(z^{(k)})}) + \sum_{c=1}^{C} p^{(c)}(z^{(c)} - z_j^{(c)}),\\ =& \log(\frac{\sum_{k=1}^{C} \exp(z_j^{(k)})}{\sum_{k=1}^{C} \exp(z^{(k)})}) + \sum_{c=1}^{C} p^{(c)}(z^{(c)} - z_j^{(c)}), \end{aligned} \tag{14}$$

where $p^{(c)}$ and $p_j^{(c)}$ denote the c-th columns of $\boldsymbol{p}$ and $\boldsymbol{p}_j$, respectively. By comparing the difference between Eq. (13) and Eq. (14), we arrive at

$$\begin{aligned} & L_{diff} - D_{KL}(\boldsymbol{p} \| \boldsymbol{p}_j)\\ =& \sum_{c=1}^{C} p^{(c)}((z_j^{(c)} - z_j^{(y)}) - (z^{(c)} - z^{(y)}),\\ =& E_{\boldsymbol{p}}(\overline{\boldsymbol{z}_j} - \overline{\boldsymbol{z}}), \end{aligned} \tag{15}$$

thus concluding the proof. □

Theorem 1. *If the snippets are pooled using average pooling (Eq. (2)), then we have*

$$L(\boldsymbol{p}, y) \leq \frac{1}{g} \sum_{j}^{g} L(\boldsymbol{p}_j, y) \leq \max_{1 \leq j \leq g} L(\boldsymbol{p}_j, y), \tag{16}$$

where $L(\cdot,\cdot)$ is given by Eq.(4).

Proof. We only need to prove the left inequality as the right inequality follows directly from it. According to Theorem 2, we have

$$\begin{aligned} & L(\boldsymbol{p}, y)\\ =& \frac{1}{g} \sum_{j=1}^{g} L(\boldsymbol{p}, y),\\ =& \frac{1}{g} \sum_{j}^{g} L(\boldsymbol{p}_j, y) - D_{KL}(\boldsymbol{p} \| \boldsymbol{p}_j) - E_{\boldsymbol{p}}(\overline{\boldsymbol{z}_j} - \overline{\boldsymbol{z}}),\\ =& \frac{1}{g} \sum_{j}^{g} (L(\boldsymbol{p}_j, y) - D_{KL}(\boldsymbol{p} \| \boldsymbol{p}_j)) - \frac{1}{g} \sum_{j}^{g} E_{\boldsymbol{p}}(\overline{\boldsymbol{z}_j} - \overline{\boldsymbol{z}}),\\ =& \frac{1}{g} \sum_{j}^{g} (L(\boldsymbol{p}_j, y) - D_{KL}(\boldsymbol{p} \| \boldsymbol{p}_j)) - E_{\boldsymbol{p}}(\frac{1}{g} \sum_{j}^{g} \overline{\boldsymbol{z}_j} - \overline{\boldsymbol{z}}),\\ =& \frac{1}{g} \sum_{j}^{g} L(\boldsymbol{p}_j, y) - \frac{1}{g} \sum_{j}^{g} D_{KL}(\boldsymbol{p} \| \boldsymbol{p}_j). \end{aligned} \tag{17}$$

In Eq. (17), we apply the average pooling, indicating that

$$\boldsymbol{z} = \frac{1}{m} \sum_{i=1}^{m} \boldsymbol{f}_i = \frac{1}{g} \sum_{j=1}^{g} \frac{1}{n} \sum_{i=1}^{n} \boldsymbol{f}_{q_{j,i}} = \frac{1}{g} \sum_{j=1}^{g} \boldsymbol{z}_j.$$

Therefore, we have the equation $\frac{1}{g}\sum_j^g E_{\boldsymbol{p}}(\overline{\boldsymbol{z}_j} - \overline{\boldsymbol{z}}) = E_{\boldsymbol{p}}(\frac{1}{g}\sum_j^g \overline{\boldsymbol{z}_j} - \overline{\boldsymbol{z}}) = 0$. Since $D_{KL}(\boldsymbol{p}\|\boldsymbol{p}_j) \geq 0$ for any case, we arrive that $L(\boldsymbol{p}, y) \leq \frac{1}{g}\sum_j^g L(\boldsymbol{p}_j, y)$. □

For other pooling operations such as the weight and max pooling, we have the following two corollaries.

Corollary 1. *If the pooling function in Eq. (2) employs the weight pooling,* i.e., $\boldsymbol{z} = \sum_{i=1}^m w_i \boldsymbol{f}_i$, *and* $\boldsymbol{z}_j = g\sum_{i=1}^n w_{q_{j,i}} \boldsymbol{f}_{q_{j,i}}$, *for* $j = 1, \cdots, g$, *then we have the same conclusion as Theorem 1, namely,*

$$L(\boldsymbol{p}, y) \leq \frac{1}{g}\sum_j^g L(\boldsymbol{p}_j, y) \leq \max_{1\leq j\leq g} L(\boldsymbol{p}_j, y), \tag{18}$$

Proof. With the weight pooling, we still have $\frac{1}{g}\sum_j^g E_{\boldsymbol{p}}(\overline{\boldsymbol{z}_j} - \overline{\boldsymbol{z}}) = 0$, thus the proof of Theorem 1 is still applicable. □

Corollary 2. *If the snippets are pooled using max pooling (Eq. (2)), then we have*

$$L(\boldsymbol{p}, y) \leq \max_{1\leq j\leq g} L(\boldsymbol{p}_j, y) + \min_{1\leq j\leq g} E_{\boldsymbol{p}}(\boldsymbol{z} - \boldsymbol{z}_j), \tag{19}$$

Proof. The definition of max pooling gives that $\boldsymbol{z} = \max\{\boldsymbol{f}_1, \boldsymbol{f}_2, \cdots, \boldsymbol{f}_m\} = \max\{\boldsymbol{z}_1, \boldsymbol{z}_2, \cdots, \boldsymbol{z}_n\}$. Then,

$$\begin{aligned}
& L(\boldsymbol{p}, y) \\
= \; & L(\boldsymbol{p}_j, y) - D_{KL}(\boldsymbol{p}\|\boldsymbol{p}_j) + (z_j^{(y)} - z^{(y)}) \\
& + \sum_{c=1}^C p^{(c)}(z^{(c)} - z_j^{(c)}), \\
\leq \; & L(\boldsymbol{p}_j, y) - D_{KL}(\boldsymbol{p}\|\boldsymbol{p}_j) + \sum_{c=1}^C p^{(c)}(z^{(c)} - z_j^{(c)}), \\
\leq \; & L(\boldsymbol{p}_j, y) + \sum_{c=1}^C p^{(c)}(z^{(c)} - z_j^{(c)}), \quad (1 \leq j \leq n) \qquad (20)
\end{aligned}$$

(21)

Thus,

$$\begin{aligned}
& L(\boldsymbol{p}, y) \\
\leq \; & \min_{1\leq j\leq n} \{L(\boldsymbol{p}_j, y) + \sum_{c=1}^C p^{(c)}(z^{(c)} - z_j^{(c)})\}, \qquad (22) \\
\leq \; & \max_{1\leq j\leq g} L(\boldsymbol{p}_j, y) + \min_{1\leq j\leq n} \sum_{c=1}^C p^{(c)}(z^{(c)} - z_j^{(c)}). \qquad (23)
\end{aligned}$$

□

Note that when the algorithm converges, the term $\min_{1\leq j\leq g} E_{\boldsymbol{p}}(\boldsymbol{z} - \boldsymbol{z}_j)$ becomes close to zero, thereby we will still have $L(\boldsymbol{p}, y) \leq \max_{1\leq j\leq g} L(\boldsymbol{p}_j, y)$ in the end.

APPENDIX B
MORE ILLUSTRATIONS ON THE KL-RULE

In the main text, we have illustrated the effectiveness of KL-rule in Figure 6 and 6. Here, we provide more experimental results to illustrate its benefits in Figure.

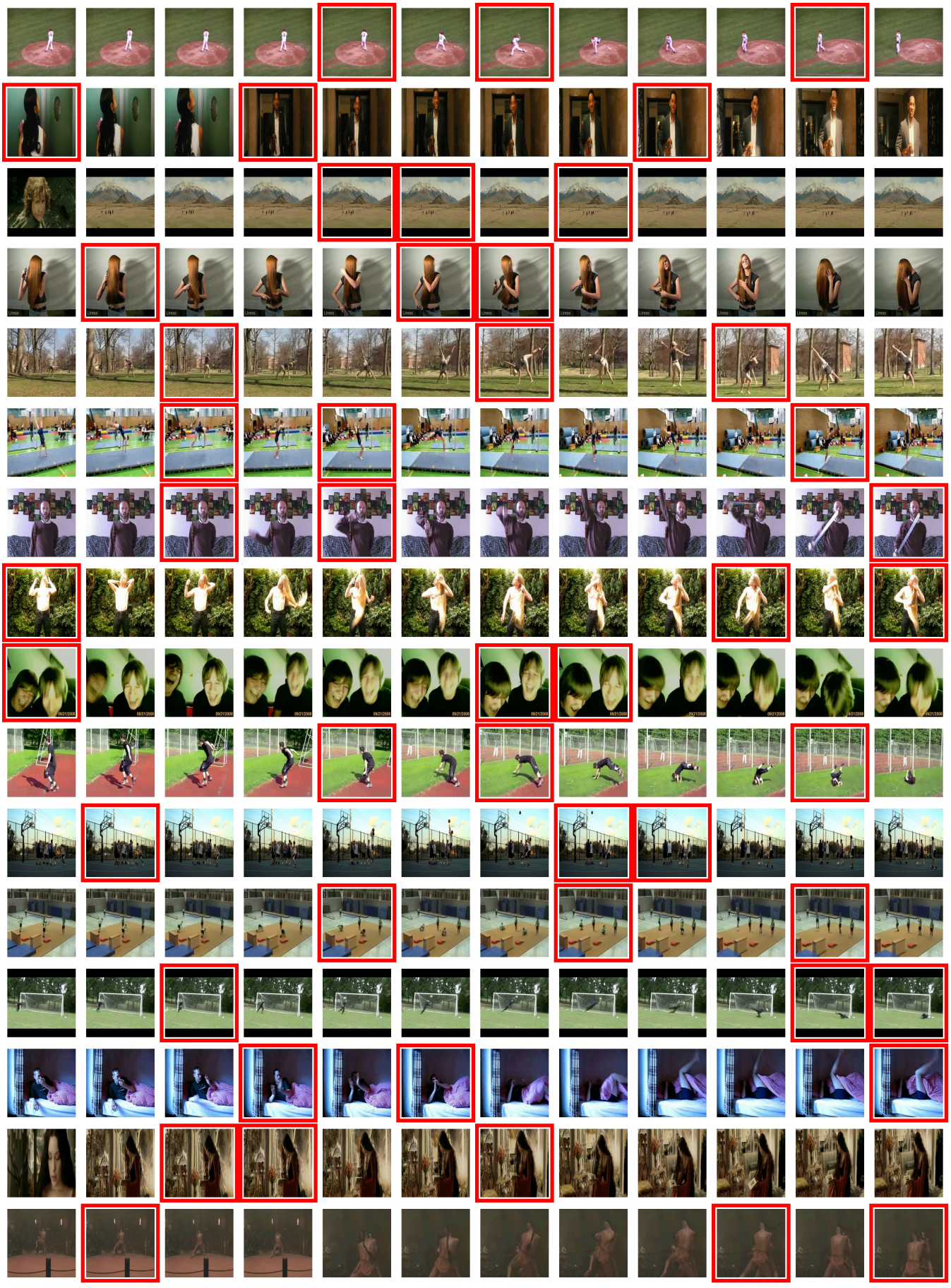


Fig. 8. Illustration on the testing example from the HMDB dataset. The experimental settings are the same as Figure 6. Each column displays one input video where the frames denoted by red box are the selected frames.