Adaptation-Oriented Feature Projection for One-Shot Action Recognition

Yixiong Zou, Yemin Shi, Daochen Shi, Yaowei Wang, *Member, IEEE*, Yongsheng Liang, and Yonghong Tian, *Senior Member, IEEE*

***Abstract*—One-shot action recognition aims at recognizing actions in unseen classes in cases where only one training video is provided. Compared with one-shot image recognition, one-shot learning on videos is more difficult due to the fact that the temporal dimension of video may lead to greater variation. To handle this variation, it is important to conduct further adaptation in the one-shot training process, despite the scarcity of the training data. While meta-learning is an option for facilitating this adaptation, it cannot be directly applied for two reasons: first, deep networks for action recognition can make current meta-learning methods infeasible to run because of their high computational complexity; second, due to the greater variation in actions, the adapted performance may not be higher than the un-adapted one, making it difficult to train the model by means of meta-learning. To address these problems and facilitate the adaptation, we propose the Adaptation-Oriented Feature (AOF) projection for one-shot action recognition. We first pre-train the base network on seen classes. The output of the network is projected to the adaptation-oriented feature space by fusing the important feature dimensions that are sensitive to adaptation. Subsequently, a small dataset (a.k.a. task) is sampled from seen classes to simulate the unseen-class training and testing settings. The feature adaptation is performed on the training data of this task to integrate the distribution information of the adapted feature. In order to reduce over-fitting, the triplet loss is applied to handle temporal variation with fewer parameters during the adaptation. On the testing data of this task, the losses on both adapted and un-adapted features are calculated to train the projection matrix. This sampling-adaptation-training procedure is then repeated on seen classes until convergence. Extensive experimental results on two challenging one-shot action recognition datasets demonstrate that our proposed method outperforms state-of-the-art methods.**

***Index Terms*—One-shot action recognition, adaptation-oriented feature projection, AOF, fast adaptation.**

I. Introduction

RECENTLY, deep learning has achieved state-of-the-art performance in action recognition. However, the training of deep networks largely relies on the availability of a large amount of training data. As such, traditional action recognition methods (e.g., [1], [2]) often train and test on the same classes with a large number of training samples. However, even with large-scale datasets, these methods remain limited in multiple aspects, particularly because not all objects or actions in our lives can be encompassed within the labels provided in the training set. To address this problem, one-shot/few-shot[1] learning has attracted widespread attention in the field of object and action recognition, which aims at recognizing new objects or actions (*unseen classes*) given only one, or only a few, training samples. Nevertheless, successfully learning from only a few examples still remains a key challenge in machine learning. This is mainly because the standard supervised learning paradigm does not offer a satisfactory solution for the rapid learning of new concepts from a small amount of data.

Limited by the scarce training samples in one-shot learning, current works always handle this problem through knowledge transferring. That is, apart from unseen classes, we have access to *seen classes* with non-overlapping classes and large amounts of data for pre-training (e.g., [3], [4]). It is therefore necessary to make good use of seen-class training data and to learn effectively from unseen-class training data. As an alternative to regular transfer learning methods (e.g., [5], [6]), distance learning is frequently used to learn an embedding space on seen classes where unseen-class samples are distinguishable [7]. For example, Matching Network [4] learned an embedding space on seen classes by simulating unseen-class testing settings, and subsequently obtained the testing image's label by comparing the nearest neighbor distance. Moreover, Prototypical Network [8] extended Matching Network to set a prototype for each unseen class, while Relation Network [9] further substituted the Euclidean distance by a trainable distance metric.

[1]For simplicity, we use the term *one-shot* to represent both *one-shot* and *few-shot* in the rest of this paper.

Manuscript received February 26, 2019; revised December 8, 2019; accepted January 26, 2020. Date of publication February 6, 2020; date of current version November 18, 2020. This work was supported in part by the National Key R&D Program of China under Grant 2017YFB1002400, in part by the National Natural Science Foundation of China under Grants U1611461 and 61825101, and in part by the NVIDIA NVAIL Program and the NVIDIA SaturnV DGX-1 AI supercomputer. The associate editor coordinating the review of this manuscript and approving it for publication was Prof. Mohammed Daoudi. *(Corresponding author: Yonghong Tian.)*

Yixiong Zou, Yemin Shi, and Yonghong Tian are with the National Engineering Laboratory for Video Technology, School of EE&CS, Peking University, Beijing 100871, China, and also with the Pengcheng Laboratory, Shenzhen 518066, China (e-mail: zoilsen@pku.edu.cn; shiyemin@pku.edu.cn; yhtian@pku.edu.cn).

Daochen Shi is with the NVIDIA AI Technology Center, Beijing 100020, China (e-mail: daochens@nvidia.com).

Yaowei Wang is with the Pengcheng Laboratory, Shenzhen 518066, China (e-mail: wangyw@pcl.ac.cn).

Yongsheng Liang is with the School of Electronic and Information Engineering, Harbin Institute of Technology, Harbin 150001, China (e-mail: liangys@hit.edu.cn).

Color versions of one or more of the figures in this article are available online at https://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TMM.2020.2972128

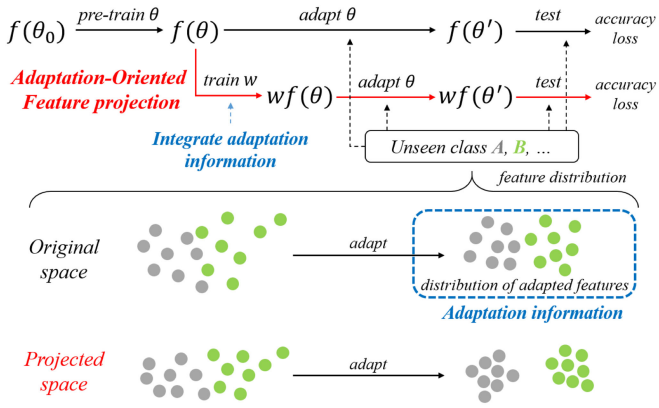

Fig. 1. **Illustration of the AOF projection.** After pre-training, the base network parameters θ need to adapt to unseen classes, and then evaluate on these classes. To facilitate the adaptation, we propose to project the base network output $f(\theta)$ to the adaptation-oriented feature $w \cdot f(\theta)$ by training a projection matrix w to integrate the adaptation information (i.e., the distribution of adapted features, marked in blue). By fusing the important feature dimensions, both the feature and the adaptation will be improved in the projected space.

It should be noted that distance-learning-based methods assume that the embedding space learned on seen classes is good enough to distinguish unseen classes. As images are simpler than videos, this assumption holds more easily for one-shot image recognition. However, this assumption may not hold for one-shot action recognition, as video's temporal dimension may lead to greater variation that will inevitably influence the distance comparison. Therefore, further training on unseen classes (also called *adaptation*) is needed in order to rectify the embedding space learned on seen classes. With this in mind, we present a typical paradigm for one-shot action recognition in Fig. 1. First, distance-learning-based methods can provide the pre-trained parameters θ on seen classes. For unseen classes, it is necessary to train the model on the training data to adapt θ accordingly. Finally, the performance on the unseen-class testing data is evaluated on the adapted parameters θ'. As the adaptation is vital, we consider utilizing meta-learning to facilitate it by finding the parameters sensitive to the adaptation [10]. To achieve this goal, while pre-training on seen classes, we constrain the adapted embedding space to be distinguishable, integrating the distribution information of the adapted features (i.e., adaptation information, marked in blue in Fig. 1) [10]. That is, we first sample a small dataset (a.k.a. *task*, containing individual training and testing data) from the seen classes to simulate the unseen-class testing setting. We then adapt the features to the sampled training data and calculate the loss on the sampled testing data. Moreover, the adaptation information complements current distance-learning-based methods which only constrain the un-adapted embedding space.

Essentially, the basic idea behind meta-learning involves taking the adaptation information into account. Some works in one-shot learning have incorporated meta-learning. For example, [11] used an LSTM [12] as the meta learner and generated gradients for the base network, while MAML [10] learned an initialization for one-shot training by constraining the loss of the adapted feature on seen classes. Meta-SGD [13] extended MAML and further learned an updating policy for adaptation. However, these methods cannot be applied to one-shot action recognition for two main reasons. First, all of these methods use small networks. For one-shot learning on images, a simple CNN (usually four modules with a convolution layer, a batch norm layer [14], a max pooling layer and ReLU [15] in each module) will be enough to achieve good performance. However, for action recognition datasets such as UCF11 [16] and HMDB51 [17], deeper networks with more parameters are applied [18], such as GoogLeNet [19]. Accordingly, it will be infeasible to run these methods, which may produce second-order gradients for all parameters with greater computational complexity. Second, these methods, which are designed for image recognition, assume that the adapted model will perform better than the un-adapted model (as this always holds for image recognition). However, for action recognition, due to the fact that greater variations exist due to the temporal dimension that may influence their recognition, it is more difficult to find an ideal adaptation method that will assure performance improvements in the adaptation, which contradicts with the assumption and makes it more difficult for the model to converge when conducting meta-learning [20].

To utilize the adaptation information while obtaining both good performance and feasible computational complexity, in this paper, we propose the **A**daptation-**O**riented **F**eature (AOF) projection for one-shot action recognition. After pre-training the base network on seen classes, we project the important dimensions, which are sensitive to adaptation, in the base network output (base network feature) to the adaptation-oriented feature. In order to simulate the one-shot learning setting on unseen classes, tasks containing individual training and testing data are sampled from seen classes. Subsequently, for each task, we adapt the feature on its training data so as to integrate the distribution information of the adapted feature. The triplet loss is applied in the adaptation to handle the temporal variation, which is aimed at reducing over-fitting by utilizing fewer parameters. By jointly combining the losses of the adapted and un-adapted features on the testing data, the projection matrix is trained to get the importance of each feature dimension. We repeat the above sampling-adaptation-training procedure until convergence occurs. Finally, we also use the triplet loss to adapt the model to unseen classes. Compared with current distance-learning-based methods, by fusing the important feature dimensions that are sensitive to the adaptation, we can improve both the extracted feature and the adaptation. Compared with current meta-learning-based methods, by training only the projection matrix instead of the whole network, our method has a feasible computational complexity. Moreover, by jointly combining the losses on both adapted and un-adapted features, our method can achieve better performance for one-shot action recognition.

A preliminary version of this work has been published in [21]. In this previous work, we proposed utilizing the triplet loss to handle the temporal variation with fewer parameters compared with the state-of-the-art Matching Video Net [18]. To learn an initialization for one-shot learning on unseen classes, we used distance learning to train the base network. The distribution of features before and after the adaptation on unseen classes can be

found in Fig. 1, indicating by *Original space*. To make better use of seen-class data and enable the model to learn effectively from unseen-class data (the distribution of features before and after the adaptation in the projected space is shown in Fig. 1, marked by *Projected space*, which is better than that of the original space), we extend our previous work in three aspects. First, we propose to use a projection matrix to project the base network feature in order to integrate the adaptation information on seen classes with a feasible computational complexity. Second, by constraining both the loss of the adapted and un-adapted feature, we train this projection matrix to improve both the feature and the adaptation, with good resulting performance. Third, extensive experiments are conducted on two challenging one-shot action datasets to evaluate the effectiveness of the proposed method.

The remainder of this paper is organized as follows. In Section II, we discuss the related work. In Section III, we present our method in detail. Extensive experiments are reported in Section IV. Finally, we conclude the paper in Section V.

II. Related Work

In this section, we will first discuss the state-of-the-art action recognition methods so as to highlight the necessity of introducing one-shot learning for videos. Subsequently, we will review the representative works in one-shot learning, including distance-learning-based and meta-learning-based methods.

A. Action Recognition

In essence, action recognition involves categorizing actions or behaviors in a video into several pre-defined classes. Two-stream [1] is a widely-used baseline method that uses the RGB stream and the optical flow stream to capture both the appearance and the motion information. To combine the most distinguishable parts of these two streams, [22] proposed using the attention mechanism to jointly train the networks for both streams. [23] took the clutter nature of body parts into account and selected the most discriminative parts in each stream. Later, [24] proposed FASNet to encode and fuse the most relevant and informative semantic cues for action recognition. Beyond these two streams, [25] and [26] further proposed using a projected dense trajectory descriptor to form the third stream. Moreover, to model the long-short dependency of an action, [27] proposed to aggregate spatial-temporal features with LSTMs, while [28] proposed a shuttle-like RNN to better handle the long-term dependency. In addition to these methods using 2D convolutional neural networks, some works such as [29], also proposed a 3D convolutional network to integrate both the spatial and the temporal information. Furthermore, [30] improved this model to recognize actions with arbitrary image sizes and video lengths. To reduce the size of C3D networks, [31] proposed a pseudo-3D network to simulate the original 3D network, thereby reducing the computational cost while maintaining the performance. Moreover, [32] used an attention mechanism to further utilize the information from the skeleton to assist action recognition.

However, these methods still require a vast number of training samples to train a model. To enable the model to work well when training samples are hard to get, such as anomaly actions, we need to focus on how one-shot learning can be applied to action recognition in videos.

Another related domain is gesture recognition [33], which aims to recognize human gestures for human-computer interaction. Some works on one-shot learning such as [34], [35] have been carried out. However, as the search space of gestures is much smaller than that of general actions, some handcrafted methods (such as Bag-of-Words in [34] and Histogram Oriented Gradients (HOG) in [35]) could be even more suitable than deep-learning-based methods in this field, although these are not directly applicable for one-shot action recognition.

B. Distance-Learning-Based One-Shot Learning

Distance learning plays an important role in one-shot learning. It involves learning an embedding space, in which intra-class samples are closer to each other while inter-class samples are farther. After training on seen classes, this embedding space can be directly used to distinguish samples in unseen classes. This concept has been widely applied in other fields, including Face Recognition [7] and Person Re-identification [36]. Following this idea, [37] proposed to use a Siamese network to build a binary classifier for input image-pairs to learn the embedding space. Currently, the evaluation protocol of one-shot learning is based on the N-way K-shot recognition job (i.e., training to recognize N classes with K training samples for each). To fit this evaluation protocol, Vinyals *et al.* [4] thought that training on seen classes should simulate the testing setting on unseen classes. Accordingly, these authors organized the training samples in the N-way K-shot style, which they denoted as *Task* or *Episode*,[2] and trained the embedding space by building an N-way classifier. Subsequently, Prototypical Network [8] extended Matching Network by substituting the cosine distance with the Euclidean distance in order to implement the Bregman divergence. It then used a larger task size (i.e., larger N) on seen classes, and computed the prototype of each class by averaging the features. Despite its simplicity, it achieved better performance compared with other complex models. Relation Network [9] further substituted the Euclidean distance with a trainable distance metric to handle the distance measuring issue in nonlinear space. [38] proposed to adopt some marginal distance learning methods such as the contrastive loss or the triplet loss [7] with margins, making it possible to insert distance learning into any other method. Meanwhile, some other recently proposed methods are trained without organizing data in N-way K-shot tasks. [39] and [40] proposed to treat the weights in the fully connected layer as the prototype of each seen class when using the cosine distance. These methods used the feature extracted on each unseen sample to represent the prototype of its own unseen class. [41] extended [39] by further adding an attention loss on each task in order to train an attention mechanism to assist the prototype representation. For one-shot action recognition, as the only deep-learning-based method so far, Matching Video Net was proposed in [18] to use GRU [42] and the attention mechanism to train the embedding space.

[2]In this paper, we use *Task* to represent this setting.

Furthermore, since the adaptation to unseen-class videos is also an important issue, we can consider integrating the seen-class adaptation information via meta-learning to assist the unseen-class adaptation.

C. Meta-Learning-Based One-Shot Learning

Meta-learning-based one-shot learning methods divide learning into two levels. Learning within each task is referred to as *adaptation*, while learning across tasks is called *meta-training*. Within each task, meta-learning-based methods typically first adapt their model to the training data, and then calculate the loss of the adapted model on the testing data of this task. This loss is later used to meta-train the model across tasks. It should be noted that meta-training is performed on seen classes, while meta-testing is conducted on unseen classes.

In order to find a better adaptation strategy, Meta-LSTM [11] proposed using the updating rule of the hidden state in LSTM to simulate the gradient descent procedure, and learned an initialization as well as a gradient generator for one-shot learning. However, Meta-LSTM was relatively difficult to train. To resolve this issue, MAML [10] proposed to constrain the loss of the adapted feature, and took the gradient descent itself into account. In meta-training, second-order gradients were required to train a good initialization for each task. Meta-SGD [13] extended MAML to further learn a learning rate vector, enabling it to apply a variable-specific learning rate to each variable. DEML [43] utilized another auxiliary dataset in addition to seen classes to perform a classification task, allowing it to maintain a good generalization ability compared with the ordinary MAML or Meta-SGD. Moreover, Meta Network [44] divided the parameters into fast and slow weights, then used the combination of these two kinds of weights to produce the output.

However, the methods listed above cannot be directly applied to one-shot action recognition. While small networks can work well for one-shot learning on images, for one-shot action recognition, deeper networks are needed (e.g., VGG [45] for Matching Video Net); thus, these methods will suffer from increased computational complexity to the extent that they may even become infeasible to run. To make the meta-learning more lightweight, Reptile [46] proposed using first-order gradients to substitute second-order gradients in the MAML, and subsequently achieved better performance than both MAML and first-order MAML (FOMAML) [10] for one-shot learning. Note that Reptile is used as a baseline method in our experiments. Compared with Reptile, our method achieves better performance on one-shot action recognition.

III. Adaptation-Oriented Feature Projection for One-Shot Action Recognition

A. Task Description

We begin with introducing the task setting of one-shot learning used in this paper. In the typical machine learning setting, we are given a regular dataset D, which is divided into a training part D_{train} and a testing part D_{test}. We are required to train our model on D_{train} and then evaluate performance on D_{test}. In one-shot learning, we are given multiple regular datasets sampled from $\mathscr{D}$, such that each $D \in \mathscr{D}$ has its own D_{train} and D_{test}. As in most previous works (e.g., [4], [11]), we are required to conduct the N-way K-shot recognition task: i.e., for each D, there are K training samples for each of the N classes in D_{train}, and the performance is evaluated on these N classes in D_{test}. In previous works, D is referred to as either *episode* or *task*; for clarity we use *task* to represent this setting in the remainder of this paper.

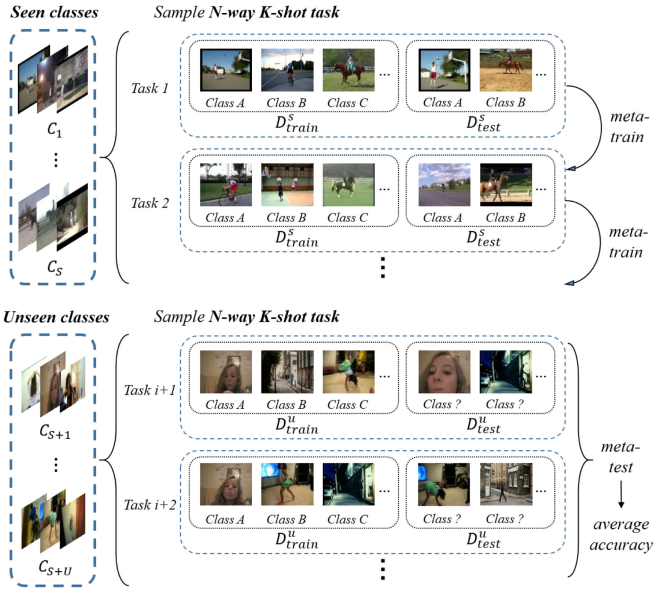

Fig. 2. Task setting in one-shot learning. A dataset is divided into seen classes (meta-training) and unseen classes (meta-testing). N-way K-shot recognition tasks (training with N classes and K samples in each class) are sampled. One-shot learning (adaptation) and testing are performed on D_{train} and D_{test} of these tasks, while meta-training is conducted across seen-class tasks. The performance is measured by averaging the accuracies of unseen-class tasks.

Following previous works, one-shot learning is formulated in a meta-learning way. The whole group of meta-sets $\mathscr{D}$ is divided into $\mathscr{D}_{train}$ and $\mathscr{D}_{test}$ for meta-training and meta-testing respectively. $\mathscr{D}_{train}$ is sampled from seen classes $C_1, \ldots, C_S$, while $\mathscr{D}_{test}$ is sampled from unseen classes $C_{S+1}, \ldots, C_{S+U}$. To make the two-level training clearer, the training within each task is termed *adaptation*, while the training across tasks is referred to as *meta-training*.

In this setting, we first need to sample $\mathscr{D}_{train}$ from the seen classes $C_1, \ldots, C_S$. For each seen-class task $D^s \in \mathscr{D}_{train}$, we adapt the model to D^s_{train} and calculate the loss on D^s_{test}; the loss of each task is used to meta-train the model across tasks. Once meta-training is complete, we sample $\mathscr{D}_{test}$ from the unseen classes $C_{S+1}, \ldots, C_{S+U}$. For each unseen-class task $D^u \in \mathscr{D}_{test}$, the accuracy on D^u_{test} is evaluated after adapting the model to D^u_{train}. The final performance on unseen classes is then averaged on $\mathscr{D}_{test}$. This setting is shown in Fig. 2.

As shown in Fig. 3, we first train our base network and the projection matrix on seen classes, and then adapt the base network and test on unseen classes. The base network pre-training will be introduced in Section III-B. Because the seen-class adaptation (which is conducted in the projection matrix training) is

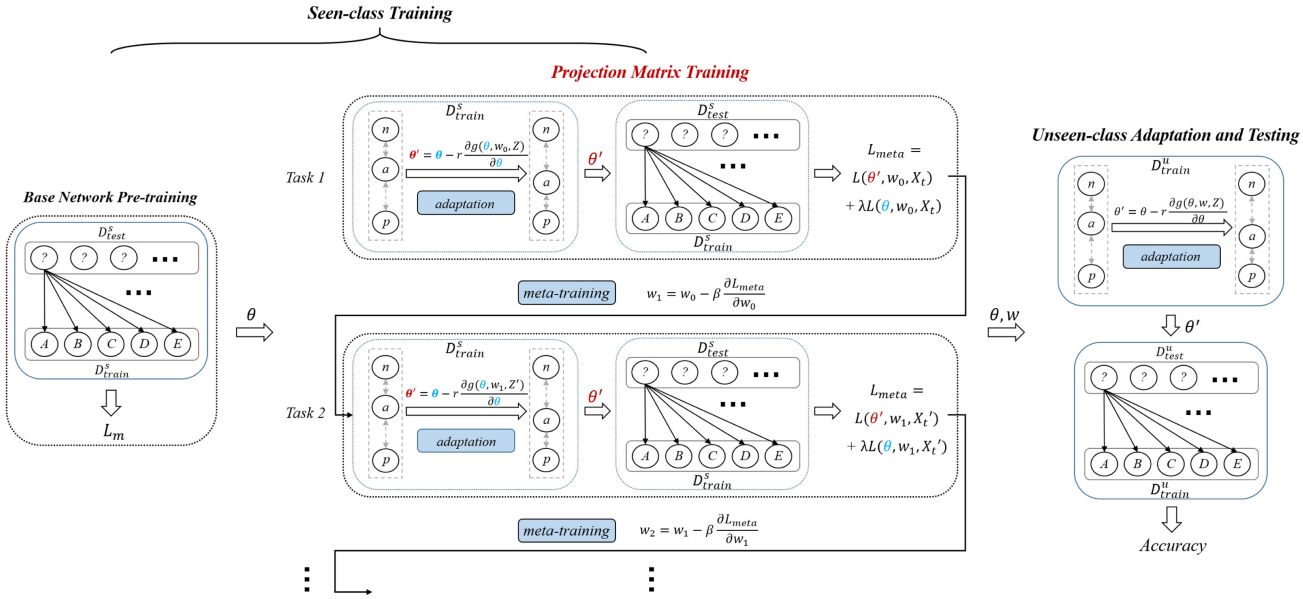

Fig. 3. **Framework of our method.** Tasks are sampled in the one-shot learning setting. We first pre-train the base network on seen classes, and then train the projection matrix. Finally, we adapt the model and test on unseen classes. In the base network training, we compare the cosine distances of samples from different classes, and minimize the distances of samples from the same class. In the projection matrix training, we use frames from each video to form triplets, and use the triplet loss to handle the temporal variation with fewer parameters. For each task, we first calculate the triplet loss $g(\theta, w, Z)$ on D^s_{train}, adapt the base network parameter θ, and use the adapted parameters θ' together with the un-adapted ones θ to calculate the meta loss $L(\theta', w, X_t) + \lambda L(\theta, w, X_t)$, and then meta-train the projection matrix w using the combined loss. The sampling-adaptation-training procedure is repeated until convergence. Finally, in unseen-class adaptation and testing, the triplet loss is applied to adapt the feature to unseen classes; the final performance is the averaged accuracy on unseen-class tasks.

a simulation of the unseen-class adaptation, we will then discuss the unseen-class adaptation and testing in Section III-C. Finally, the projection matrix training will be presented in detail in Section III-D.

B. Base Network Pre-Training

To learn a good initialization for tasks on unseen classes, we first pre-train our base network on seen classes using Matching Network [4], a distance-learning-based method.

In each iteration of this period, we sample a seen-class task containing testing videos (D^s_{test}) and training videos (D^s_{train}) from N classes, which will be referred to as $D^s_{test} = \{x_t\}$ and $D^s_{train} = \{(x_i, y_i)\}_{i=1}^N$ respectively;[3] here, x_i or x_t denote a video, while $y_i \in \mathbb{R}$ represents the label of x_i. Let $P(y_t|x_t, D^s_{train}) \in \mathbb{R}$ denotes the probability of x_t on class y_t, we compute this probability by

$$P(y_t|x_t, D^s_{train}) = \frac{e^{c(f(\theta,x_t), f(\theta,x_i))}}{\sum_j e^{c(f(\theta,x_t), f(\theta,x_j))}} \quad (1)$$

where $f(\theta, x) \in \mathbb{R}^{n\times 1}$ denotes an L_2-normalized feature extracted from video x by a CNN with parameters θ to be learned, and $c(\cdot)$ is a similarity function that typically refers to the cosine similarity. With the computed probability over all classes, we can use the cross-entropy loss computed as

$$L_m(\theta, X_t) = -\log(P(y_t|x_t, D^s_{train})) \quad (2)$$

[3]To simplify the formulation, we take one training video ($K = 1$) as an example. When $K > 1$, replace $f(\theta, x_i)$ with $\frac{1}{K}\sum_{j=1}^K f(\theta, x_i^j)$, where x_i^j is the jth video of class i.

to train the base network. For simplicity, we here use $X_t = \{x_t, D^s_{train}\}$ to represent the composition of x_t and D^s_{train}. This procedure is shown in Fig. 3 as the *Base Network Pre-training* part.

After initializing the base network on seen classes, we project the feature into another space using the matrix $w \in \mathbb{R}^{m\times n}$ and get the projected feature $f_p(\theta, w, x) = w \cdot f(\theta, x) \in \mathbb{R}^{m\times 1}$, which enables the model to learn effectively from one-shot training data in this space. Projection matrix training will be described in Section III-D.

C. Unseen-Class Adaptation and Testing

Because the seen-class adaptation, which is conducted in the projection matrix training, is a simulation of the unseen-class adaptation, we first introduce the unseen-class adaptation in this section. Assume that we have already trained a projection matrix. After the projection matrix training, we need to adapt the model and test on each task sampled from unseen classes. Unlike most distance-based works (e.g., [4], [8])) on image one-shot learning, which test the performance directly after initialization, we further train our model on the triplets sampled from D^u_{train} within each unseen-class task, enabling us to handle the temporal variation of each video.

The state-of-the-art work on one-shot action recognition, Matching Video Net [18], uses GRU [42] to handle the temporal variation. However, this also results in a large number of parameters being imported in their model, with the consequence that heavy over-fitting occurs. Therefore, we use the triplet loss

to handle the temporal variation by decreasing the intra-class distances and increasing the inter-class distances. Since far fewer parameters are thus needed in the adaptation, our model can effectively reduce the over-fitting on D^u_{train} compared with Matching Video Net.

As shown in the *Unseen-class Adaptation and Testing* part of Fig. 3, for each triplet, we first sample an anchor z^a_i and a positive video clip z^p_i from one class in D^u_{train} of this task, and then sample a negative video clip z^n_i from the other classes (this can come from D^u_{train} or seen classes). Subsequently, we calculate the loss of this triplet as follows:

$$\begin{aligned} g(\theta, w, Z) = \frac{1}{N_t}\sum_{i}^{N_t}\{&-\log(P(z^p_i)) + \log(P(z^n_i)) \\ &+ \max\{0, \alpha + ||f_p(\theta, w, z^a_i) - f_p(\theta, w, z^p_i)||^2_2 \\ &- ||f_p(\theta, w, z^a_i) - f_p(\theta, w, z^n_i)||^2_2\}\} \end{aligned} \tag{3}$$

where $||^2_2$ denotes the squared L_2-norm, N_t is the total number of sampled triplets, α is a predefined hyper-parameter, and $Z = \{z^a_i, z^p_i, z^n_i\}$ is the composition of z^a_i, z^p_i and z^n_i. We also add a cross-entropy loss in $g(\theta, w, Z)$ to make the adaptation more stable. The probabilities of the positive and negative samples are computed as follows:

$$P(z_i) = \frac{e^{c(f_p(\theta,w,z^a_i), f_p(\theta,w,z_i)))}}{\sum_{z\in\{z^n_i, z^p_i\}} e^{c(f_p(\theta,w,z^a_i), f_p(\theta,w,z)))}} \tag{4}$$

where z_i can be either z^p_i or z^n_i. After adapting to Z sampled from D^u_{train} for several iterations, given $D^u_{train} = \{(x^u_i, y^u_i)\}^N_{i=1}$ and the testing video $D^u_{test} = \{x^u_t\}$, we can calculate the probability distribution of the testing video as

$$P(y^u_t | x^u_t, D^u_{train}) = \frac{e^{c(f_p(\theta', w, x^u_t), f_p(\theta', w, x^u_i))}}{\sum_j e^{c(f_p(\theta', w, x^u_t), f_p(\theta', w, x^u_j))}} \tag{5}$$

where $y^u_i \in \mathbb{R}$ refers to the label of the video x^u_i, θ' denotes the adapted parameters, and the other symbols are the same as those in Eq. (1). With this probability distribution of the testing video x^u_t, we can get the accuracy of this task.

Moreover, by repeating this adaptation-testing procedure and calculating the accuracy of each task, we can obtain the average performance on unseen classes.

Note that if we here use f to replace f_p in Eq. (3), (4), and (5) and abandon the projection matrix training procedure, this method is equivalent to HED$_T$ in our previous work [21]. We use this previous work as a baseline in our experiments and refer to it as *HED$_T$*.

D. Projection Matrix Training

In addition to comparing the distance of samples in seen-class tasks as in [4] and [8], we also use the adaptation information within each task to facilitate the adaptation. We thus propose to use a projection matrix $w \in \mathbb{R}^{m\times n}$ to project the learned feature $f(\theta, x) \in \mathbb{R}^{n\times 1}$ to the adaptation-oriented feature $f_p(\theta, w, x) = w \cdot f(\theta, x) \in \mathbb{R}^{m\times 1}$, which enables the model to adapt more effectively to the one-shot training data. To make use of the adaptation information, we sample tasks from seen classes to simulate the unseen-class one-shot learning setting. To mimic the unseen-class adaptation, we perform the adaptation on sampled seen-class tasks in the same way it is performed on unseen classes; that is to say, both the triplet sampling and the loss function used in this seen-class adaptation have the same formulation as those used in Section III-C, except that all video clips are sampled from D^s_{train}.

For better clarity, as the meta-loss $-\log(P(y_t|x_t, D^s_{train}))$ on the testing data within tasks can be calculated either with or without adaptation, we refer to the meta-loss after adaptation as *Post-adaptation Loss*, while the meta-loss without adaptation is referred to as *Pre-adaptation Loss*. When training the projection matrix, we constrain both the pre-adaptation loss and the post-adaptation loss. We will next introduce each of these training methods separately.

1) Constraining the Pre-Adaptation Loss: Pre-adaptation loss means the loss calculated on D^s_{test} without adapting to D^s_{train}. For simplicity, we use

$$L(w, \theta, X_t) = -\log\left(\frac{e^{c(f_p(\theta,w,x_t), f_p(\theta,w,x_i))}}{\sum_j e^{c(f_p(\theta,w,x_t), f_p(\theta,w,x_j))}}\right) \tag{6}$$

to represent the classification loss calculated on D^s_{test}, where $f_p(\theta, w, x) = w \cdot f(\theta, x) \in \mathbb{R}^{m\times 1}$, while the other symbols are the same as those in Eq. (1). For preciseness, we use $\widetilde{w} = vec(w) \in \mathbb{R}^{mn\times 1}$ to represent the vectorization of w. Note that $\widetilde{w}$ and w are actually equivalent. We can update w as follows:

$$\begin{aligned} \widetilde{w} &= \widetilde{w} - \beta \cdot \frac{\partial L(w, \theta, X_t)}{\partial w} \\ &= \widetilde{w} - \beta \cdot \frac{\partial (w \cdot f(\theta, X_t))}{\partial w} \cdot \frac{\partial L}{\partial f_p} \\ &= \widetilde{w} - \beta \cdot (f(\theta, X_t) \otimes I_m) \cdot \frac{\partial L}{\partial f_p} \end{aligned} \tag{7}$$

where β is the learning rate and $\otimes$ denotes the Kronecker product. Note that $f(\theta, X_t)$ is irrelevant to w. As in the base network training, every time when we sample a task from the seen classes, we need to calculate the loss on D^s_{test}, and train the projection matrix w across tasks.

As constraining the pre-adaptation loss does not in fact require integrating the adaptation procedure, this loss can achieve the same goal as the base network training, i.e., training a good embedding by means of distance learning. From another point of view, constraining this loss can be viewed as adding a fully connected (FC) layer with no activation function, and then tuning it with the other layers fixed; that is, the improved performance may result from the deepened network structure. To avoid this kind of confusion and validate the effectiveness of the following methods, we use this method as a baseline in our experiments.

2) Constraining the Post-Adaptation Loss: As stated before, the adaptation to one-shot training samples is essential. In order to facilitate this adaptation, we first sample tasks from seen classes to simulate the unseen-class task setting. We then adapt the model to D^s_{train} by the triplet loss in Eq. (3) (note that now the triplet Z is sampled from D^s_{train}), and calculate the

post-adaptation loss of the adapted model on D^s_{test}, which allows us to make use of the distribution information of the adapted features.

More precisely, we use $g(\theta, w, Z)$ to represent the triplet loss described in Equation (3). While adapting to D^s_{train} within a task, we need to calculate the loss of this part of data, and adapt the parameters of the base CNN by

$$\theta' = \theta - r \cdot \frac{\partial g(\theta, w, Z)}{\partial \theta} \tag{8}$$

where r is the adaptation learning rate on this task. After this adaptation, we will have the new base network parameters θ', which can be used to calculate the classification loss $L(\theta', w, X_t)$ on the D^s_{test} of this task, as described in Eq. (6). Next, we update w by

$$\begin{aligned}
\widetilde{w} &= \widetilde{w} - \beta \cdot \frac{\partial L(\theta', w, X_t)}{\partial w} \\
&= \widetilde{w} - \beta \cdot \frac{\partial (w \cdot f(\theta', X_t))}{\partial w} \cdot \frac{\partial L}{\partial f_p} \\
&= \widetilde{w} - \beta \cdot \left(f(\theta', X_t) \otimes I_m + \frac{\partial f(\theta', X_t)}{\partial w} \cdot w^{\mathsf{T}} \right) \cdot \frac{\partial L}{\partial f_p} \\
&= \widetilde{w} - \beta \cdot (f(\theta', X_t) \otimes I_m \\
&\quad + \frac{\partial (\theta - r \cdot (\partial g(\theta, w, Z)/\partial \theta))}{\partial w} \frac{\partial f(\theta', X_t)}{\partial \theta'} \cdot w^{\mathsf{T}}) \cdot \frac{\partial L}{\partial f_p} \\
&= \widetilde{w} - \beta \cdot (f(\theta', X_t) \otimes I_m) \cdot \frac{\partial L}{\partial f_p} \\
&\quad - \beta \cdot r \cdot \frac{\partial (\partial [-g(\theta, w, Z)]/\partial \theta)}{\partial w} \frac{\partial f(\theta', X_t)}{\partial \theta'} \cdot w^{\mathsf{T}} \cdot \frac{\partial L}{\partial f_p} \\
&= \widetilde{w} - P_A - P_B
\end{aligned} \tag{9}$$

This procedure is represented in the *Projection Matrix Training* part of Fig. 3 (with λ set to 0, as will be described later). Note that during the adaptation, the projection matrix is fixed, while during the meta-training, the initialization of the base network is fixed; this is because the adapted parameters θ' within each task are discarded across tasks.

To analyze the post-adaptation loss, we use P_A and P_B to represent the corresponding terms in Equation (9). Note that if the adaptation learning rate r is set to 0, P_B will be 0 and the adapted base network parameter θ' will be equal to θ; thus, P_A will be degraded to the gradient of the pre-adaptation loss, as shown in Eq. (7).

Compared with the gradient of the pre-adaptation loss, P_A replaces the original base network parameters θ with the updated parameters θ'. Similar to the pre-adaptation loss, P_A is aimed at constraining the post-adaptation embedding to be distinguishable. However, unlike in the pre-adaptation loss, the computation of the adaptation from θ to θ' has been added to the gradient computation, which will train the model to find a good way to keep the adapted features discriminative enough. We interpret the training as fusing the important feature dimensions sensitive to adaptation [10], so that during the adaptation to D^u_{train}, updating on these dimensions will result in good performance promotion.

Another part of the gradient is P_B. On the one hand, as [47] and [10] suggest, the ReLU neural networks are locally almost linear, making the second derivatives close to zero. On the other hand, as the adaptation learning rate r is multiplied within this term (we always have $r << 1$), the effect of P_B can be much smaller than that of P_A. Therefore, it is the term P_A rather than P_B that plays a main role during the projection matrix training. Note that the term $\frac{\partial (\partial [-g(\theta, w, Z)]/\partial \theta)}{\partial w}$ is in P_B, which is aimed at adjusting the speed at which the model learns during the adaptation. Intuitively, we understand this as a way to find the most suitable learning speed for the adaptation, since either a higher or a lower speed will lead to poorly adapted features.

As P_A plays the main role in the projection matrix training, the key factor, how much the model learns during the adaptation, is controlled by the adaptation learning rate r and the number of adaptation steps. However, computing a multi-step adaptation is computationally expensive. Thus in the meta-training period, we use an one-step adaptation with a large r to simulate a multi-step adaptation with a small r [48]. If we set a small adaptation learning rate r_0 as in the meta-testing period, we can have

$$\theta_k = \theta_0 - r_0 \cdot \sum_{i=0}^{k-1} \frac{\partial L_i}{\partial \theta_i} \tag{10}$$

$$\approx \theta_0 - r_0 \cdot k \cdot \frac{\partial L_0}{\partial \theta_0} \tag{11}$$

where the parameters are updated for k steps in one task from the initial parameters θ_0 to the updated parameters θ_k. If we set the batch size to be k times that of the original one, and further set $r = k \cdot r_0$, we can use this approximation to efficiently proceed k step updates in only one step efficiently in the meta-training period.

As discussed in [48], this kind of approximation requires that the model cannot differ too much between steps. [48] also provides a warm-start mechanism to avoid large differences due to the training from scratch. However, for our model, an initial state has been well trained in the base network training procedure, and the projection matrix is fixed during the adaptation. Thus, if we set r properly, our base network will not differ too much between steps of the adaptation, meaning that this approximation still holds.

The effect of this adaptation learning rate will be discussed in Section IV-D2.

3) Combining the Pre-Adaptation and Post-Adaptation Loss: Unlike current meta-learning based methods, we combine the pre-adaptation loss and the post-adaptation loss.

Although constraining only the post-adaptation loss can indeed push the model to have a better adapted feature (which is the final goal of the proposed method), it may not be easy for the model to converge during meta-training under these circumstances. On the one hand, compared with constraining the pre-adaptation loss, constraining the post-adaptation loss requires a longer chain to compute the gradient for each variable, which may lead to a higher probability of gradient vanishing or explosion. On the other hand, compared with one-shot image

recognition (e.g., [10], [46]), video's temporal dimension may lead to greater variation; therefore, it is more difficult to find an ideal adaptation method to fit all action classes, leading to different performance improvements on different classes (e.g., methods can have different performance between short-time and long-time dependent actions [25], [26]). As a result, compared with adapting to images, it is more likely to over-fit D^s_{train} when adapting to videos, which is similar to the negative adaptation described in [20]. That is to say, merely constraining the post-adaptation loss may result in the model learning from a worse state (i.e., the over-fitted parameters, compared with the un-adapted ones). When encountering this kind of difficulty, merely constraining the post-adaptation loss may leave the model hard to converge during meta-training, leading to a lower performance than that would result from merely constraining the pre-adaptation loss. By contrast, combining the pre-adaptation loss will push the model to learn a good projected un-adapted feature, which can also help to reduce the post-adaptation loss by finding the distinguishable dimensions in the base network feature (i.e., these dimensions can overlap with those sensitive to the adaptation), thereby making the training relatively simpler. With the adaptation learning rate r in Eq. (9), this training will not be degraded to merely constraining the pre-adaptation loss. Thus, we finally use the combination of these two kinds of losses, which can be represented as

$$L_{meta} = L(\theta,' X_t, w) + \lambda L(\theta, X_t, w) \tag{12}$$

where λ is a pre-defined hyper-parameter. Experiments of this combination will be conducted in Section IV-D.

After training the base network and the projection matrix on seen classes, we can adapt and test on unseen-class tasks, as has already been illustrated in Section III-C.

4) Computational Complexity: Current widely used meta-learning algorithms (e.g., [10], [11]) require the computation of second-order gradients for all parameters within the base network, which makes them infeasible to run when deeper networks such as VGG [45] are applied. Compared with these methods, the computational complexity of our method is lower, as we only need the second-order gradients for the projection matrix w. In fact, the term $\frac{\partial(\partial[-g(\theta,w,Z)]/\partial\theta)}{\partial w}$ in P_B of Equation (9) is the only place where the second-order gradients are needed. For simplicity, we can understand this term as a matrix of size $n_\theta \times mn$ where n_θ denotes the number of parameters in the whole network. For the second-order gradients of all variables, the Hessian matrix will be of size $n_\theta \times n_\theta$, which means a computational complexity of $o(n_\theta^2)$. However, in practice, for deep networks such as VGG [45], n_θ can be as large as 10^8, while mn can be much smaller; this means the computational complexity of the second-order term in our method can be viewed as $o(n_\theta)$. As other terms in our methods only require the first-order gradients (with a computational complexity of $o(n_\theta)$), the overall computational complexity is still $o(n_\theta)$, which is much smaller than the original and makes it feasible for the GPU to run.

IV. Experiments

In this section, we will first describe the datasets used to evaluate our method, then present the implementation details of it. Finally, the experimental results will be reported.

A. Datasets

To evaluate our method and compare with the state-of-the-art methods, we follow [18] and [21] to conduct experiments on two challenging one-shot action recognition datasets: UCF11 [16] and HMDB51 [17].

UCF11 is a challenging dataset containing 11 actions and 1,600 videos in total. Videos in this dataset vary in camera motions, object appearances and poses, object scales, view-points, cluttered backgrounds, illumination conditions, and so on [16]. Following the experimental settings used in both [18] and [21], we randomly split 11 classes in UCF11 into 6 seen classes and 5 unseen classes, then repeat experiments on 6 different random splits to obtain the average 5-way 1-shot and 5-shot accuracy.

HMDB51 is another challenging action dataset collected from various sources, including movies and web videos. It contains 6,849 clips and 51 action categories, with at least 101 clips in each category. These categories contain both gentle actions, such as smiling or smoking, and drastic actions, such as shooting balls or swinging baseball, thus making it even more challenging. To evaluate our method, we also follow the experimental settings used in [18] and [21] to randomly split 51 classes into 41 seen classes and 10 unseen classes, and then repeat experiments on 12 different random splits to obtain the average performance.

B. Implementation Details

We use Tensorflow [49] to implement our model and use GoogLeNet [19] as our base network, following the settings in [21]. Experiments are conducted on NVIDIA Tesla V100 GPU. As shown in Section III-D, our method requires calculation of the second-order gradients. However, the standard GoogLeNet contains the local response normalization (LRN) layer, but Tensorflow does not support the calculation of the second-order gradient of that layer. Considering that LRN may not be useful as suggested in [14], we abandon this layer in our base network. This modified GoogLeNet is then pre-trained on ImageNet [50], as in [21].

After initializing the base network on ImageNet, we then pre-train the base network on the seen classes of each dataset, as described in Section III-B. In the projection matrix training period, we set the adaptation learning rate r to 0.03 and set the number of adaptation steps to 1, as the approximation of a multi-step update has been included in Section III-D2. We set the meta learning rate to 0.1 and decay to 0.01 after 3000 iterations on seen classes. For the meta-training, we set the meta batch size to 4; that is, the average gradient will be back propagated after a batch of tasks of this size is forwarded. For the input images, we resize the raw images to 256 $\times$ 256, then randomly crop them to 224 $\times$ 224, and randomly flip them left and right. Next, we randomly sample one image to represent each video during

TABLE I
5-WAY 1-SHOT PERFORMANCE COMPARISON WITH STATE-OF-THE-ART METHODS. RESULTS ON UCF11 ARE AVERAGED ON 6 SPLITS. METHODS LISTED IN THE FIRST SET ARE BASELINES

Method	UCF11(%)
Matching Video Net [18]	42.1
HED_T [21]	59.7
HED [21]	60.3
HED_T^* [21]	62.9
Reptile* [46]	62.5
Pre-Loss	63.4
Post-Loss	63.7
AOF	**64.1**

TABLE II
5-WAY 1-SHOT PERFORMANCE COMPARISON WITH STATE-OF-THE-ART METHODS. RESULTS ON HMDB51 ARE AVERAGED ON 12 SPLITS. METHODS LISTED IN THE FIRST SET ARE BASELINES

Method	HMDB51(%)
Matching Video Net [18]	42.9
HED_T [21]	46.9
HED [21]	47.1
HED_T^* [21]	47.8
Reptile* [46]	49.2
Pre-Loss	51.4
Post-Loss	52.5
AOF	**56.3**

meta-training. For meta-testing, we set the probability that the negative samples come from seen classes to 0.3 and the number of adaptation steps to 5. We follow [21] to sample 5 images with a stride of 10 to represent each video, without cropping or flipping, and then calculate the average feature for that video. To obtain the average performance on unseen classes, in each task we randomly sample 32 testing samples to form D_{test}^u, and calculate the average accuracy on 4000 randomly sampled tasks, which is taken as the final performance. In the 5-shot experiments, we follow [8] to calculate the average of five samples as the prototype of this class.

C. Comparison With State-of-the-Art Methods

The comparisons with state-of-the-art methods on UCF11 and HMDB51 are presented in Tables I and II respectively. Matching Video Net [18] and HED [21] are used as the state-of-the-art one-shot action recognition methods, while HED is our previous work. The full version of HED requires Hierarchical Temporal Memory (HTM) [51] to weight each frame; accordingly, to facilitate fair comparison, a simplified version of HED that only uses the triplet loss to adapt to unseen classes (referred to as HED_T) is used as our baseline. Moreover, because the base network used in HED and our method are not identical (GoogLeNet vs. modified GoogLeNet), we re-implemented HED_T using the modified GoogLeNet and denote the result as HED_T^*. In addition, as our method further adds a fully connected (FC) layer (without activation) to the top of the base network, performance promotion may result from the deepened network structure; to avoid any kind of confusion that may result from this, we also tested each component of our method and reported the performance after constraining the pre-adaptation loss. This baseline is denoted as *Pre-Loss*.

Reptile [46] is the state-of-the-art meta-learning method for few-shot learning, and can be considered as an improved version of MAML and Meta-SGD. Compared with MAML or Meta-SGD, it only requires the first-order gradients for training, making it feasible to run when using a deeper base network. To facilitate fair comparison, we use the same network structure (i.e., the modified GoogLeNet with a projection matrix) as the backbone: that is, we first train the base network on ImageNet, then tune it on the seen classes in each split of the dataset, and finally test it on the relevant unseen classes accordingly. We denote this baseline as *Reptile**.

Tables I and II report the effectiveness of the proposed method. To ensure fair comparison with [18] and [21], experiments were conducted on the RGB stream. We denote the projection matrix training with only the post-adaptation loss constraint as *Post-Loss*, while the training with both the pre-adaptation and post-adaptation loss is referred to as *AOF*.

As shown in Tables I and II, the modified version of GoogLeNet is better than the original version: that is, the performance is promoted from 59.7% to 62.9% on UCF11 and from 46.9% to 47.8% on HMDB51. Meanwhile, training the projection matrix by constraining the pre-adaptation loss can also improve the performance, mainly because the base network is deepened. Furthermore, constraining the post-adaptation loss is better than constraining the pre-adaptation loss, as indicated by the fact that the Post-Loss accuracies on both datasets are higher than that of Pre-Loss. However, the magnitude of this improvement is not identical on the two datasets. This is because the sizes of the seen classes in the two datasets are different: on average, UCF11 has about 870 seen-class videos in each split, while HMDB51 has 5500. This makes it easier to train on HMDB51 due to more meta-training data, even though videos in HMDB51 are much more complicated than those in UCF11. Finally, constraining both the pre-adaptation and post-adaptation losses can lead to the best performance, with about 4% improvement on HMDB51. This is because constraining these two losses allows us to better handle the training on those difficult videos, as discussed in Section III-D3.

It should be noted that Reptile, despite being the state-of-the-art meta-learning method, cannot achieve comparable performance even with a projection matrix attached. On UCF11, because Reptile is required to train all parameters, it easily over-fits on the seen classes due to the limited seen-class size. This limitation causes the performance to be even lower than HED_T, which only uses a modified GoogLeNet. It is also difficult for Reptile to converge on HMDB51, most likely because Reptile relies too heavily on the post-adaptation performance. In most cases, the post-adaptation performance should be higher than the pre-adaptation performance; on HMDB51, however, as more complicated variations exist in each video, it does not hold again. Thus, we can only use a very small adaptation learning rate to prevent the post-adaptation performance from dropping, making it very difficult to converge. Instead, our method further constrains the initial point of adaptation, i.e., the pre-adaptation loss. This loss pushes the model to find the distinguishable dimensions in the un-adapted base network feature, which can overlap with those dimensions sensitive to the adaptation, making the training relatively simpler, as discussed in Section IV-E.

TABLE III
5-WAY 5-SHOT PERFORMANCE COMPARISON WITH STATE-OF-THE-ART METHODS. RESULTS ON UCF11 ARE AVERAGED ON 6 SPLITS. METHODS LISTED IN THE FIRST SET ARE BASELINES

Method	UCF11(%)
Matching Video Net [18]	50.8
HED$^*_T$ [21]	79.9
Reptile* [46]	81.3
Pre-Loss	80.2
Post-Loss	81.5
AOF	**81.8**

TABLE IV
5-WAY 5-SHOT PERFORMANCE COMPARISON WITH STATE-OF-THE-ART METHODS. RESULTS ON HMDB51 ARE AVERAGED ON 12 SPLITS. METHODS LISTED IN THE FIRST SET ARE BASELINES

Method	HMDB51(%)
Matching Video Net [18]	58.0
HED$^*_T$ [21]	62.9
Reptile* [46]	64.7
Pre-Loss	66.5
Post-Loss	67.0
AOF	**69.5**

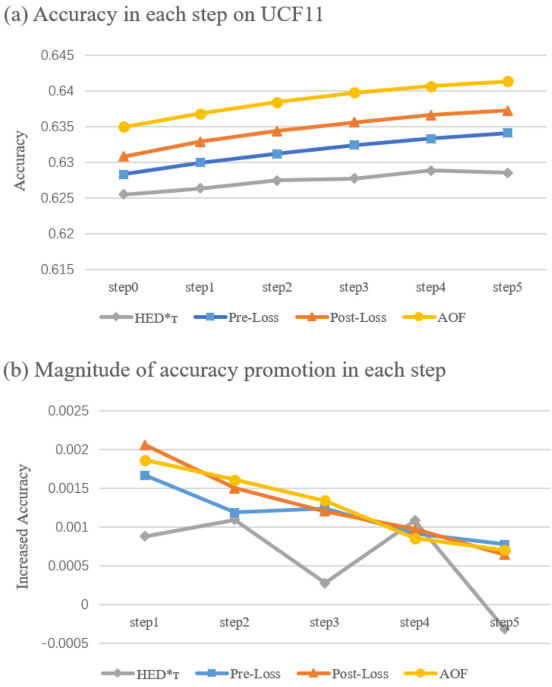

Fig. 4. (a) Accuracy in each step of adaptation on UCF11. (b) Magnitude of the accuracy promotion in each step on UCF11. Note that step(i) in (b) equals step(i) - step(i-1) in (a).

Therefore, we can have a better performance when training the projection matrix on these difficult videos.

To enable further comparison on 5-shot performances on UCF11 and HMDB51, we also reported the 5-shot performances on these two datasets in Tables III and IV respectively. These results follow the same trend as those in Tables I and II.

D. Evaluation of Meta-Training

1) Promotion to Adaptation: Fig. 4 and Fig. 5 show the promotion to the adaptation brought about by the post-adaptation

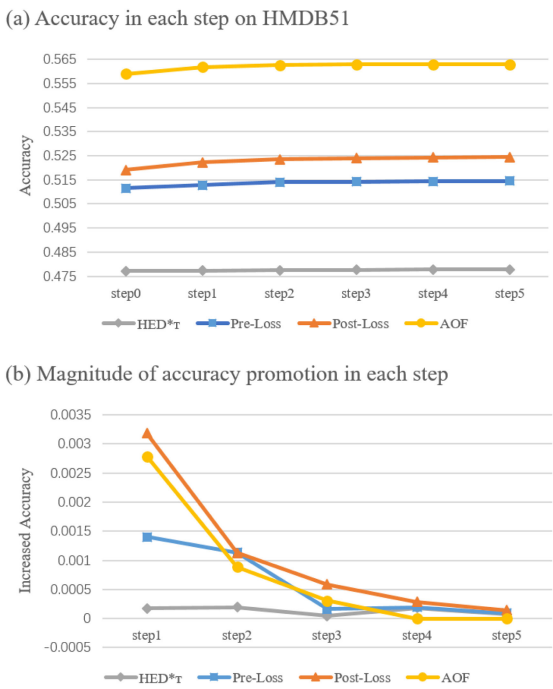

Fig. 5. (a) Accuracy in each step of adaptation on HMDB51. (b) Magnitude of the accuracy promotion in each step on HMDB51. Note that step(i) in (b) equals step(i) - step(i-1) in (a).

loss, where the average accuracy in each step during the unseen-class adaptation is plotted. Four methods are included: HED$^*_T$, Pre-Loss, Post-Loss and AOF. To better illustrate the magnitude of the performance promotion in each step, the promotion magnitude is also calculated in Fig. 4(b) and Fig. 5(b): that is, step(i) in Fig. 4(b) equals step(i) - step(i-1) in Fig. 4(a), and the same calculation is also conducted in Fig. 5(b).

From both Fig. 4(b) and Fig. 5(b), four conclusions can be drawn. Firstly, Pre-Loss can converge faster and better than HED$^*_T$, as the blue line is above the gray one at almost every step. This is because, by attaching a projection matrix, the deepened network is able to learn a better feature space to distinguish different classes, which is accomplished by finding distinguishable feature dimensions that overlap with those sensitive to adaptation.

Secondly, Post-Loss can effectively converge faster and better than both HED$^*_T$ and Pre-Loss, as shown by the fact that the orange line is always higher than both the gray and the blue ones in both Fig. 4(b) and Fig. 5(b). This is because the post-adaptation loss trains the projection matrix to fuse dimensions sensitive to the adaptation in the base network output, with the result that updating on these dimensions will lead to higher performance promotion, as discussed in Section III-D2.

Thirdly, Post-Loss is better than HED$^*_T$ and Pre-Loss even at the first step, as shown by step0 in Fig. 4(a) and Fig. 5(a). Note that the improved first-step performance also means that the projected feature is better than both the original feature and the pre-adaptation loss constrained feature. This is because the dimensions selected by the post-adaptation loss can overlap with those selected by the pre-adaptation loss; moreover, the adaptation information complements the regular

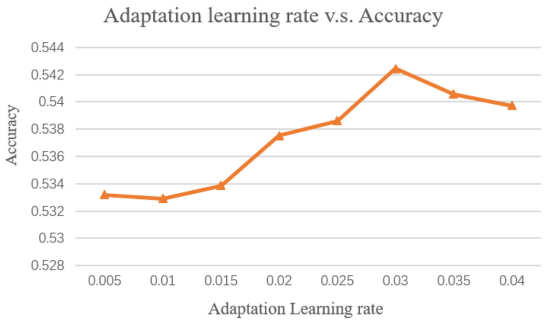

Fig. 6. Performances of the projection matrix trained with different adaptation learning rates on one random split of HMDB51.

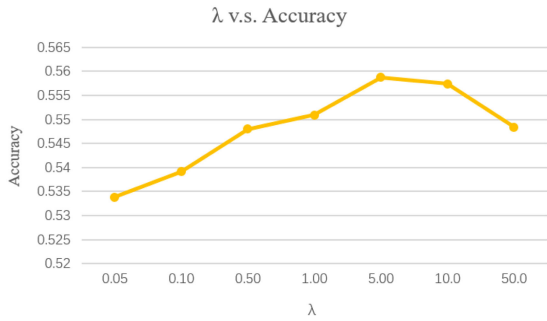

Fig. 7. Performances of the projection matrix trained with different λ on one random split of HMDB51.

distance-learning-based methods, which only constrains the pre-adaptation embedding. Intuitively, this constraint pushes the model to find its way towards better performance after adapting to the one-shot training data. There are two ways of achieving this: increasing the knowledge learned during the adaptation, and improving the un-adapted feature. Therefore, both the projected (un-adapted) feature and the adaptation are improved.

Fourthly, AOF achieves the best performance among these four methods in Fig. 4(a) and Fig. 5(a), especially on the challenging dataset HMDB51. As more complicated variations exist in each HMDB51 video, the adapted performance will not necessarily be higher than the un-adapted one, making it difficult to converge when constraining only the post-adaptation loss. In addition, combining the pre-adaptation loss will help to find the distinguishable feature dimensions that can overlap with those sensitive to the adaptation, making the training relatively simpler. Therefore, a projection matrix trained by constraining two losses is likely to be better than that trained via constraining the post-adaptation loss only. Further discussion will be conducted in Section IV-E. However, performances of AOF in Fig. 4(b) and Fig. 5(b) are not the best. This is because, due to the addition of the pre-adaptation loss, the training of the matrix pays less attention to improving the adaptation than Post-Loss does. All in all, however, AOF still outperforms both HED_T^* and Pre-Loss due to a better feature.

2) Effect of Adaptation Learning Rate: An important hyper-parameter in our model is the adaptation learning rate r, which controls how much the model should learn from the one-shot training data. To explore the effect of this hyper-parameter, the projection matrix is trained with different adaptation learning rates on one random split of HMDB51. The results are plotted in Fig. 6.

The results indicate that when the adaptation learning rate increases (while remaining smaller than 0.03), the performance increases accordingly. This is because the increase in the adaptation learning rate causes the adapted model to develop more differences from its un-adapted state, which constrains the model to learn more from the one-shot training data. On the other hand, a large adaptation learning rate can also be viewed as an approximation of a multi-step adaptation in each task [48]. This result is consistent with previous works (e.g., [10], [46]), which also demonstrate that the model will benefit from the multi-step adaptation. Compared with these previous works, the approximation of multi-step adaptation makes our model light-weight to the train the projection matrix.

Moreover, as was discussed in Section III-D2, the pre-requisite of this approximation is that the model cannot differ too much between steps. Accordingly, this prerequisite will not hold if the adaptation learning rate continues to increase, which will cause the performance to begin to decline when the adaptation learning rate grows larger than 0.03.

3) Balance Between the Pre-Loss and the Post-Loss: As discussed in Section III-D3, the hyper-parameter λ controls the balance between the pre-adaptation loss and the post-adaptation loss, which can also be viewed as the balance between easy-training and hard-training. To study the effect of this balance, we carried out experiments on different λ on one random split of the HMDB51 benchmark. As the number of training samples during the adaptation is small, the pre-adaptation model and the post-adaptation model are close; therefore, the sum $1+\lambda$ can be viewed to be multiplied with the meta learning rate. As a result, merely adjusting the hyper-parameter λ will import other irrelevant factors (i.e., different meta learning rate) into the experiments. To alleviate the effect of different sums of $1+\lambda$, the experiments are carried out on the normalized loss $L_{meta} = (L(\theta,' X_t, w) + \lambda L(\theta, X_t, w))/(1+\lambda)$, which is equivalent to the loss in Equation (12) given a fixed meta learning rate.

The results are plotted in Fig. 7; here, the X-axis denotes the hyper-parameter λ while the Y-axis denotes the corresponding 5-way 1-shot accuracy.

From Fig. 7 we can see that the performance reaches its peak when λ is around 5.0 to 10.0. With smaller λ, the effect of meta-training cannot sufficiently promote the performance. Furthermore, when λ is higher than 10.0, the model will rely too much on the post-adaptation loss, causing the model to be difficult to train and resulting in lower performance.

E. Exploration Experiments

To evaluate the effectiveness of our method on different classes, the performance on each class of HMDB51 is reported in Fig. 8, where four methods are included: HED_T^* is in gray,

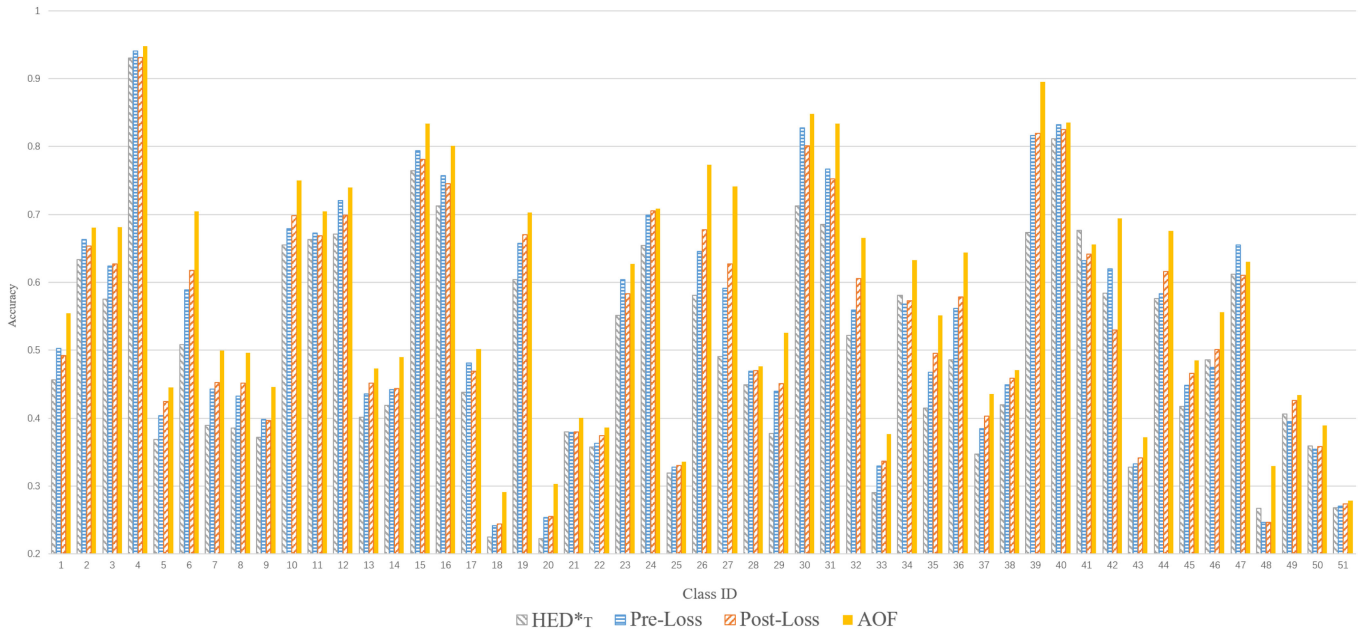

Fig. 8. Performance on each class of HMDB51. The results show that: (1) Pre-Loss is better than HED$^*_T$ in most classes; (2) Post-Loss is better than Pre-Loss in a large proportion of classes; (3) AOF is the best among most of the classes; (4) Combining this figure with Table V, we can find that for those classes whose Post-loss accuracy is lower than their Pre-loss accuracy, the first-step accuracy promotion in HED$^*_T$ is almost lower than 0. This means that the adaptation methods (i.e., the triplet loss) may not fit these classes due to the substantial variations in each action, which cause the model to over-fit these classes during the adaptation. As the adapted model is worse than the un-adapted one, it is harder for the model trained by merely constraining the post-adaptation loss to converge; however, combining the pre-adaptation loss and the post-adaptation loss can effectively handle this problem, as discussed in Section III-D3.

Pre-Loss is in blue, Post-Loss is in orange, and the combination of two losses (AOF) is in yellow (better viewed in color). Experiments are conducted on all random splits, as stated in Section IV-B. We calculate the performance of each class by accumulating its accuracy in all splits.

From this figure, three conclusions can be drawn. Firstly, attaching a projection matrix to the base network can promote the performance, as Pre-Loss is better than HED$^*_T$ in most classes. Secondly, constraining the post-adaptation loss is effective, as Post-Loss is better than Pre-Loss in a large proportion of classes. Thirdly, constraining both the pre-adaptation and post-adaptation loss results in the highest performance among all methods, as AOF is the best in most classes.

As discussed in Section III-D3, AOF can handle training on difficult classes, in which the projection matrix trained by the post-adaptation loss is worse than that trained by the pre-adaptation loss. To explore the effectiveness of AOF, these difficult classes are presented in Table V, along with the first-step accuracy promotion (i.e., the accuracy difference between the first-step adapted model and the un-adapted model) of HED$^*_T$ on these classes. For each difficult class, most of the first-step performance promotion is smaller than 0. That is, for these classes, the adaptation method (i.e., the triplet loss) may not fit these actions due to the substantial variation in each action, which causes the model to over-fit the scarce training data of them. Thus, the performance of the one-step adapted model will be lower than that of the un-adapted one, which is similar to the negative adaptation described in [20]. Training from the one-step adapted model will thus be harder, making it difficult to converge when constraining only the post-adaptation loss. This is the reason why Post-Loss cannot achieve higher performances than Pre-Loss on these classes.

TABLE V
FIRST-STEP PERFORMANCE PROMOTION OF EACH CLASS IN WHICH POST-LOSS ACCURACY IS LOWER THAN PRE-LOSS ACCURACY

Class ID	Class Name	HED$^*_T$ 1st step promotion
1	*brush hair*	-0.00156
2	*cartwheel*	-0.00080
4	*chew*	-0.00063
9	*draw sword*	+0.00034
11	*drink*	-0.00062
12	*eat*	-0.00013
15	*flic-flac*	-0.00250
16	*golf*	+0.00026
17	*handstand*	-0.00065
23	*kiss*	+0.00359
30	*push up*	-0.00046
31	*ride bike*	-0.00010
40	*smile*	-0.00386
42	*somersault*	-0.00044
47	*talk*	-0.00046

By contrast, combining the pre-adaptation loss will help to find the distinguishable dimensions in the base network feature, which can overlap with those sensitive to the adaptation, thereby simplifying the projection matrix training. Compared with Pre-Loss, further constraining the post-adaptation loss will complement it by finding feature dimensions sensitive to the adaptation. Therefore, constraining both the pre-adaptation and post-adaptation loss result in the best performance in most classes.

V. Conclusion

In this paper, we propose a new method, AOF, to integrate the adaptation information on seen classes using a projection matrix, with a feasible computational complexity. By jointly constraining the pre-adaptation and post-adaptation loss, we can achieve good performance when training this projection. Under this projection, both the feature and the adaptation are improved by fusing the important feature dimensions that are sensitive to adaptation. Experimental results on two challenging one-shot action recognition datasets, namely UCF11 and HMDB51, demonstrate that AOF can outperform the state-of-the-art methods.

We have demonstrated the importance of making better use of available data to improve the unseen-class performance in this study. However, different frames in each video and different regions in each frame are not equivalently important. Thus, the question of how better use could be made of this data is also an important topic in one-shot action recognition. An attention mechanism can help the model to focus on the most important parts in both the spatial and temporal domain, which is an approach that will certainly be investigated in the future work.

References

[1] K. Simonyan and A. Zisserman, "Two-stream convolutional networks for action recognition in videos," in *Proc. Conf. Neural Inf. Process. Syst.*, 2014, pp. 568–576.

[2] C. Feichtenhofer, A. Pinz, and A. Zisserman, "Convolutional two-stream network fusion for video action recognition," in *Proc. Conf. Comput. Vision Pattern Recognit.*, 2016, pp. 1933–1941.

[3] B. M. Lake, R. Salakhutdinov, and J. B. Tenenbaum, "Human-level concept learning through probabilistic program induction," *Science*, vol. 350, no. 6266, pp. 1332–1338, 2015.

[4] O. Vinyals, C. Blundell, T. Lillicrap, K. Kavukcuoglu, and D. Wierstra, "Matching networks for one shot learning," in *Proc. Conf. Neural Inf. Process. Syst.*, 2016, pp. 3637–3645.

[5] E. Tzeng, J. Hoffman, N. Zhang, K. Saenko, and T. Darrell, "Deep domain confusion: Maximizing for domain invariance," *CoRR*, 2014. [Online]. Available: http://arxiv.org/abs/1412.3474

[6] L. Shao, F. Zhu, and X. Li, "Transfer learning for visual categorization: A survey," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 26, no. 5, pp. 1019–1034, May 2015.

[7] F. Schroff, D. Kalenichenko, and J. Philbin, "FaceNet: A unified embedding for face recognition and clustering," in *Proc. IEEE Conf. Comput. Vision Pattern Recognit.*, 2015, pp. 815–823.

[8] J. Snell, K. Swersky, and R. Zemel, "Prototypical networks for few-shot learning," in *Proc. Advances Neural Inf. Process. Syst.*, 2017, pp. 4077–4087.

[9] F. Sung, Y. Yang, L. Zhang, T. Xiang, P. H. S. Torr, and T. M. Hospedales, "Learning to compare: Relation network for few-shot learning," in *Proc. IEEE/CVF Conf. Comput. Vision Pattern Recognit.*, 2018, pp. 1199–1208.

[10] C. Finn, P. Abbeel, and S. Levine, "Model-agnostic meta-learning for fast adaptation of deep networks," in *Proc. Int. Conf. Machine Learn.*, 2017, pp. 1126–1135.

[11] S. Ravi and H. Larochelle, "Optimization as a model for few-shot learning," in *Proc. Int. Conf. Learn. Represent.*, 2017, pp. 1–11.

[12] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, 1997.

[13] Z. Li, F. Zhou, F. Chen, and H. Li, "Meta-SGD: Learning to learn quickly for few-shot learning," 2017. [Online]. Available: https://arxiv.org/abs/1707.09835

[14] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proc. Int. Conf. Machine Learn.*, 2015, pp. 448–456.

[15] X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier neural networks," in *Proc. 14th Int. Conf. Artif. Intell. Stat.*, 2011, pp. 315–323.

[16] J. Liu, J. Luo, and M. Shah, "Recognizing realistic actions from videos 'in the wild'," in *Proc. Conf. Comput. Vision Pattern Recognit.*, 2009, pp. 1996–2003.

[17] H. Kuehne, H. Jhuang, E. Garrote, T. Poggio, and T. Serre, "HMDB: A large video database for human motion recognition," in *Proc. Int. Conf. Comput. Vision*, 2011, pp. 2556–2563.

[18] D. Kim, M. Lee, and N. Kwak, "Matching video net: Memory-based embedding for video action recognition," in *Proc. Int. Joint Conf. Neural Netw.*, 2017, pp. 432–438.

[19] C. Szegedy *et al.*, "Going deeper with convolutions," in *Proc. Conf. Comput. Vision Pattern Recognit.*, 2015, pp. 1–9.

[20] T. Deleu and Y. Bengio, "The effects of negative adaptation in model-agnostic meta-learning," 2018. [Online]. Available: https://arxiv.org/abs/1812.02159

[21] Y. Zou *et al.,* "Hierarchical temporal memory enhanced one-shot distance learning for action recognition," in *Proc. IEEE Int. Conf. Multimedia Expo.*, 2018, pp. 1–6.

[22] Y. Shi, Y. Tian, Y. Wang, and T. Huang, "Temporal attentive network for action recognition," in *Proc. 2018 IEEE Int. Conf. Multimedia Expo.*, 2018, pp. 1–6.

[23] S. Zhang, C. Gao, J. Zhang, F. Chen, and N. Sang, "Discriminative part selection for human action recognition," *IEEE Trans. Multimedia*, vol. 20, no. 4, pp. 769–780, Apr. 2018.

[24] J. Hou, X. Wu, Y. Sun, and Y. Jia, "Content-attention representation by factorized action-scene network for action recognition," *IEEE Trans. Multimedia*, vol. 20, no. 6, pp. 1537–1547, Jun. 2018.

[25] Y. Shi, Y. Tian, Y. Wang, and T. Huang, "Sequential deep trajectory descriptor for action recognition with three-stream CNN," *IEEE Trans. Multimedia*, vol. 19, no. 7, pp. 1510–1520, 2017.

[26] Y. Shi, W. Zeng, T. Huang, Y. Wang, "Learning deep trajectory descriptor for action recognition in videos using deep neural networks," in *Proc. IEEE Int. Conf. Multimedia Expo.*, 2015, pp. 1–6.

[27] J. Yue-Hei Ng *et al.,* "Beyond short snippets: Deep networks for video classification," in *Proc. Conf. Comput. Vision Pattern Recognit.*, 2015, pp. 4694–4702.

[28] Y. Shi, Y. Tian, Y. Wang, W. Zeng, and T. Huang, "Learning long-term dependencies for action recognition with a biologically-inspired deep network," in *Proc. Int. Conf. Comput. Vision*, 2017, pp. 716–725.

[29] D. Tran, L. Bourdev, R. Fergus, L. Torresani, and M. Paluri, "Learning spatiotemporal features with 3D convolutional networks," in *Proc. IEEE Int. Conf. Comput. Vision*, 2015, pp. 4489–4497.

[30] X. Wang, L. Gao, P. Wang, X. Sun, and X. Liu, "Two-stream 3-D convnet fusion for action recognition in videos with arbitrary size and length," *IEEE Trans. Multimedia*, vol. 20, no. 3, pp. 634–644, Mar. 2018.

[31] Z. Qiu, T. Yao, and T. Mei, "Learning spatio-temporal representation with pseudo-3D residual networks," in *Proc. IEEE Int. Conf. Comput. Vision*, 2017, pp. 5534–5542.

[32] Z. Fan, X. Zhao, T. Lin, and H. Su, "Attention based multi-view re-observation fusion network for skeletal action recognition," *IEEE Trans. Multimedia*, vol. 21, no. 2, pp. 363–374, Feb. 2019.

[33] F. Jiang, S. Zhang, S. Wu, Y. Gao, and D. Zhao, "Multi-layered gesture recognition with kinect," *J. Mach. Learn. Res.*, vol. 16, no. 1, pp. 227–254, 2015.

[34] L. Zhang *et al.*, "BoMW: Bag of manifold words for one-shot learning gesture recognition from kinect," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 28, no. 10, pp. 2562–2573, Oct. 2018.

[35] M. E. Cabrera, N. Sanchez-Tamayo, R. Voyles, and J. P. Wachs, "One-shot gesture recognition: One step towards adaptive learning," in *Proc. 12th IEEE Int. Conf. Autom. Face Gesture Recognit.*, 2017, pp. 784–789.

[36] L. Zheng, Y. Yang, and A. G. Hauptmann, "Person re-identification: Past, present and future," 2016. [Online]. Available: https://arxiv.org/abs/1610.02984

[37] G. Koch, R. Zemel, and R. Salakhutdinov, "Siamese neural networks for one-shot image recognition," in *Proc. Int. Conf. Mach. Learn. Deep Learn. Workshop*, 2015, vol. 2, pp. 1–8.

[38] Y. Wang *et al.*, "Large margin meta-learning for few-shot classification," in *Proc. 2nd Workshop Meta-Learn. NeurIPS*, 2018, pp. 1–8.

[39] S. Qiao, C. Liu, W. Shen, and A. L. Yuille, "Few-shot image recognition by predicting parameters from activations," in *Proc. IEEE Conf. Comput. Vision Pattern Recognit.*, 2018, pp. 7229–7238.

[40] H. Qi, M. Brown, and D. G. Lowe, "Low-shot learning with imprinted weights," in *Proc. IEEE Conf. Comput. Vision Pattern Recognit.*, 2018, pp. 5822–5830.

[41] S. Gidaris and N. Komodakis, "Dynamic few-shot visual learning without forgetting," in *Proc. IEEE Conf. Comput. Vision Pattern Recognit.*, 2018, pp. 4367–4375.

[42] K. Cho, B. Van Merriënboer, D. Bahdanau, and Y. Bengio, "On the properties of neural machine translation: Encoder-decoder approaches," in *Proc. 8th Workshop Syntax, Semantics Struct. Statistical Transl.*, 2014, pp. 103–111.
[43] F. Zhou, B. Wu, and Z. Li, "Deep meta-learning: Learning to learn in the concept space," 2018. [Online]. Available: https://arxiv.org/abs/1802.03596
[44] T. Munkhdalai and H. Yu, "Meta networks," in *Proc. Int. Conf. Machine Learn.*, 2017, pp. 2554–2563.
[45] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2014, [Online]. Available: https://arxiv.org/abs/1409.1556
[46] A. Nichol and J. Schulman, "Reptile: A scalable metalearning algorithm," 2018. [Online]. Available: https://arxiv.org/abs/1803.02999
[47] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," in *Proc. Int. Conf. Learn. Representations*, 2015, pp. 1–10.
[48] P. Goyal *et al.*, "Accurate, large minibatch SGD: Training imageNet in 1 hour," 2017. [Online]. Available: https://arxiv.org/abs/1706.02677
[49] M. Abadi *et al.*, "TensorFlow: Large-scale machine learning on heterogeneous distributed systems," 2016. [Online]. Available: https://arxiv.org/abs/1603.04467
[50] J. Deng *et al.,* "ImageNet: A large-scale hierarchical image database," in *Proc. IEEE Conf. Comput. Vision Pattern Recognit.*, 2009, pp. 248–255.
[51] J. Hawkins, S. Ahmad, S. Purdy, and A. Lavin, "Biological and machine intelligence (BAMI)," to be published.

Yixiong Zou received the B.S. degree from Nankai University, Tianjin, China, in 2016. He is currently working toward the Ph.D. degree with the School of Electrical Engineering and Computer Science, Peking University, Beijing, China. His research interests include machine learning, one-shot learning, one-shot action recognition, and computer vision.

Yemin Shi received the B.S. degree from Peking University, Beijing, China, in 2014. He is currently working toward the Ph.D. degree with the School of Electrical Engineering and Computer Science, Peking University, Beijing, China. His research interests include machine learning, anomaly detection, action recognition, and computer vision.

Daochen Shi received the M.S. degree from Peking University, Beijing, China, in 2016. He is currently working as Solution Architect with NVIDIA AI Technology Center, Beijing, China. His research interests include machine learning, recommender systems, reinforcement learning, and computer vision.

Yaowei Wang (Member, IEEE) received the Ph.D. degree in computer science from the Graduate University of Chinese Academy of Sciences, Beijing, China, in 2005. He is currently an Assistant Professor with the Pengcheng Laboratory, Shenzhen, China. He is also a Guest Assistant Professor with the National Engineering Laboratory for Video Technology, Peking University, Beijing, China. He is the author or coauthor of more than 40 refereed journals and conference papers. His research interests include machine learning, multimedia content analysis and understanding. His team was ranked as one of the best performers in the TRECVID CCD/SED tasks from 2009 to 2012 and PETS 2012. He is a member of CIE.

Yongsheng Liang received the Ph.D. degree in communication and information systems from the Harbin Institute of Technology, Harbin, China, in 1999. From 2002 to 2005, he was a Research Assistant with the Harbin Institute of Technology, where he has been a Professor with the School of Electronic and Information Engineering, since 2017. His research interests include video coding and transferring and machine learning and applications. He was the recipient of the Wu Wenjun Artificial Intelligence Science and Technology Award for Excellence and the Second Prize in Science and Technology of Guangdong Province.

Yonghong Tian (Senior Member, IEEE) is currently a Full Professor with the School of Electronics Engineering and Computer Science, Peking University, Beijing, China, and the Deputy Director of AI Research Center, Pengcheng Lab, Shenzhen, China. His research interests include computer vision, multimedia big data, and brain-inspired computation. He is the author or coauthor of more than 170 technical articles in refereed journals and conferences. He was/is an Associate Editor for the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS FOR VIDEO TECHNOLOGY (2018), IEEE TRANSACTIONS ON MULTIMEDIA (2014–2018), *IEEE Multimedia Magazine* (2018), and IEEE Access (2017). He co-initiated the IEEE International Conference on Multimedia Big Data (BigMM) and served as the TPC Co-Chair of BigMM 2015, and also served as the Technical Program Co-Chair of IEEE ICME 2015, IEEE ISM 2015 and IEEE MIPR 2018/2019, and the organizing committee member of ACM Multimedia 2009, IEEE MMSP 2011, IEEE ISCAS 2013, IEEE ISM 2015/2016. He is the steering member of IEEE ICME (2018) and IEEE BigMM (2015), and is a TPC Member of more than ten conferences such as CVPR, ICCV, ACM KDD, AAAI, ACM MM and ECCV. He was the recipient of the Chinese National Science Foundation for Distinguished Young Scholars in 2018, two National Science and Technology Awards and three ministerial-level awards in China, and received the 2015 EURASIP Best Paper Award for the EURASIP Journal on Image and Video Processing and the Best Paper Award of IEEE BigMM 2018. He is a senior member of CIE and CCF and a member of ACM.