Adversarial Action Prediction Networks

Yu Kong, *Member, IEEE,* Zhiqiang Tao, *Student Member, IEEE* and Yun Fu, *Senior Member, IEEE*

Abstract—Different from after-the-fact action recognition, action prediction task requires action labels to be predicted from partially observed videos containing incomplete action executions. It is challenging because these partial videos have insufficient discriminative information, and their temporal structure is damaged. We study this problem in this paper, and propose an efficient and powerful deep network for learning representative and discriminative features for action prediction. Our approach exploits abundant sequential context information in full videos to enrich the feature representations of partial videos. This information is encoded in latent representations using a variational autoencoder (VAE), which are encouraged to be progress-invariant. Decoding such latent representations using another VAE, we can reconstruct missing information in the features extracted from partial videos. An adversarial learning scheme is adopted to differentiate the reconstructed features from the features directly extracted from full videos in order to well align their distributions. A multi-class classifier is also used to encourage the features to be discriminative. Our network jointly learns features and classifiers, and generates the features particularly optimized for action prediction. Extensive experimental results on UCF101, Sports-1M and BIT datasets demonstrate that our approach remarkably outperforms state-of-the-art methods, and shows significant speedup over these methods. Results also show that actions differ in their prediction characteristics; some actions can be correctly predicted even though only the beginning 10% portion of videos is observed.

Index Terms—Action Prediction, Action Recognition, Sequential Context, Variational Autoencoder, Adversarial Learning.

1 Introduction

Predicting an action **before** the action execution ends in real-world videos is an emerging and important computer vision problem with a wide range of applications such as visual surveillance and traffic accident avoidance. In contrast to action recognition, action prediction approaches do not have the luxury of waiting for the entire action execution before having to infer the action label. For example, it would be beneficial if an intelligent system on a vehicle can predict a traffic accident before it happens; opposed to recognizing the dangerous accident event thereafter. More importantly, it is essential that the intelligent system can make accurate predictions at the very beginning stage of a video, for instance, when only the beginning 10% frames of a full video is observed. The prediction results would have dual benefits, not only would it demonstrate the comprehension of the entire action executions, but it would also serve an efficient and proactive alerting of forthcoming actions before they are fully executed.

Despite its importance, action prediction is a very challenging task because its input data are temporally incomplete, and decisions must be made based on such incomplete action executions. Nevertheless, certain actions are predictable at their early stage if particular temporal patterns are observed and temporal context is available. Consider, for example, a video of a triple jump. We could imagine that a player is very likely to jump after running since we have seen this type of sport elsewhere[1]. The sequential context of the full video observation provides us with the knowledge that the triple jump action consists of running and jumping, and how the action appearance evolves in the temporal domain. This crucial information transferred along the temporal axis is the key to action prediction as it helps us to understand the action evolution in the full action observation.

In this paper, we propose a novel Adversarial Action Prediction Network (AAPNet), an extension of our previous work DeepSCN [1]. Our goal is to learn representative and discriminative features from partial videos for action prediction. AAPNet encodes the sequential context information of a video in a latent disentangled representation that is invariant to appearance variations at various progress levels. In a latent feature space, latent representations of videos are clustered according to their action labels and progress levels. When only the beginning portion of a video is observed, the sequential context information in the latent representation is discovered and transferred to the partial video (see Figure 1), making the generated feature being representative enough to convey the entire information in the corresponding full video. In addition, the features and the classifiers are jointly trained, making the features discriminative and optimized for action prediction. Therefore, AAPNet enriches the feature representation for the partial video, and improves its discriminative power even though the representation is extracted from incomplete sequences.

AAPNet is based on variational autoencoder [2], [3] and adversarial learning [4], and is developed to accommodate sequential data. We use two variational autoencoders and share one encoder. The encoder is trained to disentangle

- *Yu Kong is with B. Thomas Golisano College of Computing and Information Sciences, Rochester Institute of Technology, Rochester, NY. This work was primarily done when Yu Kong was with Department of ECE, Northeastern University, Boston, MA.*
  E-mail: yu.kong@rit.edu.
- *Zhiqiang Tao is with Department of ECE, Northeastern University, Boston, MA.*
  E-mail: zqtao@ece.neu.edu.
- *Yun Fu is with Department of ECE and College of CIS, Northeastern University, Boston, MA.*
  E-mail: yunfu@ece.neu.edu.

Manuscript received February 9, 2018; revised February 30, 2018.

1. We acknowledge that the scene gist also plays an important role here.

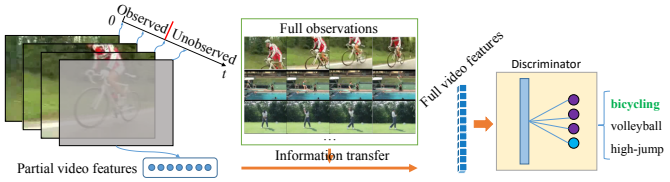


Fig. 1. Our AAPNet predicts the action label given an unfinished action video. Given features extracted from a partially observed video, AAPNet gains extra discriminative information from fully observed video, and generate more representative and discriminative features for action prediction.

the motion information that heavily depends on progress levels from the latent representation. It encourages the representation to encode primitive motion information that is shared among all progress levels as much as possible, thereby progress-invariant. Two decoders are utilized in our framework to generate the corresponding partial observation and full observation, respectively. The partial observation decoder regularizes the latent features to capture action evolutions in the input partial video. This will provide the full observation decoder with prior information about the partial observation, and help generate a representative feature representation that contains the entire motion information of the corresponding full video. To align the feature distribution of the generated full observation with the true full observation, we adopt an adversarial learning scheme and use a discriminator to differentiate the fake generated feature and the true feature directly extracted from full videos. The discriminator also classifies the features into various action categories, and thus is helpful for generating discriminative features for action prediction. The entire network learns feature representations and classifies actions jointly. Extensive results show that AAPNet outperforms state-of-the-art action prediction methods.

Our work focuses on *short-duration* prediction such as "biking" and "diving", while [5] focuses on long-duration compositional action prediction where an action can be further decomposed into semantic meaningful primitives. For example, an activity "make an omelet" can be decomposed into primitives "crack", "pour", "stir", etc. We simulate sequential data arrival while [6] assumes data are randomly observed in a sequence. We aim at predicting the label for a partially observed video. By comparison, [7], [8], [9] predict what will happen in the future, and [10], [11] localizes the starting and ending frames of an incomplete event.

The main contribution of this paper is the development of the adversarial action prediction network, which generates representative and discriminative features given partial action videos. It builds upon the popular variational autoencoder framework [2] to learn representative features by mapping partial video features into the feature space of full videos. The discriminative power of the learned features are further enhanced by learning action classifiers and feature learners jointly. In addition, we learn latent progress-invariant features to capture appearance information in partial videos using an encoder, and provide the corresponding full video features with a strong prior information. A particular technical challenge in our network is how to generate target features from such progress-invariant features. Inspired by the conditional GAN, we use a feature generator conditioned on the target progress level of full videos to address this problem. Extensive results on various datasets show superior results of our method over state-of-the-art action prediction methods.

This paper is an extension of our previous work [1]. These extensions include: a refined network architecture for action prediction; redefined encoder and decoders; a new discriminator for aligning feature distributions; an improved objective function; a joint training scheme for learning features and classifiers; and more experimental results on various datasets.

2 Related Work

Action recognition methods take as input fully observed videos and output labels of human actions. Existing approaches can be roughly categorized into low-level feature-based approaches [12], [13], [14], [15], [16], [17] and mid-level feature-based approaches [18], [19], [20], [21], [22]. Low-level features, such as dense trajectory [16] and poselet key-frames [15], utilize local appearance information and spatio-temporal structures, and have shown great success in action recognition. Mid-level feature-based approaches, such as semantic descriptions [18] or data-driven concepts [20], have shown to be capable of recognizing more complex human actions. Furthermore, some deeply learned features [23], [24] were recently proposed to learn high-level information for classification. However, most existing methods expect to observe temporally complete action executions. Their performance is unknown if they are given videos with temporally incomplete action executions. There are some interesting work [25], [26] that demonstrates reasonable action recognition accuracy can be achieved using a few frames in a video. This inspires us to study *at what stage an action can be predicted* or *action predictability* in this work.

Action prediction methods [6], [9], [27], [28] were proposed to predict the action given a partially observed video. Ryoo [27] proposed integral and dynamic bag-of-words approaches for action prediction. The former one models feature distribution variations over time, while the latter technique depicts the sequential nature of human activities. Cao *et al.* [6] generalized human activity recognition. In their work, frames were randomly removed in a video to simulate missing data. They formulated the problem as a posterior-maximization problem, where the likelihood is computed by feature reconstruction error using sparse coding. However, [6] suffers from high computational complexity as the inference is performed on the entire training data. Lan *et al.* [9] designed a coarse-to-fine hierarchical representation to capture the discriminative human movement at different levels, and used a max-margin framework for final prediction. Kong *et al.* [28] proposed a structured SVM learning method to simultaneously consider both local and global temporal dynamics of human actions. By enforcing a label consistency of temporal segments, the performance of prediction can be effectively improved.

The proposed approach is significantly different from existing action prediction and early detection approaches [6], [9], [10], [11], [27], [28]. The proposed AAPNet elegantly gains extra sequential context information from full videos

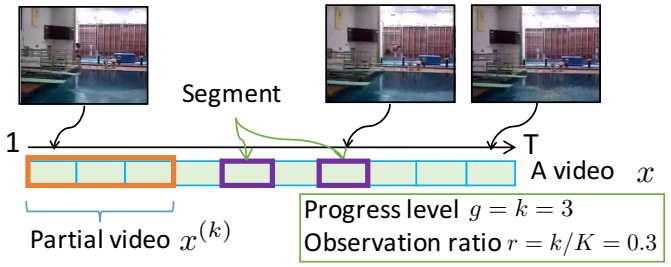


Fig. 2. Example of a temporally partial video, and graphical illustration of progress level and observation ratio.

to partial videos, while [10], [11], [28] capture increasing confidence score or decreasing detection loss in temporal sequence. Action models are computed by averaging action representations in training data [27], building action dictionaries [6] or describing actions at both coarse and fine levels [9]. By comparison, we build action models by transferring information from full videos in order to improve the discriminative power of partial videos. In addition, our approach learns features that are particularly optimized for action prediction, while [6], [9], [27], [28] only use hand-crafted features.

The prediction of future events was also investigated in other applications, such as predicting events in recommendation systems [29], [30], predicting future visual representation [31], and reasoning about the preferred path for a person [32], [33]. Their goals are different from our work as we focus on predicting the action labels of a video.

3 Revisit of DeepSCN

This section reviews the DeepSCN proposed in our prior work [1], and discusses its relationship to the new network proposed in this paper.

3.1 Problem Setup

Our goal is to predict the action class y of an action video $\mathbf{x}$ before the ongoing action execution ends [6], [27], [28]. We follow the problem setup described in [6], [9], [27], [28]. A complete video $\mathbf{x}$ containing T frames is uniformly segmented into K segments ($K = 10$ in this work), mimicking sequential video arrival at various observation ratios. Each segment contains $\frac{T}{K}$ frames. Note that for different videos, their lengths T may vary, causing different lengths in their segments. The k-th segment ($k \in \{1, \cdots, K\}$) of the video ranges from the $[(k-1)\cdot\frac{T}{K}+1]$-th frame to the $(\frac{kT}{K})$-th frame. A temporally *partial video* or *partial observation* $\mathbf{x}^{(k)}$ is a temporal subsequence that contains the beginning k out of K segments of the video. The *progress level* g of the partial video $\mathbf{x}^{(k)}$ is k: $g = k$, and its *observation ratio* r is $\frac{k}{K}$: $r = \frac{k}{K}$ (see Figure 2). For a given partial video, its progress level (or level) g and observation ratio r have $g = r \times K$.

3.2 Deep Sequential Context Network

Given N training videos $\{\mathbf{x}_i, y_i\}_{i=1}^N$, DeepSCN simulates sequential data arrival, and temporally decompose each training video $\mathbf{x}_i$ into partial observations $\{\mathbf{x}_i^{(k)}\}|_{k=1}^K$ at various progress levels. Note that $\mathbf{x}_i^K$ and $\mathbf{x}_i$ are the same full video: $\mathbf{x}_i^K = \mathbf{x}_i$. DeepSCN learns a feature mapping function $G : \mathbf{x}^{(k)} \rightarrow \mathbf{z}$ and a prediction function $F : \mathbf{z} \rightarrow y$, where $\mathbf{x}^{(k)} \in \mathcal{R}^d$ is a partial video at progress level k, $\mathbf{z} \in \mathcal{R}^D$ is the learned feature vector with high discriminative power, and $y \in \mathcal{Y}$ is the action label.

Sequential context is one of the major components in DeepSCN. Its key idea is to improve the discriminative power of partial videos by gaining extra information from full videos. The assumption is that if the features from a partial video can be geometrically close to the features from the full video, then their discriminative abilities would be similar. DeepSCN defines the discrepancy between partial observations $\{\mathbf{x}_i^{(k)}\}$ and their corresponding full observations $\{\mathbf{x}_i^{(K)}\}$ as

$$\sum_{i=1}^{N}\sum_{k=1}^{K} \|\mathbf{x}_i^{(K)} - \mathbf{W}\mathbf{x}_i^{(k)}\|_2^2 = \|\bar{\mathbf{X}}^{(K)} - \mathbf{W}\bar{\mathbf{X}}\|_F^2, \quad (1)$$

where $\mathbf{W}$ is a feature transformation matrix of size $d \times d$ learned during model training, and $\|\cdot\|_F$ is the Frobenius norm. Matrices $\bar{\mathbf{X}}^{(K)}$ is a $d \times KN$ matrix containing all the full observations and $\bar{\mathbf{X}}$ is also a $d \times KN$ matrix containing all the partial observations:

$$\begin{aligned} \bar{\mathbf{X}}^{(K)} &= (\underbrace{\mathbf{x}_1^{(K)}, \cdots, \mathbf{x}_1^{(K)}}_{K \text{ times}}, \cdots, \mathbf{x}_N^{(K)}, \cdots, \mathbf{x}_N^{(K)}), \\ \bar{\mathbf{X}} &= (\mathbf{x}_1^{(1)}, \cdots, \mathbf{x}_1^{(K)}, \cdots, \mathbf{x}_N^{(1)}, \cdots, x_N^{(K)}). \end{aligned} \quad (2)$$

By minimizing the discrepancy defined in Eq. (1), a partial observation $\mathbf{x}_i^{(k)}$ is mapped onto a feature space using the learned projection matrix $\mathbf{W}$ under the guidance of its corresponding full observation $\mathbf{x}_i^K$ or $\mathbf{x}_i$. The reconstructed feature $\mathbf{W}\mathbf{x}_i^{(k)}$ is expected to be geometrically closer to its corresponding full observation $\mathbf{x}_i^K$. Therefore, the learned feature vector $\mathbf{W}\mathbf{x}_i^{(k)}$ will gain extra crucial information for action prediction from the full observation $\mathbf{x}_i^{(K)}$, and its discriminative power is thus enhanced.

Robust features. During information transfer, noise could be introduced to partial observations, which may degrade the prediction performance. DeepSCN overcomes this problem by regularizing $\mathbf{W}$ and constructing robust features for partial videos that are insensitive to noise. Recent work in robust feature learning [34], [35] shows that robust features should be able to be reconstructed from partial and random corruption. Inspired by this idea, DeepSCN reconstructs features of partial observations with the mapping matrix $\mathbf{W}$:

$$\sum_{i=1}^{N}\sum_{k=1}^{K} \|\mathbf{x}_i^{(k)} - \mathbf{W}\tilde{\mathbf{x}}_i^{(k)}\|_2^2 = \|\bar{\mathbf{X}} - \mathbf{W}\tilde{\mathbf{X}}\|_F^2, \quad (3)$$

where $\tilde{\mathbf{x}}_i^{(k)}$ is the corrupted version of the original data $\mathbf{x}_i^{(k)}$ obtained by setting a fraction of the feature vector $\mathbf{x}_i^{(k)}$ to 0 with probability $p \geqslant 0$. Matrix $\tilde{\mathbf{X}}$ is the corrupted version of $\bar{\mathbf{X}}$ defined as $\tilde{\mathbf{X}} = (\tilde{\mathbf{x}}_1^{(1)}, \cdots, \tilde{\mathbf{x}}_1^{(K)}, \cdots, \tilde{\mathbf{x}}_N^{(1)}, \cdots, \tilde{\mathbf{x}}_N^{(K)})$. To reduce data variance, "infinite" passes of corruptions are performed over the training data [34].

Label information. Partial observations in the same category may vary greatly in appearance, duration, etc. Therefore, the learned prediction model may not be able to capture complex classification boundaries. DeepSCN addresses this problem by incorporating label information to

our feature learner, and expecting the learned features of partial observations at the same progress level in the same category to be geometrically close to each other. The within-class within-progress-level variance is defined in order to regularize the learning of parameter matrix $\mathbf{W}$:

$$\Psi(\mathbf{W}) = \frac{1}{2}\sum_{k=1}^{K}\sum_{i,j=1}^{N} a_{ij}\|\mathbf{W}\mathbf{x}_i^{(k)} - \mathbf{W}\mathbf{x}_j^{(k)}\|_2^2 \quad (4)$$

where $\mathbf{X}^{(k)} = (\mathbf{x}_1^{(k)}, \cdots, \mathbf{x}_N^{(k)}) \in \mathcal{R}^{d\times N}$, and the (i,j)-th element a_{ij} is 1 if $y_i = y_j$ and $i \neq j$; and 0 otherwise.

Putting Eq. (1), Eq. (3), and Eq. (4) together, optimal parameter matrix $\mathbf{W}$ can be learned by

$$\begin{aligned} &\min_{\mathbf{W}} \|\bar{\mathbf{X}} - \mathbf{W}\tilde{\mathbf{X}}\|_F^2 + \alpha\|\bar{\mathbf{X}}^{(K)} - \mathbf{W}\mathbf{X}\|_F^2 + \beta\Psi(\mathbf{W}), \\ &\text{s.t. } \Delta^{(k+1)} \leqslant \Delta^{(k)}, k = 1, \cdots, K-1, \end{aligned} \quad (5)$$

where α and β are trade-off parameters balancing the importance of the corresponding terms, and $\Delta^{(k)}$ is the reconstruction error $\Delta^{(k)} = \|\mathbf{X}^{(K)} - \mathbf{W}\mathbf{X}^{(k)}\|_F^2$. As progress level k increases, the partial video feature $\mathbf{x}^{(k)}$ is geometrically approaching the corresponding full video $\mathbf{x}^{(K)}$. Consequently, the amount of information transferred from the full observation should be decreasing. Such prior knowledge is incorporated using the constraints in optimization problem (5). These constraints also implicitly capture temporal ordering information of inhomogeneous temporal units. Support vector machine is trained independently from DeepSCN to classify actions.

3.3 Discussion

One limitation of DeepSCN is that the features are learned independently from the classifiers. DeepSCN learns features by minimizing the reconstruction error which cannot be used to measure the discrimination power of the learned features. Even though it uses label information to separate features in different action categories, it may not work well for data under complex distributions. Therefore, the features may not be particularly optimized for action prediction.

In addition, DeepSCN does not verify the distribution discrepancy between the generated features and the true features directly extracted from full videos. Although the generated features can be geometrically close to the true features from full videos, their distributions can be significantly different, thereby affecting the prediction performance.

Furthermore, DeepSCN uses a unified encoder-decoder, which may not be powerful enough for encoding complex action videos. Consequently, the generated features are not expressive for describing actions with large appearance and pose variations.

4 Adversarial Action Prediction Network

In this section, we present a novel Adversarial Action Prediction Network (AAPNet) to address the aforementioned problems. AAPNet is generic and compatible with both deep features (e.g., C3D features [36]) and handcrafted features (e.g., spatiotemporal interest points [37] and dense trajectory features [14]). In this work, the input $\mathbf{x}^{(k)}$ to our approach is a partial video represented by a feature vector. This feature vector can be obtained by performing 3D convolutions over the partial video in C3D network, or by the bag-of-words model over spatiotemporal interest points and dense trajectories. The features we use for datasets will be described in the experiments.

4.1 Network Architecture

The proposed AAPNet extends DeepSCN by redefining some of the key components in DeepSCN. AAPNet inherits the loss function in Eq. (5), and utilizes adversarial learning scheme [3], [4] to learn more discriminative features for action prediction. We expect to learn a "representative" and "discriminative" feature for partial video $\mathbf{x}^{(k)}$. Here, a "representative" feature means the learned feature for a partial video should be geometrically close to the feature of its corresponding full video so that the learned feature would encode most of the information that is contained in the full video. In addition, the learned feature should also be "discriminative" for action classification in order to achieve high classification accuracy at the early stage of a video observation.

We use a probabilistic framework to describe the idea of feature generation in AAPNet. In a nut shell, we would like to generate the feature of a full video $\hat{\mathbf{x}}^{(K)}$ from a partial video feature $\mathbf{x}^{(k)}$: $p(\hat{\mathbf{x}}^{(K)}|\mathbf{x}^{(k)})$. However, directly solving this problem is very challenging due to the large appearance variations in the full video and randomness in the unobserved portion in the full video. Instead, we utilize a latent variable $\mathbf{z}$ in this work, which alleviates this problem by summarizing all the possibilities in the full videos in the training data. Using the latent variable $\mathbf{z}$, the full video feature $\hat{\mathbf{x}}^{(K)}$ can be generated by

$$\begin{aligned} p(\hat{\mathbf{x}}^{(K)}|\mathbf{x}^{(k)}) &= \int_{\mathbf{z}} p(\hat{\mathbf{x}}^{(K)}, \mathbf{z}|\mathbf{x}^{(k)})\, d\mathbf{z} \\ &= \int_{\mathbf{z}} p(\hat{\mathbf{x}}^{(K)}|\mathbf{z}) p(\mathbf{z}|\mathbf{x}^{(k)})\, d\mathbf{z} \end{aligned} \quad (6)$$

AAPNet is proposed to achieve this goal. As shown in Figure 3, the proposed AAPNet contains four major components, an encoder E, a discriminator D, and two decoders G_1 and G_2. Given a partial video $\mathbf{x}^{(k)}$ at progress level k, a convolutional neural network is first employed to extract features from frames in the partial video. The extracted frame features are averaged and then fed into the proposed AAPNet. The encoder E in the AAPNet maps partial video features $\mathbf{x}^{(k)}$ into a progress-invariant feature vector $\mathbf{z}$: $\mathbf{z} = E(\mathbf{x}^{(k)})$. The feature $\mathbf{z}$ is expected to encode shared information that appears in all progress levels. The partial video feature $\hat{\mathbf{x}}^{(k)}$ can be estimated from the latent feature $\mathbf{z}$ using the decoder G_1, conditioned on its progress level k: $\hat{\mathbf{x}}^{(k)} = G_1(\mathbf{z}, k)$. The encoder E and decoder G_1 form an autoencoder for partial observations. To reconstruct full observation $\hat{\mathbf{x}}^{(K)}$, we feed the latent feature $\mathbf{z}$ and the expected progress level K into another decoder G_2: $\hat{\mathbf{x}}^{(K)} = G_2(\mathbf{z}, K)$, and then use a discriminator to tell if the generated feature $\hat{\mathbf{x}}^{(K)}$ comes from multi-class data distribution or not.

Compared with other GAN-like networks, we incorporate an encoder to avoid random sampling of latent features $\mathbf{z}$. As videos in the same class have large appearance variations and camera motions, it would be challenging to

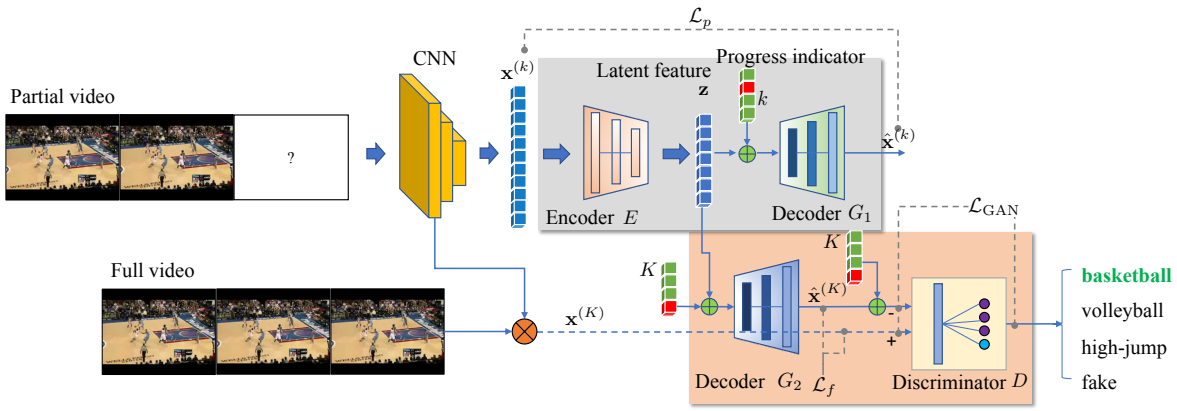


Fig. 3. Framework of the proposed adversarial action prediction network. AAPNet contains four major components, an encoder E, a discriminator D, and two decoders G_1 and G_2. The network learns sequential context information from full videos, and transfer it to the features extracted from partial videos, thereby making the learned features more representative and discriminative.

sample a video data from a simple distribution such as Gaussian distribution. In this work, we use an autoencoder E and G_1 to better capture specific dynamic characteristics of an action in the latent feature $\mathbf{z}$. Consequently, the latent feature $\mathbf{z}$ incorporates rich prior information of the partial video that can be used to generate the corresponding full video. The encoding function E of the autoencoder defines an aggregated posterior distribution of $q(\mathbf{z})$ on the hidden code vector of the autoencoder [3] as follows:

$$p(\mathbf{z}) = \int_{\mathbf{x}^{(k)}} p(\mathbf{z}|\mathbf{x}^{(k)}) p_d(\mathbf{x}^{(k)})\, d\mathbf{x}^{(k)}, \tag{7}$$

where $p(\mathbf{z}|\mathbf{x}^{(k)})$ is the encoding distribution and $p_d(\mathbf{x}^{(k)})$ is the data distribution. The aggregated posterior $p(\mathbf{z})$ encodes the information of the partial video that is necessary for generating full video features.

The use of two decoders G_1 and G_2 allows us to learn progress-invariant feature $\mathbf{z}$ from partial videos. Guided by the two decoders, the latent feature $\mathbf{z}$ needs to balance the information encoded in it in order to minimize the losses. This would regularize the latent feature $\mathbf{z}$ to encode most of the information shared between partial and full videos, making it progress invariant.

4.2 Network Components

Encoder E takes a partial observation $\mathbf{x}^{(k)}$ as an input. The input $\mathbf{x}^{(k)}$ is essentially a feature vector obtained by performing C3D [36] on temporal segments followed by average pooling over segment features, or by using bag-of-words model on top of interest points [37] and dense trajectories [16]. Inspired by [2], we first compute hidden layers $\mathbf{h}$ from input $\mathbf{x}^{(k)}$ using a fully-connected layer, and then compute the mean μ and variance σ^2 from the final hidden layer $\mathbf{h}$ to compute the latent feature vector $\mathbf{z}$ using the reparameterization trick:

$$\mathbf{h} = \phi(\text{fc}_1(\mathbf{x}^{(k)})) \tag{8}$$

$$\mathbf{z} = \mu + \boldsymbol{\sigma^2} \odot \epsilon, \text{where } \mu = \phi(\text{fc}_2(\mathbf{h})), \boldsymbol{\sigma^2} = \phi(\text{fc}_3(\mathbf{h})) \tag{9}$$

Here, $\phi(\cdot)$ denotes an activation function, which is a leaky version of a rectified linear unit in this paper. $\text{fc}(\cdot)$ defines a fully-connected layer. μ, σ denote the parameters of the approximate posterior, and $\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$. Eq. (8) defines a multi-layer perceptron (MLP) with one hidden layer, and it can be easily extended to multiple layers.

Partial Observation Decoder G_1 decodes the partial observation $\hat{\mathbf{x}}^{(k)}$ from the input latent feature vector $\mathbf{z}$ and the progress level k of the partial observation. We define the decoder G_1 as a MLP with hidden layers, and use sigmoid function as the activation function. The encoder E and the decoder G_1 comprise a partial observation autoencoder. Its takes a partial video feature vector $\mathbf{x}^{(k)}$ as input, outputs the reconstructed feature vector $\hat{\mathbf{x}}^{(k)}$ through an intermediate latent feature vector $\mathbf{z}$.

Full Observation Decoder G_2 translates the input latent feature vector $\mathbf{z}$ and the progress level K of the partial observation to the full observation $\hat{\mathbf{x}}^{(K)}$. Similar to decoder G_1, we also define the decoder G_2 as a MLP with sigmoid function as the activation function.

The encoder E and the full observation decoder G_2 constitute a full observation autoencoder, which aims at reconstructing full observations from partial observations. The two autoencoders learn a feature space that is shared by partial observations and full observations. Using the decoders G_1 and G_2, the latent feature vector $\mathbf{z}$ is optimized to reconstruct both partial observations and full observations. Consequently, it is regularized to extract progress-invariant information from videos. The decoder G_1 provides a prior information for generating full observations. Without the decoder G_1, shared information between partial and full observations may not able to be learned, and thus the features of full observations may not be able to be learned.

Discriminator D is basically used to differentiate the full observation $\hat{\mathbf{x}}^{(K)}$ generated by G_2 from the true full observation $\mathbf{x}^{(K)}$, conditioned on the progress level K of the full observation. We define $|Y|+1$ classes in this discriminator D, where $|Y|$ is the number of actions in a dataset, and the extra one category is reserved for a fake category. If the discriminator D categorizes the given input as "real", then it classifies its action using the first $|Y|$ classes $D_{1:|Y|}$; otherwise, it categorizes the input as "fake", which is the extra one category. Note that the first $|Y|$ classes $D_{1:|Y|}$ in the discriminator D are also used to predict actions in testing. This will be discussed in Section 4.5.

Discriminator D enable us to learn a better representation $\hat{\mathbf{x}}^{(K)}$ from a partial observation $\mathbf{x}^{(k)}$ so that it encodes all the information in the full observation $\mathbf{x}^{(K)}$ and cannot be distinguished from the full observation $\mathbf{x}^{(K)}$. In addition, the multi-class classifiers further improve the discrimination power of the features and make them particularly optimized for action prediction. In this work, we use a fully-connected layer without hidden layers in the discriminator D. We do not use hidden layers in here as it generally achieves poor prediction performance in this work.

4.3 Loss Function

Adversarial learning [4] is a popular learning formulation for deep learning. It creates a competition game, in which a discriminator determines a sample is real (from data distribution) or fake (from model distribution), while a generator tries to produce fake samples without being detected. This competition game will finally generate indistinguishable samples. Inspired by this idea, using an appropriate loss function, we can encourage the distributions of the features from partial video and from full video to be similar, and thus improve the prediction performance.

Adversarial loss $\mathcal{L}_{\text{GAN}}$ is optimized to train the encoder E, the decoder G_2 and the discriminator D. The encoder E is trained to generate progress-invariant feature $\mathbf{z}$, and the decoder G_2 is trained to generate full video features from the feature $\mathbf{z}$ manipulated by the progress label k. Our discriminator $D \in \mathcal{R}^{|Y|+1}$ is used for action classification with $|Y|$ as the total number of actions in the training set, and the additional dimension for the fake class. Given a video feature vector $\mathbf{x}$ extracted from real (partial or full) videos, the discriminator D aims to infer its action label; while if the feature vector $\hat{\mathbf{x}}$ is generated from the decoder G_2, the discriminator D attempts to classify the generated feature $\hat{\mathbf{x}}$ as fake. We define the adversarial loss $\mathcal{L}_{\text{GAN}}$ as

$$\begin{aligned}\mathcal{L}_{\text{GAN}} =& \mathbb{E}_{\mathbf{x}^{(k)},k\sim p_{\text{data}}(\mathbf{x}^{(k)},k)}[\log D_{1:|Y|}(\hat{\mathbf{x}}^{(k)}) \\ &+ \log D_{|Y|+1}(\mathbf{x}^{(K)})] \\ &+ \mathbb{E}_{\mathbf{x}^{(k)},k\sim p_{\text{data}}(\mathbf{x}^{(k)},k)}[\log(1 - D_{|Y|+1}(G_2(E(\mathbf{x}^{(k)}))))].\end{aligned} \tag{10}$$

Here, $D_{1:|Y|}(\cdot)$ denotes the discriminator for the first $|Y|$ classes used for classifying actions, and $D_{|Y|+1}(\cdot)$ is the discriminator for the $(|Y|+1)$-th fake class.

Full observation reconstruction loss $\mathcal{L}_f$ is inspired by the Sequential Context component in Eq. (1). As shown in [6], [28], it is essential to improve the discriminative power of features extracted from partial observations in order to achieve high prediction performance. This is even more important for predicting the beginning portion of a video since a large number of useful cues for classification are not observed in the early stage of the video. Furthermore, the features extracted from the beginning portion of a video cannot fully convey the information of the entire video.

Intuitively, people are more confident about the action category if more frames are observed. Recent studies [6], [9], [27], [28] show that the best prediction performance is generally made when all the frames are observed. This suggests that full observations contain all the useful information for classification. Motivated by this observation, in this work, we propose to minimize the difference between the real full video feature $\mathbf{x}^{(K)}$ from training data and the fake full video feature $\hat{\mathbf{x}}^{(K)}$ generated from its corresponding partial video feature $\mathbf{x}^{(k)}$. A standard practice in learning the encoder-decoder is to use the Euclidean distance between the input and the generated output, i.e.,

$$\mathcal{L}_f = \mathbb{E}_{\mathbf{x}^{(k)},k\sim p_{\text{data}}(\mathbf{x}^{(k)},k)}\|\mathbf{x}^{(K)} - G_2(E(\mathbf{x}^{(k)}),K)\|_2^2. \tag{11}$$

A partial video observation $\mathbf{x}^{(k)}$ is first encoded onto a latent feature vector $\mathbf{z} = E(\mathbf{x}^{(k)})$, and then decoded by the full video generator G_2 conditioned on the progress level of the expected full video: $G_2(E(\mathbf{x}^{(k)}),K)$. By minimizing the discrepancy defined in Eq. (11), the partial video observation $\mathbf{x}^{(k)}$ is mapped onto the feature space of full videos using the encoder E and the generator G_2. The generated feature $\hat{\mathbf{x}}^{K} = G_2(E(\mathbf{x}^{(k)}),K)$ is expected to be geometrically closer to its corresponding full observation $\mathbf{x}^{(K)}$. Consequently, the generated feature $\hat{\mathbf{x}}^{(K)}$ will gain extra information for action prediction, and thus its discriminative power is enhanced.

Partial observation reconstruction loss $\mathcal{L}_p$ is inspired by the Robust Feature component defined in Eq. (3). In this work, we measure the reconstruction performance for partial observations $\mathbf{x}^{(k)}$ made by the encoder E and the partial observation generator G_1. A standard practice in learning the encoder-decoder is to use the Euclidean distance between the input and the generated output, i.e.,

$$\mathcal{L}_p = \mathbb{E}_{\mathbf{x}^{(k)},k\sim p_{\text{data}}(\mathbf{x}^{(k)},k)}\|\mathbf{x}^{(k)} - G_1(E(\mathbf{x}^{(k)}),k)\|_2^2, \tag{12}$$

where $\mathbf{x}^{(k)}$ is a partial video feature at progress level k. Here, the encoder E maps the input partial video feature $\mathbf{x}^{(k)}$ into a feature vector $\mathbf{z}$, and then the generator G_1 reconstructs the feature $\hat{\mathbf{x}}^{(k)}$ from $\mathbf{z}$.

Eq. (11) measures the reconstruction error for the generated full video feature, while Eq. (12) is used for the generated partial video feature. Similar to Eq. (1) and Eq. (3), minimizing Eq. (11) and Eq. (12) allows us to find a common latent feature space for partial observations and full observations. Given a partial observation $\mathbf{x}^{(k)}$, in Eq. (11), the encoder E projects the feature into a latent feature vector $\mathbf{z}$. Minimizing Eq. (11) and Eq. (12) makes the latent feature $\mathbf{z}$ to be able to reconstruct both the corresponding partial observation $\hat{\mathbf{x}}^{(k)}$ and full observation $\hat{\mathbf{x}}^{(K)}$, under the manipulation of the desired progress level k and K, respectively. Therefore, the latent feature $\mathbf{z}$ lies in the common feature space, and is progress-invariant. Note that Robust Feature component defined in Eq. (3) uses "infinite" passes of corruptions to reduce data variance and learn robust features. In this work, we use Dropout regularization scheme instead.

In summary, the objective function of the proposed network is given by

$$\min_{E,G_1,G_2} \max_{D} \mathcal{L}_{\text{GAN}} + \alpha\mathcal{L}_p + \beta\mathcal{L}_f, \tag{13}$$

where α and β are trade-off parameters for balancing the importance of the corresponding components.

4.4 Model Learning

This section specifies the learning of encoder parameter θ_e, partial video generator parameter θ_{g1}, full video generator parameter θ_{g2}, and discriminator θ_d.

As illustrated in the above section, our training is defined by three loss functions, 1) loss of GAN, $\mathcal{L}_{\text{GAN}}$, loss of partial video reconstruction $\mathcal{L}_p$, loss of full video reconstruction $\mathcal{L}_f$. The key idea behind our generative-adversarial training is to introduce a video generator G_2 that can transfer progress-invariant feature extracted from a partial video into its corresponding full video feature.

We formulate an adversarial learning algorithm that iteratively optimizes the following three objectives:

1. *Update encoder and partial video generator parameters* $\{\theta_e, \theta_{g1}\}$. We minimize the partial video reconstruction loss $\mathcal{L}_p$ to optimize parameters $\{\theta_e, \theta_{g1}\}$. The encoder E and the decoder G_1 are trained to reconstruct partial video features. Meanwhile, the encoder E is also regularized to generate progress-invariant feature as it also needs to generate full video feature, which will be discussed in the following.

2. *Update discriminator parameter* $\{\theta_d\}$, optimize $\mathcal{L}_{GAN}$. We solve the following optimization problem to update parameter $\{\theta_d\}$, $\max_D \mathcal{L}_{\text{GAN}}$

3. *Update encoder and full video generator parameters* $\{\theta_e, \theta_{g2}\}$. We minimize $\min_{G_2} \mathcal{L}_{GAN} + \mathcal{L}_f$.

E, G_1, G_2 and D improve each other during the alternative training process. With D being more powerful in differentiating generated (fake) full video features and real full video features and classifying actions, G_2 strives to produce full-dynamics preserved features to compete with the discriminator D. To achieve this, the encoder E is optimized to generate progress-invariant latent feature $\mathbf{z}$ as it is optimized to minimize the reconstruction loss for both partial videos and full videos. The encoder E and the partial video decoder G_1 are driven to encode dynamics in partial videos into latent feature $\mathbf{z}$. The full video feature decoder then learns how to map the latent feature $\mathbf{z}$ to full video features corresponding to input partial videos.

Multi-scale Temporal Data. Actions appear in various paces and thus introduce temporal intra-class variations. To alleviate this problem, we introduce a mixture of temporal neighboring segments to the training data in order to consider various cases of partial videos with different paces.

Specifically, when training a partial video $\mathbf{x}^{(k)}$ at progress level k, we will take a group of partial videos $\{\mathbf{x}^{(k+\Delta)}\}|_\Delta$ as its training data, where $\Delta \in [-1, +1]$. For example, we will consider partial videos $\{\mathbf{x}^{(k-0.2)}, \mathbf{x}^{(k)}, \mathbf{x}^{(k+0.8)}\}$ as the training data for the training video $\mathbf{x}$ at progress level k. This can be considered as a video augmentation scheme. We expect these additional partial videos can better address intra-class action variations. These partial videos have varied video lengths and small temporal shifts, and we use average pooling to generate a feature vector from these partial videos.

4.5 Action Prediction

The flowchart for action prediction is shown in Figure 4. Given a testing video with an unknown progress level k and an unknown action label y, a feature extraction method is first employed on this video and represent it as a feature vector. For example, we can use C3D method [36] to extract a group of features for temporal segments, and then use average pooling to generate a feature vector $\mathbf{x}$ [2]. The feature vector $\mathbf{x}$ is then fed into the proposed AAPNet to learn more informative features that are contained in the corresponding full video, and in the meanwhile classify its action. The encoder E generates latent progress-invariant feature $\mathbf{z}$ from the input feature $\mathbf{x}$: $\mathbf{z} = E(\mathbf{x})$. Then, the full observation decoder G_2 concatenates the latent progress-invariant feature $\mathbf{z}$ and the target progress level $k = 1.0$ in one-hot format, and produce the generated feature vector $\hat{\mathbf{x}}$, which is considered to be close to the true full observation $\mathbf{x}^{(K)}$. The generated vector $\hat{\mathbf{x}}$ is fed into the discriminator D to generate the scores $D_{1:|Y|}(\hat{\mathbf{x}})$ for each action category. In addition, the input partial video $\mathbf{x}$ is also fed into the discriminator D to generate the scores $D_{1:|Y|}(\mathbf{x})$. The two scores are averaged to provide the score for the test partial video $\mathbf{x}$.

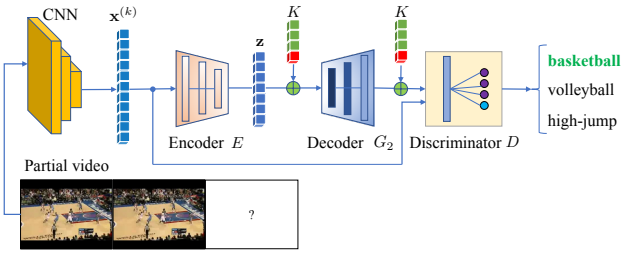


Fig. 4. Flowchart for making prediction for a given testing partial video.

4.6 Comparison with DeepSCN

The proposed AAPNet shares similar ideas to our prior work DeepSCN but uses different components. The key idea of the two methods is to enrich the partial video feature and reconstruct its corresponding full video feature. The major differences are discussed in the following.

Adversarial learning. DeepSCN does not use adversarial learning to align the distributions of the generated features and the true features from full videos; while it is used in AAPNet to verify the distribution discrepancy. In addition, DeepSCN trains SVM separately, while AAPNet trains a multi-class classifier jointly with feature learning, and makes features optimized for action prediction.

Partial video reconstruction loss. DeepSCN uses Eq. (3) to reconstruct partial video feature from a partial video feature, which minimizes the reconstruction loss represented by the Euclidean distance between the original partial video and the generated partial video: $\|\mathbf{x}_i^{(k)} - \mathbf{W}\tilde{\mathbf{x}}_i^{(k)}\|_2^2$. It uses a unified linear mapping as the encoder and decoder $\mathbf{W}$. Similar to this idea, AAPNet also minimizes the Euclidean distance in Eq. (12) between the two: $\mathcal{L}_p = \|\mathbf{x}^{(k)} - G_1(E(\mathbf{x}^{(k)}), k)\|_2^2$. However, instead of using a simple linear mapping as in DeepSCN, we use multiple non-linear mappings in an autoencoder E and G_1 to reconstruct the partial video. This improves the representative power of the reconstructed partial video, and provides a more accurate prior information extracted from the partial video.

Full video reconstruction loss. DeepSCN uses Eq. (1) to reconstruct full video feature from partial video feature.

2. We use $\mathbf{x}$ instead of $\mathbf{x}^{(k)}$ here because the progress level k is unknown in the test video.

Similar to the partial video reconstruction loss discussed above, DeepSCN minimizes the Euclidean distance $\|\mathbf{x}_i^{(K)} - \mathbf{W}\mathbf{x}_i^{(k)}\|_2^2$ to learn a linear mapping $\mathbf{W}$. The proposed AAPNet uses separate encoder and decoder in Eq. (11) to learn the feature mapping: $\mathcal{L}_f = \|\mathbf{x}^{(K)} - G_2(E(\mathbf{x}^{(k)}), K)\|_2^2$. In addition, DeenSCN uses the same mapping $\mathbf{W}$ for reconstructing both partial and full video features, while AAPNet uses different decoders G_1 and G_2 for the two purposes.

Label information. DeepSCN uses a Laplacian matrix in Eq. (4) to regularize hidden features in the same class at the same progress level to be similar. This solution may not be effective for data under more complex distributions because some complex data samples may not be able to be regularized to be close to a simple data sample. By comparison, AAPNet considers label information in a multi-class classifier $D_{1:|Y|}$, and uses the classification loss to guide feature learning. This process makes features particularly optimized for action prediction.

5 Experiments

5.1 Dataset and Experiment Setup

We evaluate our approach on three datasets: UCF101 dataset [38], Sports-1M dataset [39], and BIT-Interaction dataset [18]. UCF101 dataset consists of $13,320$ videos distributed in 101 actions. Sports-1M dataset contains $1,133,158$ videos divided into 487 classes. BIT dataset consists of 8 classes of human interactions, with 50 videos per class. It should be noted that N videos will be $10N$ videos to action prediction approaches due to the modeling of 10 progress levels. This larger volume of data increases the complexity of the prediction problem, and will validate the scalability of our approach. However, a majority of existing action prediction approaches [6], [27], [28] are not able to deal with large datasets as they are not trained in a stochastic fashion. To make a fair comparison, we use the first 50 classes in the Sports-1M datasets, and sample $9,223$ videos. This results in $92,230$ partial videos to prediction approaches.

Our approach works with both deep features and handcrafted features. We extract C3D features [36] from partial videos in UCF101 and Sports-1M dataset as C3D generates features for both segments and full videos. Pre-trained C3D model on Sports-1M dataset is used on UCF101 and Sports-1M datasets. To demonstrate compatibility with handcrafted features, we extract spatiotemporal interest points (STIPs) [37] and dense trajectory features (DTs) [14] from partial videos in BIT dataset. Bag-of-words model (with 500 visual words) is adopted to encode STIPs and DTs features.

We follow the split scheme of [36] for UCF101 and [39] for Sports-1M datasets, respectively. The prediction performance on various splits are averaged and reported in this paper. The first 15 groups of videos in UCF101 are used for training; the next 3 groups for cross-validation; and the remaining 7 groups for testing. We also follow the same experiment settings in [28] for BIT dataset, and use the first 34 videos in each class for training (in total 272 training videos) and use the remaining for testing. The number of hidden layers M in E, G_1, G_2 is 2, default parameters settings are $\alpha = 1, \beta = 1$ on all the three datasets.

5.2 Prediction Performance

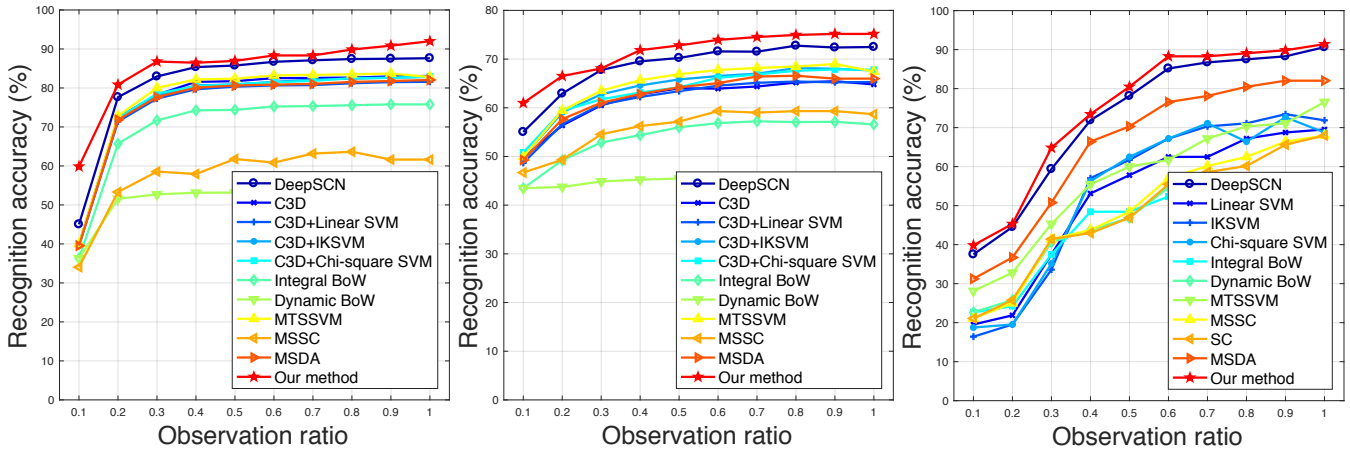
We compare with DeepSCN [1], Dynamic BoW (DBoW) and Integral BoW (IBoW) [27], MSSC and SC [6][3], and MTSSVM [28]. C3D [36] method, SVMs with linear kernel, intersection kernel (IKSVM), chi-square kernel, and marginalized stacked autoencoder (MSDA) [34] are used as baselines. IBOW, DBOW, MTSSVM, and all baselines require the ground-truth progress levels to be known in testing. To perform a fair comparison, the ground-truth progress levels of testing videos are known to all comparison methods, and all the comparison methods on one dataset are fed with the same features. K IKSVMs are adopted for DeepSCN and MSDA, each of which is used for one single progress level, as in [1].

UCF101 dataset. Results in Figure 5(a) show that our method consistently outperforms all the comparison methods. Our method achieves an impressive 59.85% prediction results when only 10% frames are observed, which is 14.83% higher than DeepSCN and 19.80% higher than the MTSSVM. This clearly demonstrates the effectiveness of using the full observation decoder G_2 and the discriminator D in AAPNet. Although an observation with only 10% frames of its full video contains little discriminative information, the discriminator D encourages the full observation decoder G_2 to generate visual features that are similar to features in the full video. This significantly enhances the discriminative power of the partial observation. In addition, AAPNet learns features and classifiers jointly, making the feature representation optimized for action classifiers. The performance of our method at observation ratio 0.3 is already higher than the best performance of all the other comparison methods except DeepSCN. The performance of our AAPNet at observation ratio 0.6 outperforms DeepSCN with full observations. These remarkable results demonstrate the superiority of our method over comparison methods. It should be noted that DBoW achieves extremely low performance on this dataset. This is possibly because its action models computed by averaging features are not expressive enough to capture highly diverse action dynamics in the same category. End-to-end C3D performs slightly better than C3D+SVM because it is end-to-end trained and has certain techniques for improving the classification performance such as non-linear activation ReLU and regularization dropout.

We also analyze the prediction performance of our AAPNet on RGB and flow data. A two-stream network named TSN [40] is used to provide frame feature vectors for RGB and flow videos. As shown in Table 1, C3D+AAPNet performs better than TSN+AAPNet at the beginning portions of videos. C3D uses 3D convolutions to capture information in a longer temporal span than TSN. This is useful for aggregating discriminative information in the temporal domain, especially when the information is limited in the beginning portions of videos. When more frames are observed, TSN performs better than C3D as it captures more descriptive information through its RGB and flow streams. The prediction performance of the flow stream is consistently better than the RGB stream in all the observation ratios.

Sports-1M dataset. Results in Figure 5(b) demonstrate the superiority of our method over all the comparison

3. The code is available at http://www.visioncao.com/publications.html.



(a) UCF101 dataset (b) Sports1M dataset (c) BIT-Interaction dataset

Fig. 5. Prediction results on (a) UCF101, (b) Sports-1M, and (c) BIT dataset. Note that these prediction approaches are optimized for partial videos and thus cannot be directly compared to action recognition approaches given full videos (observation ratio $r = 1.0$). Please refer to the supplemental material for the numbers in the figure.

TABLE 1
Prediction results (%) on UCF101 dataset using C3D, TSN, RGB stream and flow stream in TSN as the CNN component in our architecture. Observation ratios are $r \in \{0.2, 0.4, 0.6, 0.8, 1.0\}$.

Methods	Avg.	0.2	0.4	0.6	0.8	1.0
C3D [36]+AAPNet	85.03	80.85	86.47	88.34	89.85	91.99
TSN [40]+AAPNet	85.06	80.44	86.07	88.34	90.85	92.02
RGB stream [40]+AAPNet	75.38	68.54	76.37	78.91	81.05	84.38
Flow stream [40]+AAPNet	76.76	69.84	78.85	79.96	83.82	85.38

methods. Our AAPNet achieves 60.98% accuracy when only 10% frames are observed, 5.96% higher than DeepSCN, suggesting the superiority of using the discriminator D to differentiate the generated fake full observations and true full observations. This helps to generate more representative features. Note that the accuracy of 60.98% is already higher than MSSC, IBoW, and DBoW with full observations. Our AAPNet achieves an impressive 72.81% when only 50% frames are observed, higher than the best performance in 10 cases of all the other comparison methods. Note that our AAPNet makes accurate predictions at an early stage, demonstrating the effectiveness of its deep architecture, and joint learning of features and classifiers. AAPNet outperforms MSDA, showing the benefits of learning extra information from full videos. Our method consistently outperforms MTSSVM, MSSC, DBoW, and IBoW, suggesting the benefit of learning sequential context information.

It should be noted that the performance of our method and C3D+SVMs methods given full videos cannot be directly compared with the original C3D method [36]. We use all the frames in this work while [36] randomly sampled 5 two-second clips from a video. C3D performs slightly better than C3D+SVM on Sports-1M dataset since C3D is trained in an end-to-end fashion. In addition, non-linear activation ReLU and regularization dropout are also helpful in improving classification accuracy on this large data.

It is interesting to see that there is a huge drop in performance for UCF101 at 10% observation ratio compared to Sports-1M dataset. This indicates that the beginning segments in UCF101 are less discriminative than the ones in Sports-1M. The underlying reason is that the average length of a segment in UCF101 is much shorter (17.6 frames vs over 650 frames) than that of a segment in Sports-1M dataset. Therefore, the beginning segments in UCF101 videos may contain less discriminative information compared to the beginning segments in Sports-1M videos, which causes the performance drop.

BIT dataset. Results in Figure 5(c) show that our method consistently outperforms all the other comparison methods on this small dataset. Our method achieves 39.84% accuracy at observation ratio $r = 0.1$, 2.44% higher than the runner-up DeepSCN method. At observation ratio $r = 0.3$, the prediction performance improvement of our AAPNet over DeepSCN is 5.47%, which is the largest performance gap between these two methods in 10 cases. In 10 cases, AAPNet achieves an average performance improvement of 2.10% over DeepSCN. At $r = 0.6$, our method achieves an impressive result of 88.28%, higher than the best performance of all the other comparison methods except DeepSCN on 10 observation ratios. Our method remarkably outperforms MSSC and SC [6] in all the 10 cases, demonstrating its ability of learning more discriminative features for action prediction. The most noticeable improvement occurs at $r = 0.5$ where the performance increases over MSSC and SC are 32.03% and 33.59%, respectively. Our AAPNet achieves notably higher performance compared with DBoW and IBoW. We achieve 64.84% accuracy with only the first 30% frames of testing videos being observed, which is higher than DBoW and IBoW at all observation ratios. End-to-end C3D does not outperform C3D+SVM possibly because the dropout regularization is not necessary on small the dataset and using it would hurt the performance.

5.3 Running Time

The total training and testing time of the proposed AAPNet, DeepSCN [1], MTSSVM [28] and MSSC [6] are summarized in Table 2. This may not be a fair comparison because these methods are implemented in different programming languages; AAPNet is implemented in Python and the other

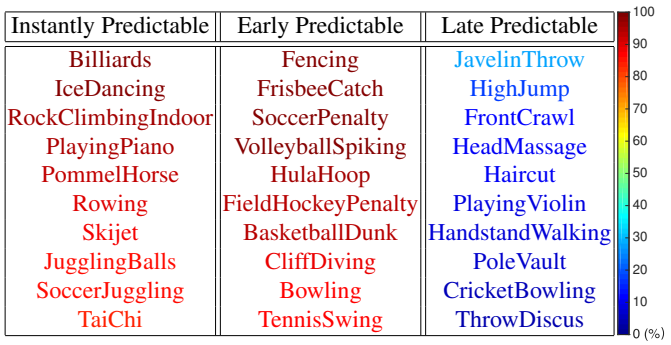


Fig. 6. Top 10 instantly, early, and late predictable actions in UCF101 dataset. Action names are colored and sorted according to the percentage of their testing samples falling in the category of IP, EP, or LP. This figure is best viewed in color.

three methods are implemented in MATLAB. On a 3.4GHz CPU, our AAPNet spends 1 hour in training and testing, while DeepSCN uses 4 hours. MTSSVM and MSSC use 140 hours and 420 hours, respectively, which are two orders of magnitude slower than AAPNet and DeepSCN. It takes our AAPNet about 0.5 hours in training and testing videos in Sports-1M dataset. By comparison, DeepSCN, MTSSVM, and MSSC use 2.5, 50, and 770 hours, respectively, remarkably slower than our AAPNet method. On BIT dataset, our AAPNet is slower than DeepSCN because it spends lots of time in data loading, but it is still faster than MTSSVM and MSSC on this dataset.

AAPNet also runs on GPU and it is much faster than running on a CPU. On a Titan XP GPU, AAPNet spends 0.07 hour in training and testing UCF101 dataset, which is 14× faster than running on a CPU. On Sports-1M dataset and BIT, the speedup is 16× and 3.5×, respectively. The most time-consuming component in our architecture is feature extraction using C3D.

TABLE 2
Training and testing time (hours) of comparison methods on UCF101, Sports-1M, and BIT datasets.

Methods	CPU/GPU	UCF101	Sports-1M	BIT
MTSSVM [28]	CPU	140h	50h	0.12h
MSSC [6]	CPU	420h	770h	0.2h
DeepSCN [1]	CPU	4h	2.5h	0.002h
Ours	CPU	1h	0.5h	0.007h
Ours	GPU	0.07h	0.03h	0.002h

5.4 Instantly, Early, and Late Predictable Actions

It should be noted that actions differ in their prediction characteristics. Discriminative patterns of actions may appear early or late in an action video. This affects the portion of a video that needs to be observed before being classified correctly, i.e., the *predictability* of an action. We analyze the predictability of actions in three datasets, and study at what stage an action can be predicted. We define three categories of action videos according to their predictability: *instantly predictable* (IP), *early predictable* (EP), and *late predictable* (LP). An action video is IP means that the video can be predicted after only observing the beginning 10% portion of the video.

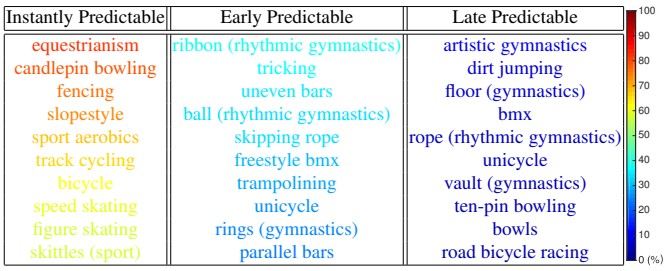


Fig. 7. Top 10 instantly, early, and late predictable actions in Sports1M dataset. Action names are colored and sorted according to the percentage of their testing samples that fall in the category of IP, EP, or LP. For example, "artistic gymnastics" has 12% testing samples that are late predictable (require to observe more than 50% video frames in order to make accurate predictions). This figure is best viewed in color.

EP means that an action video is not IP but can be predicted if the beginning 50% portion of the video is observed. LP means that an action video is neither IP nor EP, and can only be predicted if more than 50% portion of the video is observed.

5.4.1 UCF101 Dataset

Top 10 IP, EP, and LP actions in UCF101 dataset are listed in Figure 6. Results show that actions "Billiards" and "IceDancing" are the easiest to predict; all of their testing samples are instantly predictable. In our experiment, there are 33 action categories having over 50% of their respective testing videos instantly predictable (correctly classified after only observing the beginning 10% frames). Figure 6 also shows that 4 actions have all their testing samples early predictable. In fact, there are 38 actions out of 101 actions having over 50% of their respective testing videos that are early predictable (less than 50% video frames need to be observed). The action "JavelinThrow" can be considered as the most challenging class to predict as 29% of its testing samples are late predictable (more than 50% video frames need to be observed), higher than all the other actions. In all the 37,830 testing partial videos, 35.45% of them are instantly predictable, and 43.78% are early predictable; only 2.09% are late predictable. The remaining 18.69% partial videos cannot be correctly predicted. This suggests that a majority of action videos can be correctly classified using our approach after observing the beginning 50% frames of the videos. Please refer to the supplement for action predicability results on RGB data and flow data.

It is interesting to see that there are 11 of the 24 classes in UCF101-24 dataset used in [41] falling in the category of IP, which are "Biking", "SkyDiving", "IceDancing", "RopeClimbing", "SalsaSpin", "Skiing", "Skijet", "SoccerJuggling", "Surfing", "TrampolineJumping", "WalkingWithDog"[4]. This explains that [41] has rather flat progression for action prediction accuracy. In addition, [41] makes predictions based on the localized person, while our method does not. Localization of the person help reduce a large amount of background noise, and thus improves the performance.

5.4.2 Sports-1M Dataset

Top 10 IP, EP, and LP actions in Sports-1M dataset are listed in Figure 7. Results show that actions "equestrianism"

4. Please note that Fig. 6 is not a complete list of instantly predictable actions and thus some of the 10 actions here are not shown in Fig. 6.

is the easiest action to predict among all the 50 actions; 83% of their testing samples can be instantly predicted. In our experiment, there are 15 action categories having over 50% of their respective testing videos instantly predictable (correctly classified after only observing the beginning 10% of the frames). Figure 7 also shows that 39% testing samples of "ribbon (rhythmic gymnastics)" and 25% testing samples of "parallel bars" are early predictable. The action "artistic gymnastics" can be considered as the most challenging class to predict as 12% of its testing samples are late predictable (require more than 50% video frames to be observed in order to make accurate prediction), higher than all the other actions. In all the 18610 testing partial videos, 42.24% testing samples are instantly predictable, and 23.16% are early predictable; only 3.22% are late predictable. The remaining 31.38% testing samples cannot be correctly classified. This suggests that a majority of action videos can be correctly classified using our approach after observing the beginning half of videos.

5.4.3 BIT Dataet

The distributions of actions on BIT dataset in IP, EP, and LP categories are listed in Figure 8. It should be noted that an action (e.g.,"pat" and "handshake") may appear in IP, EP, and LP at the same time as some of its testing samples are IP, but some of them are LP. This is because the discriminative patterns in an action appear in different stages of videos. For example, in some "pat" videos, hand motion occurs instantly while in other video samples the motion occurs at the end of the videos. Action "bow" contains 44% videos that are IP, significantly higher than the other 6 actions listed in IP. Therefore, "bow" can be considered as the easiest action to predict. By comparison, action "boxing" has no samples that are instantly predictable (and thus not listed in IP). Note that the percentage of 8 actions falling in IP in BIT dataset is significantly lower than the percentage in UCF101 and Sports-1M datasets. The underlying reason is that in BIT dataset people behave similarly in the beginning stage in different actions (they tend to be standing still), which may not be easily differentiated. Actions "hug" and "high-five" have 88% of their testing samples that are early predictable. The other 5 actions (excluding "pat") all have over 50% of their respective testing samples that are early predictable. This suggests that a majority of action videos can be correctly classified after observing the beginning half of the videos. Action "pat" can be considered as the most challenging one to predict as 19% of its testing samples are late predictable, higher than all the other actions. In all the 1280 testing partial videos, 13.28% are instantly predictable, 55% are early predictable, 4.69% are late predictable, and 27.03% testing samples cannot be correctly classified.

5.5 Unknown vs. Known Progress Level

In practical scenarios, progress levels of testing partial videos are unknown. Nevertheless, previous work in [1], [42] shows that the availability of the progress levels of testing partial videos has an impact on the performance of action prediction methods. In this experiment, we are interested in such an impact, and evaluate the performance variation of our AAPNet in two scenarios where progress

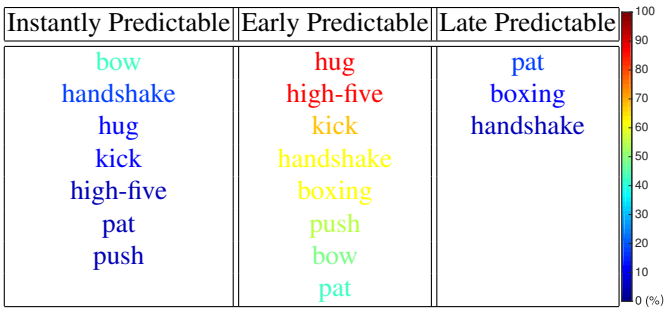


Fig. 8. Instantly, early, and late predictable actions in BIT dataset. Action names are colored and sorted according to the percentage of their testing samples that fall in the category of IP, EP, or LP. For example, "pat" has 19% testing samples that are late predicable (require to observe more than 50% video frames in order to make accurate predictions). This is higher than all the other actions, and thus "pat" is the most challenging action for prediction. This figure is best viewed in color.

levels are known and unknown in testing. *Scenario 1:* the progress levels are available, we train K AAPNets for action prediction, where the k-th AAPNet corresponds to partial observations at progress level k. In testing, the ground-truth progress level k of a testing video $\mathbf{x}$ is required to choose the k-th AAPNet to make predictions. This is referred to as the TRUE method. *Scenario 2:* the progress levels are unavailable, which is practical in real-world applications. We train one single AAPNet for prediction. In testing, all the partial videos are treated to be at the same progress level. We call it the ONE method.

Performance variations of the two methods on UCF101, Sports-1M, and BIT datasets are shown in Table 3. Results show that the average performance variation between the TRUE method and the ONE method is within 0.5% on UCF101 and Sports-1M datasets, and it is within 1.5% on BIT dataset. This demonstrates that ONE method can be used in practical scenarios without significant performance decrease where the progress levels are unknown. Thanks to the proposed AAPNet, partial videos at various progress levels can be accurately represented, thereby making one AAPNet powerful enough for predicting these partial videos and making the progress levels unnecessary in testing. In addition, training ONE method is significantly faster than training TRUE method as ONE method only trains one AAPNet while TRUE method needs to train K AAPNets. The performance variation on BIT dataset is relatively larger than UCF101 and Sports-1M datasets because short video clips (most of the videos are less than 100 frames) and non-cyclic actions (such as "push" and "handshake") are present in the dataset. Using inaccurate progress levels in testing would confuse action predictors, and thus decreases the performance.

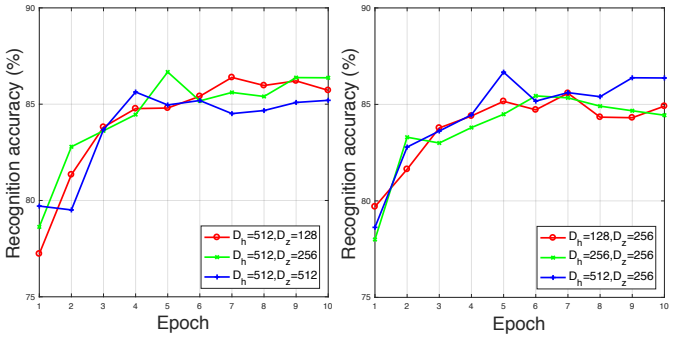
5.6 Convergence

We analyze the convergence speed of the proposed AAPNet in training, and show results in Figure 9. We set the dimensionality D_h of the hidden features $\mathbf{h}$ and the dimensionality D_z of the latent feature $\mathbf{z}$ to $128, 256, 512$ to see their convergence differences. We train on UCF101 dataset (split 1) 10 times, and report the average performance.

Results in Figure 9(a) indicate that the three methods converge after training for 6 epochs. After 6 epochs, the

TABLE 3
Prediction results (%) on UCF101, Sports-1M and BIT datasets using deep networks methods. Observation ratios $r \in \{0.1, 0.3, 0.5, 0.7, 1.0\}$. The average performance is computed over all 10 observation ratios.

	UCF101						Sports-1M						BIT					
	avg.	0.1	0.3	0.5	0.7	1.0	avg.	0.1	0.3	0.5	0.7	1.0	avg.	0.1	0.3	0.5	0.7	1.0
TRUE	85.52	60.32	86.94	86.94	89.48	92.62	71.40	60.24	68.35	72.54	74.48	75.93	76.56	41.41	68.75	78.91	88.28	92.97
ONE	85.03	59.84	86.78	86.65	88.34	91.99	71.40	60.98	68.08	72.81	74.53	75.17	75.08	39.84	64.84	80.47	88.28	91.40



(a) (b)

Fig. 9. Average prediction performance of our AAPNet in different training epochs. (a) The dimensionality D_z of the progress-invariant feature $\mathbf{z}$ is set to $128, 256, 512$. (b) The dimensionality D_h of the hidden feature $\mathbf{h}$ is set to $128, 256, 512$. The proposed AAPNet generally converges after 5 epochs, and its performance variation is within 1.2%.

prediction performance variations for the methods with $D_z = 128, 256, 512$ are $0.98\%, 1.21\%, 0.69\%$, respectively. Therefore, we do not need to train AAPNet for hundreds of epochs as other deep networks did, and can save a lot of training time. Similar convergence result can also be found in Figure 9(b). After 6 training epochs, the prediction performance variations of the three methods with $D_h = 128, 256, 512$ are $1.27, 1.00, 1.21$, which are all within 1.5%. Therefore, we only train AAPNet for 10 epochs throughout this work as more training epochs do not necessarily increase the prediction performance.

5.7 Effectiveness of Components and Parameters

We evaluate the effectiveness of model components in our method, and the sensitivity to the parameters α and β on UCF101 dataset. The sensitivity to the number of layers M can be found in the supplemental material.

5.7.1 Components.

AAPNet has four major components, including the encoder E, the partial observation decoder G_1, the full observation decoder G_2, and the discriminator D. To validate their effectiveness, we remove each of them from the full network and create four variants, which are no-E method, no-D method, no-G_1 method, and no-G_2 method. The averaged prediction results over 10 observation ratios and prediction results over observation ratios $0.1, 0.3, 0.5, 0.7, 1.0$ are summarized in Table 4. C3D method is utilized as a baseline.

Our method significantly outperforms the no-D method by 6.83% on average, demonstrating the effectiveness of adversarial learning to differentiate the generated full observations and the true full observations. This module encourages the network to generate full observations that are similar to the true full observations, and thereby improving the performance. The performance gap between our method and the no-G_2 method shows the importance of generating full observations from the latent feature $\mathbf{z}$ using G_2 as G_2 can provide additional information to the latent feature $\mathbf{z}$ and make the generated features closer to the features in the corresponding full observation. The variant method no-G_1 does not allow us to learn a shared feature space between partial observations and full observations. Although we can still reconstruct a full observation by minimizing the full observation reconstruction loss, without G_1, we are not able to regularize the latent feature $\mathbf{z}$ to contain the prior information in the partial observation. Therefore, the network is not capable of providing the partial observations with the correct information from its corresponding full observation. The strength of the encoder E can be seen from the performance variance between our method and the no-E method. The encoder summarizes partial observations of various observation ratios into a latent feature vector $\mathbf{z}$, which is beneficial for encoding actions of variable lengths.

TABLE 4
Comparison experiments among variants on partial videos of observation ratios $r \in \{0.1, 0.3, 0.5, 0.7, 1.0\}$. The average performance is computed over all 10 observation ratios.

Methods	avg.	0.1	0.3	0.5	0.7	1.0
C3D	76.35	39.99	78.56	80.97	81.81	82.92
no-D	78.20	51.18	78.09	78.19	79.62	83.35
no-G_2	80.93	56.09	82.82	83.06	84.51	88.18
no-G_1	81.88	56.17	82.39	83.19	85.96	88.84
no-E	81.94	56.49	83.32	83.48	84.96	88.61
Ours	85.03	59.84	86.78	86.65	88.34	91.99

5.7.2 Parameters α and β

The sensitivity of our method to parameter α and β is reported in Table 5. Parameters α and β are set to $0.001, 0.01, 0.1, 1, 10, 100$, respectively. Results in Table 5(a) indicate that our method is not sensitive to parameter $\alpha \geqslant 0.01$. The largest performance variation is only within 0.8%. The average performance slightly drops to 84.47% if $\alpha = 0.001$ as the learned model parameter will pay less emphasis on learning robust features from partial observations themselves. This may generate less robust features for prediction, and thus degrade the performance slightly.

Table 5(b) shows that our method is also not sensitive to the parameter β. The largest average performance gap is within 0.9%. The insensitivity of our method to parameter β significantly saves time in parameter tuning.

TABLE 5
Average prediction performance (%) of our method on UCF101 dataset with various parameter α and β values.

(a) Performance with various α.

α	0.001	0.01	0.1	1	10	100
Acc.	84.47	85.14	84.76	85.03	84.86	85.56

(b) Performance with various β.

β	0.001	0.01	0.1	1	10	100
Acc.	85.41	85.84	85.36	85.03	85.60	85.93

5.8 Fusion Strategy

We also evaluate the performance fusion function $g(\cdot)$ used for integrating the prediction scores given by given test observation $\mathbf{x}$ and the generated observation $\hat{\mathbf{x}}^{(K)}$. $g(D_{1:|Y|}(\mathbf{x}), D_{1:|Y|}(\hat{\mathbf{x}}^{(K)}))$. The fusion function $g(\cdot)$ is defined as mean, max, and weighted average over the two score outputs. Weighted average is defined by $\gamma \times D_{1:|Y|}(\hat{\mathbf{x}}^{(K)}) + (1-\gamma) \times D_{1:|Y|}(\mathbf{x})$. Results in Table 6 show that weight average ($\gamma = 0$) achieves the worst performance compared with other fusion schemes because it does only focuses on the given observation $\mathbf{x}$, and does not use the information transferred from full observations in the generated observations $\hat{\mathbf{x}}^{(K)}$. As the action information in the given observation $\mathbf{x}$ is not complete, the prediction performance is the worst in this comparison experiment. Using a combination of both given observation $\mathbf{x}$ and generated observation $\hat{\mathbf{x}}^{(K)}$ generally achieves high performance in this comparison. The best result is achieved at $\gamma = 0.2$ and similar performance is also achieved if $\gamma = 0.5$ (Average). The scheme $\gamma = 1$ only uses the information provided by the generated observation $\hat{\mathbf{x}}$. Even though the generated observation $\hat{\mathbf{x}}$ capture the information in its corresponding full observation, it may gain irrelevant noise information in the full observation, and thus slightly decrease the performance.

TABLE 6
Accuracy (%) of our method using various fusion strategies for integrating the prediction scores of generated features $\hat{\mathbf{x}}^{(K)}$ and given features $\mathbf{x}$ in test.

#Method	avg.	0.1	0.3	0.5	0.7	1.0
Max	83.21	57.20	83.51	83.27	84.69	89.32
Average	84.93	59.68	86.62	86.94	88.18	91.99
Weighted ($\gamma = 0$)	79.50	54.42	81.26	81.47	82.90	86.65
Weighted ($\gamma = 1$)	84.62	59.47	86.31	86.62	87.95	91.75
Weighted ($\gamma = 0.2$)	84.97	59.79	86.68	87.02	88.29	92.07

5.9 Performance on Action Recognition

Our method is also compared with state-of-the-art action recognition methods including [23], [36], [43], [44], [45], [46], [47]. Results in 7 show that our method outperforms [23], [36], [45], [46] and achieves slightly lower performance compared to [43], [44], [47]. Note that our goal is different from these comparison methods. Our method is optimized for classifying incomplete actions in partial videos, while these methods were developed for recognizing complete actions, assuming the action in each testing video has been fully executed. This makes them unsuitable for predicting action labels in partial videos. Even though the networks in [48], [49] achieve even higher performance, they require pre-training on external datasets such as Kinetics [50].

TABLE 7
Comparison results with action recognition methods.

Method	Accuracy
Two stream [45]	88.0%
C3D [36]	85.2%
Two steam+LSTM [46]	88.6%
TDD+FV [23]	90.3%
KVMF [47]	93.1%
ST-ResNet [44]	94.2%
TVNet [43]	94.5%
Ours	92.0%

6 Conclusion

This work addresses the problem of predicting the action label of a video before the action execution ends. We have proposed an efficient and powerful approach for recognizing unfinished human actions from videos. Our approach learns extra information from fully observed actions to improve the discriminative power of the features from temporally partial observations. We further improve the representation and discrimination power of the features by using an adversarial loss and a classification loss. Our method is evaluated on UCF101, Sports-1M, and BIT-Interaction datasets, and shows significant improvements with considerable speedup over state-of-the-art methods. An interesting finding shows that actions differ in their *predicability*. This inspires us to further explore the temporal structures of actions for prompt and accurate prediction in future work.

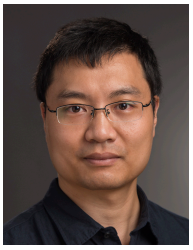
Acknowledgments

This research is supported in part by the NSF IIS Award 1651902 and U.S. Army Research Office Award W911NF-17-1-0367.

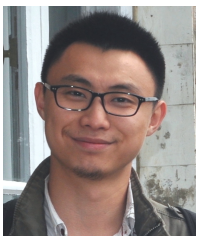
References

[1] Y. Kong, Z. Tao, and Y. Fu, "Deep sequential context networks for action prediction," in *CVPR*, 2017.
[2] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," in *ICLR*, 2014.
[3] A. Makhzani, J. Shlens, N. Jaitly, and I. Goofellow, "Adversarial autoencoders," in *ICLR Workshop*, 2016.
[4] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *NIPS*, 2014, pp. 2672–2680.
[5] K. Li and Y. Fu, "Prediction of human activity by discovering temporal sequence patterns," *TPAMI*, vol. 36, no. 8, pp. 1644 – 1657, 2014.
[6] Y. Cao, D. Barrett, A. Barbu, S. Narayanaswamy, H. Yu, A. Michaux, Y. Lin, S. Dickinson, J. Siskind, and S. Wang, "Recognizing human activities from partially observed videos," in *CVPR*, 2013.
[7] Y. Zhou and T. L. Berg, "Temporal perception and prediction in ego-centric video," in *ICCV*, 2015.
[8] M. Pei, Y. Jia, and S.-C. Zhu, "Parsing video events with goal inference and intent prediction," in *ICCV*, 2011.
[9] T. Lan, T.-C. Chen, and S. Savarese, "A hierarchical representation for future action prediction," in *ECCV*, 2014.
[10] M. Hoai and F. D. la Torre, "Max-margin early event detectors," in *CVPR*, 2012.
[11] S. Ma, L. Sigal, and S. Sclaroff, "Learning activity progression in lstms for activity detection and early detection," in *CVPR*, 2016.

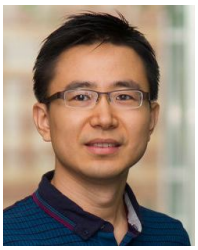
[12] I. Laptev, "On space-time interest points," *IJCV*, vol. 64, no. 2, pp. 107–123, 2005.
[13] B. Wu, C. Yuan, and W. Hu, "Human action recognition based on context-dependent graph kernels," in *CVPR*, 2014.
[14] H. Wang, A. Kläser, C. Schmid, and C.-L. Liu, "Action recognition by dense trajectories," in *CVPR*, 2011, pp. 3169–3176.
[15] M. Raptis and L. Sigal, "Poselet key-framing: A model for human activity recognition," in *CVPR*, 2013.
[16] H. Wang, A. Kläser, C. Schmid, and C.-L. Liu, "Dense trajectories and motion boundary descriptors for action recognition," *IJCV*, vol. 103, no. 1, pp. 60–79, 2013.
[17] B. Ni, P. Moulin, X. Yang, and S. Yan, "Motion part regularization: Improving action recognition via trajectory selection," in *CVPR*, June 2015.
[18] Y. Kong, Y. Jia, and Y. Fu, "Interactive phrases: Semantic descriptions for human interaction recognition," in *TPAMI*, vol. 36, no. 9, 2014, pp. 1775–1788.
[19] Y. Tian, R. Sukthankar, and M. Shah, "Spatiotemporal deformable part models for action detection," in *CVPR*, 2013.
[20] Y. Yang and M. Shah, "Complex events detection using data-driven concepts," in *ECCV*, 2012.
[21] Y. Zhou, B. Ni, R. Hong, M. Wang, and Q. Tian, "Interaction part mining: A mid-level approach for fine-grained action recognition," in *CVPR*, June 2015.
[22] S. Ma, L. Sigal, and S. Sclaroff, "Space-time tree ensemble for action recognition," in *CVPR*, June 2015.
[23] L. Wang, Y. Qiao, and X. Tang, "Action recognition with trajectory-pooled deep-convolutional descriptors," in *CVPR*, 2015, pp. 4305–4314.
[24] Y. Du, W. Wang, and L. Wang, "Hierarchical recurrent neural network for skeleton based action recognition," in *CVPR*, June 2015.
[25] K. Schindler and L. V. Gool, "Action snippets: How many frames does human action recognition require?" in *CVPR*, 2008.
[26] D.-A. Huang, V. Ramanathan, D. Mahajan, L. Torresani, M. Paluri, L. Fei-Fei, and J. C. Niebles, "What makes a video a video: Analyzing temporal information in video understanding models and datasets," in *CVPR*, 2018.
[27] M. S. Ryoo, "Human activity prediction: Early recognition of ongoing activities from streaming videos," in *ICCV*, 2011.
[28] Y. Kong, D. Kit, and Y. Fu, "A discriminative model with multiple temporal scales for action prediction," in *ECCV*, 2014.
[29] B. Letham, C. Rudin, and D. Madigan, "Sequential event prediction," *Machine Learning*, vol. 93, pp. 357–380, 2013.
[30] C. Rudin, B. Letham, A. Salleb-Aouissi, E. Kogan, and D. Madigan, "Sequential event prediction with association rules," in *COLT*, 2011, pp. 615–634.
[31] C. Vondrick, H. Pirsiavash, and A. Torralba, "Anticipating visual representations from unlabeled video," in *CVPR*, 2016.
[32] K. M. Kitani, B. D. Ziebart, J. A. Bagnell, and M. Hebert, "Activity forecasting," in *ECCV*, 2012.
[33] A. Alahi, K. Goel, V. Ramanathan, A. Robicquet, L. Fei-Fei, and S. Savarese, "Social lstm: Human trajectory prediction in crowded spaces," in *CVPR*, 2016.
[34] M. Chen, Z. E. Xu, K. Q. Weinberger, and F. Sha, "Marginalized denoising autoencoders for domain adaptation," in *ICML*, 2012.
[35] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, and P.-A. Manzagol, "Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion," *JMLR*, vol. 11, pp. 3371–3408, 2010.
[36] D. Tran, L. Bourdev, R. Fergus, L. Torresani, and M. Paluri, "Learning spatiotemporal features with 3d convolutional networks," in *ICCV*, 2015.
[37] P. Dollar, V. Rabaud, G. Cottrell, and S. Belongie, "Behavior recognition via sparse spatio-temporal features," in *VS-PETS*, 2005.
[38] K. Soomro, A. R. Zamir, and M. Shah, "UCF101: A Dataset of 101 Human Action Classes From Videos in The Wild," CRCV-TR-12-01, Tech. Rep., 2012.
[39] A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar, and L. Fei-Fei, "Large-scale video classification with convolutional neural networks," in *CVPR*, 2014.
[40] L. Wang, Y. Xiong, Z. Wang, Y. Qiao, D. Lin, X. Tang, and L. V. Gool, "Temporal segment networks: Towards good practices for deep action recognition," in *ECCV*, 2016.
[41] G. Singh, S. Saha, M. Sapienza, P. Torr, and F. Cuzzolin, "Online real-time multiple spatiotemporal action localisation and prediction," in *ICCV*, 2017.
[42] Y. Kong and Y. Fu, "Max-margin action prediction machine," *TPAMI*, vol. 38, no. 9, pp. 1844 – 1858, 2016.
[43] L. Fan, W. Huang, C. Gan, S. Ermon, B. Gong, and J. Huang, "End-to-end learning of motion representation for video understanding," in *CVPR*, 2018.
[44] C. Feichtenhofer, A. Pinz, and R. P. Wildes, "Spatiotemporal multiplier networks for video action recognition," in *CVPR*, 2017, pp. 7445–7454.
[45] K. Simonyan and A. Zisserman, "Two-stream convolutional networks for action recognition in videos," in *NIPS*, 2014.
[46] J. Y.-H. Ng, M. Hausknecht, S. Vijayanarasimhan, O. Vinyals, R. Monga, and G. Toderici, "Beyond short snippets: Deep networks for video classification," in *CVPR*, 2015.
[47] W. Zhu, J. Hu, G. Sun, X. Cao, and Y. Qiao, "A key volume mining deep framework for action recognition," in *CVPR*, 2016.
[48] Y. Zhao, Y. Xiong, and D. Lin, "Recognize actions by disentangling components of dynamics," in *CVPR*, 2018.
[49] D. Tran, H. Wang, L. Torresani, J. Ray, Y. LeCun, and M. Paluri, "A closer look at spatiotemporal convolutions for action recognition," in *CVPR*, 2018.
[50] J. Carreira and A. Zisserman, "Quo vadis, action recognition? a new model and kinetics dataset," in *CVPR*, 2017.



Yu Kong (M'15) received B.Eng. degree in automation from Anhui University in 2006, and PhD degree in computer science from Beijing Institute of Technology, China, in 2012. He is now a tenure-track Assistant Professor in the B. Thomas Golisano College of Computing and Information Sciences at Rochester Institute of Technology, Rochester, NY, USA. Prior to that, he visited the National Laboratory of Pattern Recognition (NLPR), Chinese Academy of Science, and the Department of Computer Science and Engineering, University at Buffalo, SUNY. He was a postdoc in the Department of ECE, Northeastern University. Dr. Kong's research interests include computer vision, social media analytics, and machine learning. He is a member of the IEEE and ACM.



Zhiqiang Tao received the B.E. degree in software engineering from the School of Computer Software, and the M.S. degree in computer science from the School of Computer Science and Technology, Tianjin University, Tianjin, China, in 2012 and 2015, respectively. He is currently pursuing the Ph.D. degree in Northeastern University, Boston. His research interests include subspace learning, ensemble clustering and representation learning.



Yun Fu (S'07-M'08-SM'11) received the B.Eng. degree in information engineering and the M.Eng. degree in pattern recognition and intelligence systems from Xi?an Jiaotong University, China, respectively, and the M.S. degree in statistics and the Ph.D. degree in electrical and computer engineering from the University of Illinois at Urbana-Champaign, respectively. He is an interdisciplinary faculty member affiliated with College of Engineering and the College of Computer and Information Science at Northeastern University since 2012. His research interests are Machine Learning, Computational Intelligence, Big Data Mining, Computer Vision, Pattern Recognition, and Cyber-Physical Systems. He has extensive publications in leading journals, books/book chapters and international conferences/workshops. He serves as associate editor, chairs, PC member and reviewer of many top journals and international conferences/workshops. He received seven Prestigious Young Investigator Awards from NAE, ONR, ARO, IEEE, INNS, UIUC, Grainger Foundation; nine Best Paper Awards from IEEE, IAPR, SPIE, SIAM; many major Industrial Research Awards from Google, Samsung, and Adobe, etc. He is currently an Associate Editor of the IEEE Transactions on Neural Networks and Leaning Systems (TNNLS). He is fellow of IAPR, OSA and SPIE, a Lifetime Senior Member of ACM, Lifetime Member of AAAI and Institute of Mathematical Statistics, member of ACM Future of Computing Academy, Global Young Academy, AAAS, INNS and Beckman Graduate Fellow during 2007-2008.