A Dynamic Frame Selection Framework for Fast Video Recognition

Zuxuan Wu, Hengduo Li, Caiming Xiong, Yu-Gang Jiang, and Larry S. Davis, *Fellow, IEEE*

Abstract—We introduce AdaFrame, a conditional computation framework that adaptively selects relevant frames on a per-input basis for fast video recognition. AdaFrame, which contains a Long Short-Term Memory augmented with a global memory to provide context information, operates as an agent to interact with video sequences aiming to search over time which frames to use. Trained with policy search methods, at each time step, AdaFrame computes a prediction, decides where to observe next, and estimates a utility, i.e., expected future rewards, of viewing more frames in the future. Exploring predicted utilities at testing time, AdaFrame is able to achieve adaptive lookahead inference so as to minimize the overall computational cost without incurring a degradation in accuracy. We conduct extensive experiments on two large-scale video benchmarks, FCVID and ActivityNet. With a vanilla ResNet-101 model, AdaFrame achieves similar performance of using all frames while only requiring, on average, 8.21 and 8.65 frames on FCVID and ActivityNet, respectively. We also demonstrate AdaFrame is compatible with modern 2D and 3D networks for video recognition. Furthermore, we show, among other things, learned frame usage can reflect the difficulty of making prediction decisions both at instance-level within the same class and at class-level among different categories.

Index Terms—Video classification, conditional computation, deep neural networks, reinforcement learning

1 Introduction

The ubiquity of smartphones and sharing activities on social media have driven tremendous growth of Internet videos—it is estimated that around 300 hours of video are uploaded to YouTube every minute of every day! As a result, automated approaches that can categorize actions and events both effectively and efficiently are demanded for applications like indexing, searching, recommendation, etc. While a plethora of work has been developed to derive discriminative and powerful video representations for improved recognition accuracy [1], [2], [3], [4], [5], limited effort has been devoted to accelerating the inference of video classification [6], [7], [8], [9], [10].

In standard video classification and action recognition pipelines, classification scores from multiple uniformly sampled frames, if not every single frame [11], are fused averagely as final video-level predictions at testing time. Here, we use frame as a general term, and it can take the forms of a single RGB image, stacked RGB frames (typically referred to as snippets/clips) for 3D convolutional networks, and stacked optical flow images in two-stream networks [3]. Such a uniform sampling strategy while straightforward has been widely used in state-of-the-art video classification frameworks due to its effectiveness and simplicity. However, the computational cost to analyze a single frame is still high since high-capacity backbone networks, such as ResNet [12], InceptionNet [13], I3D [14], Non-local Networks [2], SlowFast [5], etc., are usually used to ensure decent performance. In addition, uniform sampling expects information to be evenly distributed over time, and thus background frames or noisy frames that are not helpful for recognizing the class of interest might be included as well and can hurt recognition accuracy.

Moreover, frames needed to make correct predictions differ for distinct categories—we usually only need a single frame to recognize videos containing objects (e.g., "elephants" and "giraffes") or still scenes (e.g., "dessert" and "river") and we need to obverse more frames to differentiate similar actions (e.g., "drinking beer" and "drinking coffee") and procedural events (e.g., "making pizza" or "making cake"). In addition, frames required are also different even for videos within the same class given that there are large-intra class variations for Internet videos. For example, a "playing football" event can be recorded by different devices (e.g., smartphones or DSLR cameras) from various view points (e.g., different locations of a stadium), take place either indoor or outdoor, with players of different professional levels. Consequentially, the number of frames demanded to make accurate predictions for the same category is different and instance-specific.

In light of this, we study how to dynamically allocate computational resources within a framework on a per-video basis to achieve efficient video recognition—-conditioned on different input videos, we aim to select a small number of frames to produce accurate predictions (See Fig. 1 for a conceptual overview). However, this is a non-trivial problem, given that video classification can be considered as learning with weak supervision—only a single label is

- *Zuxuan Wu is with the School of Computer Science, Fudan University, Shanghai 200433, China. E-mail: zxwu@fudan.edu.cn.*
- *Hengduo Li and Larry S. Davis are with the Department of Computer Science, University of Maryland, College Park, MD 20742 USA. E-mail: {hdli, lsd}@umiacs.umd.edu.*
- *Caiming Xiong is with the Salesforce Research, Palo Alto, CA 94301 USA. E-mail: cxiong@salesforce.com.*
- *Yu-Gang Jiang is with the School of Computer Science, Fudan University, Shanghai 200433, China. E-mail: ygj@fudan.edu.cn.*

Manuscript received 26 Feb. 2020; revised 14 Sept. 2020; accepted 28 Sept. 2020. Date of publication 7 Oct. 2020; date of current version 4 Mar. 2022. (Corresponding author: Yu-Gang Jiang.) Recommended for acceptance by V. Lepetit. Digital Object Identifier no. 10.1109/TPAMI.2020.3029425

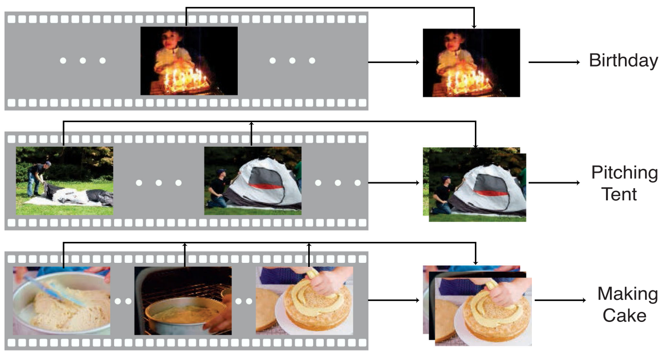


Fig. 1. *A conceptual overview of our approach.* To minimize the overall computational cost, AdaFrame learns to choose a small number of frames in order to make accurate predictions on a per-video basis.

provided for an entire video sequence—and there is no ground-truth information indicating which frames are relevant and critical for recognizing the class of interest. As a result, it is extremely challenging to search through the temporal horizon of videos in order to select which frames to incorporate, and effectively model the temporal information of these chosen frames.

In this paper, we present AdaFrame, a generic conditional computation framework, which learns to dynamically select frames conditioned on input videos to achieve fast video recognition. More specifically, it consists of a Long Short-Term Memory (LSTM) network augmented with a global memory, which is derived with lightweight feature representations to provide context information. The global memory, containing features computed from either spatially and temporally downsampled video frames or audio frames, guides the LSTM to exploit over time so as to learn video-specific frame usage policies. Interacting with video sequences as an agent, the memory-augmented LSTM, at each time step, observes the present frame and queries the global memory to derive global context information in order to produce a classification score, to determine which frame to observe next, and to compute the utility of taking in more frames in the future. AdaFrame is trained with policy gradient methods in a reinforcement learning paradigm to maximize a carefully designed reward function, which incentives more accurate and confident predictions after analyzing one more frame each time. During inference, conditioned on input videos, AdaFrame leverages predicted future utilities, which indicate the advantages of observing more frames, to achieve adaptive lookahead inference.

Extensive experiments are conducted on two large-scale video datasets, FCVID [15] for generic video classification and ActivityNet [16] for activity recognition, both of which contain untrimmed videos with an average duration over 100 seconds. With a standard ResNet-101 as its backbone, AdaFrame performs better or on par with the popular uniform sampling strategy, a simple yet strong baseline, on FCVID and ActivityNet respectively, while requiring 58.9 and 63.3 percent less computation on average, going as high as 90.6 percent. AdaFrame also outperforms by clear margins alternative frame selection approaches [17], [18]. We also show that AdaFrame is compatible with modern network architectures designed for videos like SlowFast [5] and DPN [19] trained with temporal segment networks [4]. In particular, AdaFrame only requires 42 and 55.4 percent of computation of a standard SlowFast-8 $\times$ 8 model to achieve similar accuracy, i.e., 83.0 and 84.0 percent on FCVID and ActivityNet, respectively. In addition, we demonstrate, among other things, the number of frames required to make correct predictions is video-specific—frame usage policies differ not only at category-level among different classes but also at instance-level within the same class.

A preliminary version of this paper appeared in [10]. The present paper includes a complete review of literatures on efficient video analysis; more comparisons with alternative frame selection methods; more thorough discussions on learned frame usage policy; new experiments of using audio features as a global memory to provide context information; additional experiments with more powerful yet computationally expensive 2D and 3D backbones, demonstrating the compatibility of AdaFrame with modern architectures.

2 Related Work

Efficient Video Analysis. There is a plethora of work on video classification and action recognition, most of which focuses on extending 2D convolution to the temporal domain in videos and learning robust video representations [1], [2], [3], [4], [5], [14], [20], [21]. In contrast, only a few approaches have been introduced to achieve efficient video classification [3], [7], [8], [22], [23], [24], [25], [26], [27]. More specifically, Zhang *et al.* [7] speed up the extraction of motion information with motion vectors in two-stream networks; researchers also present approaches that operate on compressed videos to extract motion clues [6], [28]. More recently, Lin *et al.* present temporal shifting modules by shifting feature maps along feature dimensions [22] and Zolfaghari *et al.* divide videos into sections uniformly and sample each frame from each section for temporal modeling with 3D convolutions [8]. In addition, Slowfast adopts a lightweight CNNs with fewer channels for the fast pathway and heavy CNNs with more channels for the slow pathway for recognition [5]. However, these CNN-based approaches all adopt the same strategy for inference—deriving prediction scores on uniformly sampled frames (typically 25 or 10) and averaging them as the prediction of a video clip; and they produce the same set of parameters for all videos. In contrast, we aim to choose a small number of relevant frames conditioned on input videos with an aim to achieve efficient video recognition. It is worth pointing out that our framework is generic and can be applied to both 2D and 3D CNNs designed for video recognition; the inputs to our framework can be either a single frame or stacked frames, depending on the backbone architectures.

Recently, a deep feature flow framework is introduced, which propagates feature maps of key frames to other frames with a flow field [29]. Pan *et al.* propose Recurrent Residual Modules by modeling the similarity of feature maps between neighboring frames to speed up inference [30]. They process videos frame by frame for video object detection tasks and attempt to reduce computation cost by exploring frame similarities, while our goal is to selectively choose relevant frames directly based on inputs.

Our work is more related to [17] and [18] that learn frame usage policy in videos with reinforcement learning. In particular, Yeung *et al.* introduce an agent for action detection [17] to learn whether to stop inference and where to look next by sampling from a video sequence. Although action detection is generally harder than classification, feedback about whether observed frames are important is straightforward and strong for training the agent, since frame-level annotations are readily available. In contrast, for classification tasks, frame-level supervision is missing, and thus learning to sample over time in videos is challenging. To mitigate this issue, Fan *et al.* propose to reduce the search space by sampling from a predefined action set, which indicates how many steps to jump [18]. While making sampling feasible for classification, it sacrifices flexibility since the steps are fixed. In contrast, we use a global memory to provide context information to help the search for relevant frames. In addition, we leverage predicted utilities for adaptive lookahead inference rather than learning a separate policy to decide when to stop. Korbar *et al.* recently introduce SCSampler to select salient clips with high prediction confidence using a separately trained lightweight model, and use those selected clips to compute final predictions with a heavy model [9]. However, the framework is trained in two-stages and could bring discrepancies between the lightweight model and the heavy model, whereas our model is an end-to-end framework that learns to select relevant frames. In addition, the lightweight model in SCSampler needs to observe all frames to select a few salient frames, where the computationally cost grows linearly with the number of frames.

Conditional Computation. Conditional/dynamic computation aims to save computation by learning whether to exit a model and stop inference conditioned on the confidence of classifiers. Cascaded classifiers [31] that quickly reject easy negative sub-windows for fast face detection are among the earliest approaches along this line of search. More recently, several approaches propose to incorporate decision branches to different layers of neural networks to determine whether to proceed inference when high classification confidence has been achieved [32], [33], [34], [35]. There are also a few methods that learn to decide which convolutional layers in residual networks to use on a per-image basis for dynamic inference [36], [37], [38]. In this paper, we aim to select a small number of frames in videos conditioned on inputs rather than layers/neurons in networks to achieve fast inference. Note that our framework can be used together with approaches that select layers in neural networks by replacing the standard backbone networks with dynamic inference backbones.

3 Approach

Given a video at test time, we aim to produce a frame usage policy, conditioned on the input, which uses as few frames as possible while making correct classification predictions. Towards this goal, we present AdaFrame, a memory-augmented LSTM (Section 3.1), to effectively explore the temporal horizon of videos assisted with context information derived from a global memory. During training, AdaFrame is optimized to learn instance-specific frame usage policies and model the temporal information of those observed frames. At test time, given an optimized model, we leverage predicted utilities of seeing more frames in the future to perform adaptive lookahead inference (Section 3.2) to accommodate different computational budget.

3.1 Memory-Augmented LSTM

Given a video sequence of T frames with representations denoted as $\{\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_T\}$, the memory-augmented LSTM operates as an agent recurrently interacting with the video. More formally, at the tth time step, features of the current step $\boldsymbol{v}_t$, hidden states $\boldsymbol{h}_{t-1}$ and cell states $\boldsymbol{c}_{t-1}$ from the previous time step, as well as a global context vector $\boldsymbol{u}_t$ computed by querying a global memory $\mathbf{M}$, are input into the the LSTM to generate the hidden states $\boldsymbol{h}_t$ and cell states $\boldsymbol{c}_t$ of the the tth time step:

$$\boldsymbol{h}_t,\, \boldsymbol{c}_t = \texttt{LSTM}([\boldsymbol{v}_t, \boldsymbol{u}_t],\, \boldsymbol{h}_{t-1},\, \boldsymbol{c}_{t-1}), \tag{1}$$

where [,] represents the concatenation of two features. The hidden states $\boldsymbol{h}_t$ are then used as inputs to a prediction network $f_p(\boldsymbol{h}_t)$ to generate classification scores, based on which a reward r_t is given to evaluate whether the transition to the current frame increases the prediction confidence. In addition, the hidden states $\boldsymbol{h}_t$ are also input to a selection network $f_s(\boldsymbol{h}_t)$ to determine where to look next, and a utility network $f_u(\boldsymbol{h}_t)$ to compute the advantage of observing more frames in the future. An overview of the proposed framework is illustrated in Fig. 2. Below we introduce each component in AdaFrame in detail.

Global Memory. The goal of the LSTM is to choose a small number of informative frames and make accurate predictions through searching the temporal horizon guided by rewards provided. However, the enormous search space over time and limited capacity of hidden states to remember input history [39], [40] pose significant challenges for learning where to observe next. To mitigate this issue, we associate a global memory with each video consisting of representations that can be derived economically to provide context information. To generate the global memory, one can use a lightweight CNN model to compute features from spatially and temporally downsampled RGB frames or exploit acoustic clues based on log-mel audio spectrograms (see more details in Section 4.1 for different instantiations of the memory). More formally, we represent the memory as $\mathbf{M} = [\boldsymbol{v}^s_1, \boldsymbol{v}^s_2, \ldots, \boldsymbol{v}^s_{T_d}]$, where T_d is the number of frames ($T_d < T$) in the memory. The constraint $T_d < T$ is to guarantee the computational overhead of the global memory is small. Since there is no explicit order information in the memory as features are generated frame-by-frame, we use positional encoding [41] to inject positions into these lightweight features. To derive a vector containing global context information, we first use the hidden states of the LSTM to query the global memory. This results in a normalized attention vector $\boldsymbol{\beta}_t = \texttt{Softmax}(z_t)$, where $z_{t,j}$ indicates an unnormalized attentional weight for the corresponding element in the memory:

$$z_{t,j} = (\boldsymbol{W}_h \boldsymbol{h}_{t-1})^\top PE(\boldsymbol{v}^s_j). \tag{2}$$

Here $\boldsymbol{W}_h$ is a learnable matrix transforming hidden states to the same dimension as the jth representation $\boldsymbol{v}^s_j$ in the memory and PE represents the function of applying positional encoding to features. Then we compute the global context vector as the weighted sum of the global memory: $\boldsymbol{u}_t = \boldsymbol{\beta}_t^\top \mathbf{M}$. The core idea of using soft-attention to compute a global context vector as inputs to the LSTM is to estimate the current progress roughly using temporal information

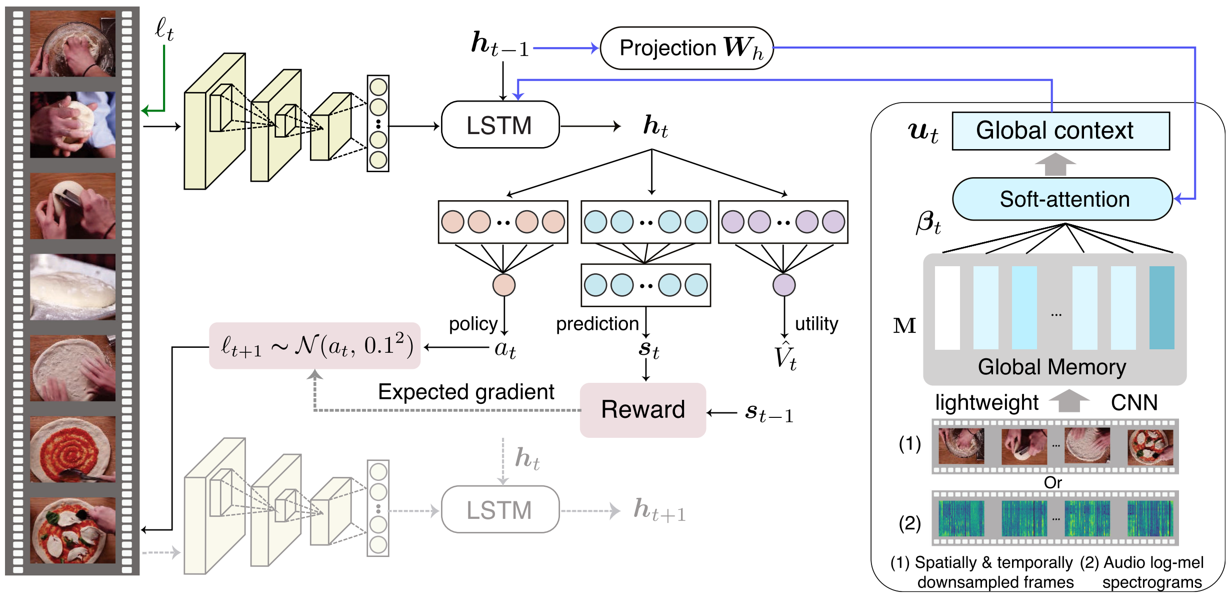


Fig. 2. *An overview of AdaFrame*. AdaFrame contains a memory-augmented LSTM, which serves as an agent to interact with a video sequence. At each time step, features from the current step, previous states, and a global context vector are used as inputs to the LSTM to produce the current hidden states. Conditioned on the hidden states, a prediction is emitted, a policy determining where to look next is produced, and a utility estimating expected future rewards is computed. See texts for more details.

encoded in the memory. This provides global context information that can help the agent to learn where to look in the future.

Prediction Network. The prediction network $f_p(\boldsymbol{h}_t; \boldsymbol{W}_p)$ parameterized by weights $\boldsymbol{W}_p$ transforms the hidden states $\boldsymbol{h}_t$ to outputs $\boldsymbol{s}_t \in \mathbb{R}^C$ using one fully-connected layer, where C denotes the number of categories. Furthermore, we normalize $\boldsymbol{s}_t$ with `Softmax` to generate probability scores for each class, and we train the prediction network with a standard cross-entropy loss, using outputs from the last time step K of the LSTM :

$$\mathcal{L}_{cls}(\boldsymbol{W}_p) = -\sum_{c=1}^{C} \boldsymbol{y}^c \log(\boldsymbol{s}_K^c). \tag{3}$$

Here $\mathbf{y}$ is a one-hot label vector of the corresponding video. During training, we also constrain $K \ll T$, since we wish to use as few frames as possible.

Reward Function. After the prediction network emits classification scores $\boldsymbol{s}_t$ for the tth time step, we associate the action, i.e., the transition from the last step to the current frame, with a reward to measure whether the action brings additional information gain. With one more frame observed, the agent is expected to make more accurate predictions. Therefore, we introduce a reward function that incentives the classifier to be more confident when observing more frames. More formally, when $t > 1$, the reward function is written as:

$$\begin{aligned} r_t = &\ \max\{0,\, m_t - \max_{t' \in [0t-1]} m_{t'}\}, \\ \text{where} \quad m_t = &\ \boldsymbol{s}_t^{gt} - \max\{\boldsymbol{s}_t^{c'} | c' \neq gt\}. \end{aligned} \tag{4}$$

Here m_t represents the difference between the scores of the ground-truth category (denoted by gt) and the largest probabilities from other classes. This is to ensure the probability of the ground-truth class is larger by a margin than the remaining classes. The reward function in Eq. (4) assigns a positive reward if the current margin is larger than historical ones, encouraging the classification confidence to increase with more frames observed. Since there is no ground-truth supervision explicitly informing whether a selected frame is informative, the margin based constraint serves a proxy to estimate whether the jump from the previous time step is useful for categorizing targets.

Selection Network. The selection network $f_s(\boldsymbol{h}_t; \boldsymbol{W}_s)$, parameterized by $\boldsymbol{W}_s$, defines a policy using a Gaussian distribution with fixed variance, to determine where to look next. More specifically, conditioned on hidden states $\boldsymbol{h}_t$, which consists of information of current inputs and historical context, the selection network produces a 1-D output $f_s(\boldsymbol{h}_t; \boldsymbol{W}_s) = a_t = \mathtt{sigmoid}(\boldsymbol{W}_s^\top \boldsymbol{h}_t)$, representing the mean of the location policy. Following [42], during training, we sample from the policy $\ell_{t+1} \sim \pi(\cdot|\boldsymbol{h}_t) = \mathcal{N}(a_t, 0.1^2)$; and during inference, we simply use the outputs as the location. In addition, we also normalize ℓ_{t+1} to be in the range of $[0, 1]$, so that it can be mapped to a frame index multiplying by the total number of frames. During training, at each time step, the selection network searches over the entire time space for informative frames without any constraints on where it can go. It can not only jump forward into the future but also go back to re-examine past information. The selection network is trained to maximize expected future rewards:

$$J_{sel}(\boldsymbol{W}_s) = \mathbb{E}_{\ell_t \sim \pi(\cdot|\boldsymbol{h}_t; \boldsymbol{W}_s)}\left[\sum_{t=0}^{K} \gamma^t r_t\right], \tag{5}$$

where γ is the discount factor fixed to 0.9. Maximizing expected future rewards incentivizes the agent at each state during the temporal horizon to select an action that is likely to increase its prediction confidence in the future

Algorithm 1. The Pseudo-Code for Training Our Network

```
Require: An input video $\mathbf{V}$, its global memory $\mathbf{M}$ and label $\boldsymbol{y}$
1:  Set the number of epochs for training $E$.
2:  Initialize hidden states $\boldsymbol{h}_0$ and cell states $\boldsymbol{c}_0$
3:  Set $\ell = 0$, and $v_0 = 0$
4:  for $e \leftarrow 1$ to $E$ do
5:      for $t \leftarrow 1$ to $K$ do
6:          $\boldsymbol{u}_t \leftarrow \boldsymbol{\beta}_t^\top \mathbf{M}$    ▷ obtain global context vector
7:          $\boldsymbol{h}_t, \boldsymbol{c}_t \leftarrow \texttt{LSTM}([\boldsymbol{v}_l, \boldsymbol{u}_t], \boldsymbol{h}_{t-1}, \boldsymbol{c}_{t-1})$
8:          $\boldsymbol{s}_t \leftarrow f_p(\boldsymbol{h}_t; \boldsymbol{W}_p)$    ▷ compute prediction
9:          Evaluate reward $r_t$ with Eq. (4)
10:         $a_t \leftarrow f_s(\boldsymbol{h}_t; \boldsymbol{W}_s)$
11:         $\ell \sim \mathcal{N}(a_t, 0.1^2)$
12:         Back-propagate gradients computed with Eq. (9)
13:     end for
14: end for
```

Algorithm 2. The Pseudo-Code for Adaptive Inference

```
Require: An input video $\mathbf{V}$, its global memory $\mathbf{M}$
Require: A trained network model AdaFrame-$K$, where $K$ indicates the number of steps it is trained for.
1:  Set $\mu$ and $p$ to accommodate different computational budget, $p = K/2 + 1$ if $\mu < 0.7$ otherwise $p = K$.
2:  Set $\ell = 0$, $c = 0$, $t = 0$ and $\hat{V}_t^{max} = 0$.
3:  while $c < p$ and $t < K$ do
4:      Compute the utility $\hat{V}_t$ for the current step $t$
5:      if $\hat{V}^{max} - \hat{V}_t > \mu$ then
6:          $c \leftarrow c + 1$
7:      end if
8:      $\hat{V}^{max} \leftarrow \max(\hat{V}^{max}, \hat{V}_t)$
9:      $t \leftarrow t + 1$
10: end while
11: Compute the prediction $p_t$ as the video-level score
```

conditioned on frames seen so far; a large γ ensures the agent to consider the future rather than being myopic.

Utility Network. The utility network, parameterized by $\boldsymbol{W}_u$, maps the hidden states $\boldsymbol{h}_t$ to a 1-D output $f_u(\boldsymbol{h}_t; \boldsymbol{W}_u) = \hat{V}_t = \boldsymbol{W}_u^\top \boldsymbol{h}_t$ with one fully-connected layer. The utility network predicts the advantage of seeing more frames in the future, serving as a critic to estimate expected future rewards from the current state, which is also referred to as the value function [43]:

$$V_t = \mathbb{E}_{\boldsymbol{h}_{t+1:K}, a_{t:K}} \left[\sum_{i=0}^{K-t} \gamma^i r_{t+i} \right]. \tag{6}$$

The rationale of approximating the value function V_t from empirical rollouts using the outputs of the utility network $\hat{V}_t$ is to update policy parameters in the direction of performance improvement. The outputs of the utility network can be used as a baseline function to stabilize training [43]; but more importantly, at test time, they equip the agent with the ability to look ahead, since they reflect the advantage of subsequently seeing more frames. The utility network is optimized to minimize the following regression loss:

$$\mathcal{L}_{utl}(\boldsymbol{W}_u) = \frac{1}{2} \| \hat{V}_t - V_t \|_2. \tag{7}$$

Optimization. Let Θ represent all trainable parameters, and we can derive the overall loss function by combining Eqs. (3), (4), and (7). The final objective function now reads:

$$\underset{\Theta}{minimize}\ \mathcal{L}_{cls} + \mathcal{L}_{utl} - J_{sel}. \tag{8}$$

Since the first two terms are differentiable, we can use back propagation with SGD to derive the optimal weights. Here, we only discuss how to optimize J_{sel} in Eq. (5) to maximize expected future rewards. Following [43], we compute the expected gradient of J_{sel} as:

$$\nabla_\Theta J_{sel} = \mathbb{E} \left[\sum_{t=t}^{\infty} (R_t - \hat{V}_t) \nabla_\Theta \log \pi_\theta(\cdot | \boldsymbol{h}_t) \right], \tag{9}$$

where R_t represents the expected future reward, and $\hat{V}_t$ is a baseline function to stabilize training by reducing variance [43]. We approximate the expected gradient in Eq. (9) by Monte-Carlo sampling using videos sampled in a mini-batch, and the approximated gradient is further back propagated downstream for training.

In summary, AdaFrame works in the following way. At the tth time step, it examines the current frame feature, which is further combined with a global context vector derived from the global memory. The concatenated features are then input into the LSTM to generate current hidden states to produce a prediction, based on which a reward is given. In addition, conditioned on the hidden states, the selection network further decides which frame to look at next, and the utility network estimates the advantage of seeing more frames. Algorithm 1 presents the training algorithm of AdaFrame.

3.2 Adaptive Lookahead Inference

Although AdaFrame is trained for a fixed number of steps, we wish to perform adaptive inference at test time—conditioned on different input videos, AdaFrame selects a small number of informative frames while producing accurate predictions. This is achieved with the help of the utility network, which is optimized to compute expected future rewards, measuring the advantage of observing more frames in the future.

In particular, we leverage the outputs from the utility network as stop signals to decide whether to proceed inference by looking ahead. One can directly select a threshold, and stop inference once the predicted utility $\hat{V}_t$ of the tth time step is smaller than the predefined threshold. While straightforward and simple, it is challenging to set a universal optimal threshold for all samples, regardless of their complexity. To address this issue, we store the maximal utility $\hat{V}^{max}$ for each sample over time, and the current predicted utility $\hat{V}_t$ is compared with its running max $\hat{V}_t^{max}$ at each time step. If $\hat{V}_t$ is smaller than $\hat{V}_t^{max}$ by a margin μ more than p times, we will exit the model and use probabilities of the current time step as the final video-level predictions. Here, we introduce μ to accommodate different computational budget by balancing the trade-off between cost and accuracy. More specifically, a small μ will force early exiting when the predicted utility no longer increases; on the other hand, when μ is large, the model is able to tolerate a decrease in utility, allowing more "pondering" by seeing more frames before emitting predictions. In addition, p is used as a patience metric, allowing the current utility to be smaller than the running max for a few times. This is motivated by learning rate scheduling on plateaus for network training, which waits for a few more epochs before reducing learning rates when the loss function does not further drop. In summary, when accurate predictions are needed with sufficient computation budget, we use a large μ and p to explore more frames, whereas we set a small μ and p to force early prediction strictly when computational budget is limited.

While we use the same threshold μ for all videos, whether to stop inference is conditioned on sample-specific utility distributions. Such a stopping criterion is softer than comparing $\hat{V}_t$ with μ directly. Note that one can learn an additional binary policy to decide whether to stop inference with another network as in [17], [18], however jointly learning a frame selection policy and a stopping policy is challenging, as will be demonstrated in the experiments. Instead, we use the outputs of the utility network without incurring additional parameters for adaptive lookahead inference. Algorithm 2 presents the pseudo-code for adaptive lookahead inference.

4 Experiments

In this section, we first introduce the experimental setup and then present extensive qualitative and quantitative experiments to evaluate AdaFrame.

4.1 Experimental Setup

Datasets and Evaluation Metrics. We use two challenging large-scale video datasets, Fudan-Columbia Video Datasets (FCVID) [15] and ActivityNet [16], to evaluate the proposed framework.

There are 91,223 YouTube videos in FCVID with an average duration of 167 seconds, manually labeled into 239 categories. These classes cover a wide range of topics, including scenes (e.g., "forest"), objects (e.g., "gorilla"), activities (e.g., "playing football"), and complex events (e.g., "making egg tarts"). FCVID is split to a training set with 45,611 videos, and a testing set with 45,612 videos. ActivityNet (version 1.3) contains $20K$ YouTube videos with an average duration of 117 seconds. There are 200 classes in ActivityNet, all of which are activity-focused. ActivityNet is split into a training set with 10,024 videos, a validation set of 4,926 videos and a testing set of 5,044 videos. We evaluate our approach on the validation set since labels for the testing set are not publicly available. Unlike trimmed videos in UCF-101 [44] with an average duration of 8 seconds, videos in both FCVID and ActivityNet are *untrimmed*, for which efficient recognition is extremely critical given the redundant nature of videos.

To evaluate the performance, we use average precision (AP) for each category and calculate mean average precision (mAP) to measure the overall accuracy on both datasets. We measure the computational cost to make predictions with Giga Floating-point Operations Per Second (GFLOPS), which is a hardware independent metric.

Implementation Details. We use a one-layer LSTM with 2,048 and 1,024 hidden units for FCVID and ActivityNet respectively. To extract inputs for the LSTM, we decode videos at 1fps and compute features from the penultimate layer of different CNN backbones, aiming to verify the compatibility of AdaFrame with modern architectures. In particular, for 2D based architectures, we experiment with a ResNet-101 model [12] and a Dual Path Network (DPN-107) model [19], which achieves a top-1 accuracy of 77.4 and 80.3 percent on ImageNet validation set, respectively. For 3D based networks, we adopt a SlowFast-8 $\times$ 8 model with a backbone of ResNet-50 [5], which takes in as inputs 8 stacked RGB frames sampled uniformly from 64 frames with a stride of 8 and it obtains a top-1 accuracy of 77.0 percent on Kinetics [45]. And these networks are further finetuned on target datasets; the DPN is finetuned with temporal segment networks [4]. The computational cost for ResNet-101, DPN-107, and SlowFast-8 $\times$ 8 is 7.82 GFLOPs, 18.34 GFLOPs, and 65.71 GFLOPs for a single frame,[1] respectively.

To generate the global memory that provides context information, we use lightweight CNNs to compute economical visual or audio features to reduce overhead. In particular, for visual features, we operate on spatially and temporally downsampled frames by lowering the resolution of video frames to 112×112 and sampling 16 frames uniformly. We use a pretrained MobileNetv2 [46] as the lightweight CNN, offering a top-1 accuracy of 52.3 percent on ImageNet with input images downsampled to 112×112. To extract audio features, we first convert audio to spectrogram with Short-Time Fourier Transform, and then we derive log-mel spectrogram with 64 mel bins. We then sample 16 images with a size of 96×64 from the log-spectrogram and compute features with a MobileNetv1 [47] pretrained on AudioSet [48] for each audio frame. The computational cost for visual and audio feature extraction is 0.08 GFLOPs and 0.07 GFLOPs per frame, respectively. Throughout the experiments, we use visual global memory by default.

Our implementation is based on PyTorch and we use SGD for optimization. We set the momentum to 0.9, the weight decay to $1e-4$. The network is trained for 100 epochs using a batch size of 128 and 64 for FCVID and ActivityNet, respectively. We set the initial learning rate to $1e-3$ and further decrease it by a factor of 10 every 40 epochs. The patience metric p is set to $K/2+1$ when $\mu < 0.7$, and K when $\mu >= 0.7$, where K denotes the number of time steps the model is trained for.

4.2 Main Results

In this section, we first present results of AdaFrame using a ResNet-101 as backbone, which offers decent recognition accuracy with moderate computational cost. We then explore more powerful yet computationally more expensive backbones to demonstrate that AdaFrame can be used together with modern architectures for video recognition.

4.2.1 Results With ResNet-101

Effectiveness of Learned Frame Usage. During training, AdaFrame is optimized with a fixed number of K steps; at test time, adaptive lookahead inference is performed with $\mu = 0.7$, resulting in K' frames used on average to produce similar recognition accuracy as observing all K frames. We compare

1. We abuse the term frame here; for SlowFast, it is a snippet with 8 frames.

TABLE 1
Results of Different Frame Selection Strategies on FCVID and ActivityNet

	FCVID							ActivityNet						
Method	R8	U8	R10	U10	R25	U25	All	R8	U8	R10	U10	R25	U25	All
AvgPooling	78.3	78.4	79.0	78.9	79.7	80.0	80.2	67.5	67.8	68.9	68.6	69.8	70.0	70.2
LSTM	77.8	77.9	78.7	78.1	78.0	79.8	80.0	68.7	68.8	69.8	70.4	69.9	70.8	71.0
AvgPooling-G	78.2	79.1	78.7	79.5	79.7	80.1	80.3	66.8	68.0	67.4	68.9	69.1	70.1	70.3
LSTM-G	78.8	79.2	79.0	79.7	79.4	80.1	80.4	69.6	70.3	70.5	70.8	70.7	71.2	71.8
SCSampling	79.6		79.9		80.9			65.8		66.8		70.0		
AdaFrame	78.6 5 $\rightarrow$ 4.92		79.2 8 $\rightarrow$ 6.15		80.2 10 $\rightarrow$ 8.21			69.5 5 $\rightarrow$ 3.8		70.4 8 $\rightarrow$ 5.82		71.5 10 $\rightarrow$ 8.65		

We use $K \rightarrow K'$ to denote the number of frames used by AdaFrame, which takes in K frames during training and uses an average of K' frames with adaptive inference. Here, R and U represents random and uniform sampling, respectively. See texts for more details.

AdaFrame with the following alternative approaches to compute final predictions during inference: (1) AvgPooling, which simply derives a classification score for each sampled frame and then averages these scores as the video-level prediction; (2) LSTM, which uses hidden states from the last time step of an LSTM to generate predictions; (3) AvgPooling-G, which fuses the prediction scores of the ResNet-101 with those from the MobileNetv2; (4) LSTM-G, which augments a standard LSTM with a global memory, and derives video-level predictions with hidden states from the last time step; (5) SCSampling [9], which uses frames, selected based on the confidence of the MobileNetv2 model, to compute averagely-pooled scores on these frames with the ResNet-101 model. We also use different number of frames ($K + \Delta$) as inputs for AvgPooling, LSTM and SCSampling; for AvgPooling, LSTM, we explore both uniform (U) or random (R) sampling strategies. To offset the additional computation cost incurred (as will be discussed later), we use K for AdaFrame while $K + \Delta$ for alternative approaches.

Table 1 presents the results of AdaFrame and comparisons with other methods. We can see that AdaFrame outperforms AvgPooling and LSTM in accuracy while requiring fewer frames under different settings on both FCVID and ActivityNet. More specifically, AdaFrame only uses an average of 4.92 and 3.8 frames to achieve a mAP of 78.6 and 69.5 percent on FCVID and ActivityNet, respectively. With 3.08 and 4.2 fewer frames used, AdaFrame outperforms AvgPooling and LSTM with 8 frames and achieves comparable results with 10 frames used for AvgPooling and LSTM. We also observe that AdaFrame matches the accuracy of using all frames with only 8.21 and 8.65 frames on FCVID and ActivityNet. This demonstrates that AdaFrame can indeed derive frame selection policies that use a small number of frames while maintaining the same accuracy. Although adding the global memory to AvgPooling and LSTM improves the performance, AdaFrame still achieves similar results using fewer frames. Furthermore, the performance of random sampling is similar to that of uniform sampling with AvgPooling and LSTM; AvgPooling is slightly better than LSTM on FCVID, possibly resulting from large intra-class variations brought by a diverse set of categories. It worth pointing out that while AvgPooling is straightforward and simple, it is a very powerful baseline and has been widely adopted for almost all CNN-based approaches due to its effectiveness. AdaFrame-10 also outperforms SCSampling with 10 frames on both datasets, and it is worth noting SCSampling observes all frames with the MobileNetv2 model before selecting salient clips used for the ResNet model, and thus the overhead with the lightweight model will be high. Instead, we only rely on 16 spatially and temporally down-sampled MobileNetv2 features as global memory.

Computational Savings With Adaptive Inference. We now evaluate computational savings of AdaFrame by performing adaptive inference and compare with alternative frame selection methods. We use AdaFrame-K to denote a model optimized with fixed K time steps during training; and for each derived model we adjust μ for adaptive inference to accommodate different computational budget.

We not only compare with heuristics to select frames, but also compare with FrameGlimpse [17] and FastForward [18]. In particular, FrameGlimpse learns to select frames and to stop inference for action detection tasks, in which frame-level ground-truth labels are provided to measure whether selected frames are informative. These frame-level temporal boundaries, are however not available for classification tasks. For fair comparisons, we use our reward function to train FrameGlimpse. FastForward decides how many steps to jump forward/back by sampling from a predefined action set, and learns when to stop inference with a stop branch. Furthermore, we also augment FrameGlimpse and FastForward with the global memory for fair comparisons, denoted as FrameGlimpse-G and FastForward-G, respectively. The results are summarized in Fig. 3. We observe that accuracy slowly increase until it becomes saturated for AvgPooling and LSTM when more computational resources (frames) are allocated. Since the most expensive component in video classification pipelines is to compute features with CNNs, the computational cost increases linearly with the number of frames processed. As 1.32 GLOPs and 7.82 GLOPs are needed for the global memory and feature extraction with a ResNet-101 for a single frame, respectively, more computational savings will occur when more frames are used.

Offering similar recognition accuracy to AvgPooling and LSTM using 25 frames, AdaFrame-10 only requires 58.9 and 63.3 percent less computation on average on FCVID (80.2 versus ~195 GFLOPs) [2] and ActivityNet (71.5 versus ~195 GFLOPs), respectively. We also observe similar trends for

2. 195.5 GFLOPS for AvgPooling and 195.8 GFLOPs for LSTM.

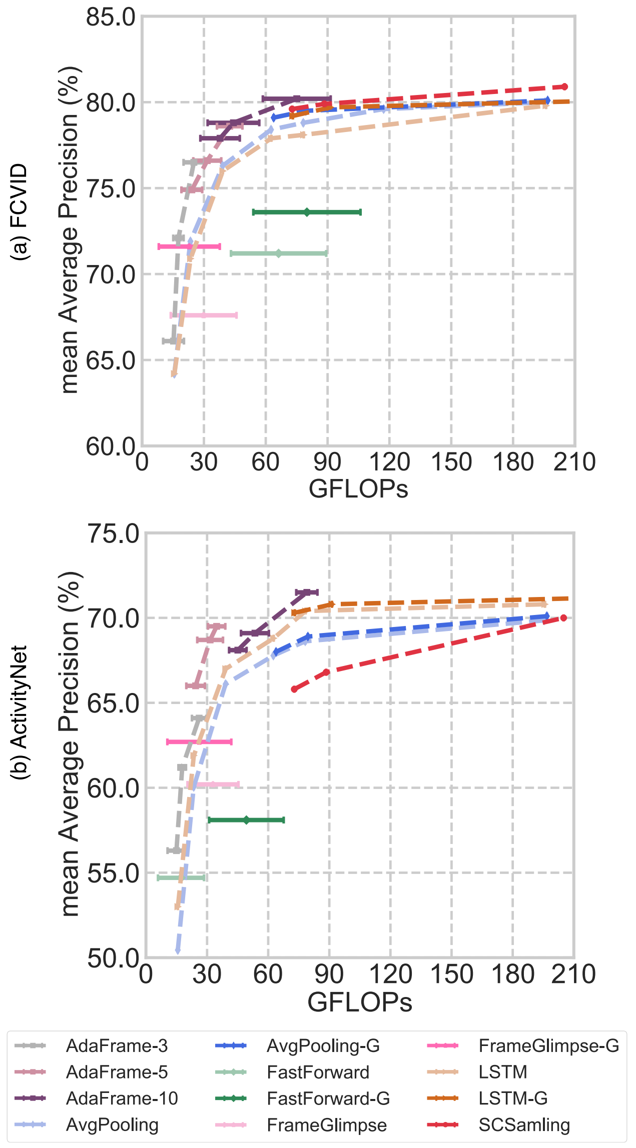


Fig. 3. *Mean average precision versus computational cost*. Results of AdaFrame and comparisons with FrameGlimpse [17], FastForward [18], as alternative frame selection strategies. For all AdaFrame models, each point in the line indicates a different μ (from left to right, μ is 0.2, 0.5 and 0.7).

AdaFrame-5 and AdaFrame-3 on both datasets. Although with fewer frames, fewer computational savings are achieved by AdaFrame over AvgPooling and LSTM, AdaFrame still produces better accuracy, i.e., 66.1 versus 64.2 percent on FCVID, and 56.3 versus 53.0 percent on ActivityNet. In addition, AdaFrame achieves better results than FrameGlimpse [17] and FastForward [18] in terms of both computational cost and accuracy. This suggests that coupling the learning of frame selection policy and stopping policy with reinforcement learning on large-scale datasets is challenging. In addition, we can see that attaching the global memory to provide context information can improve the accuracy of both FrameGlimpse and FastForward.

Furthermore, we observe that adjusting the threshold μ within the same model balances the trade-off between computational cost and accuracy. With a smaller threshold, the model is forced to make predictions as soon as possible, giving rise to lower accuracy and less frame usage (i.e., computational cost) at the same time. However, the derived frame usage policies using different thresholds still offer better performance than counterparts measured by both accuracy and computation needed. Comparing across different variants of AdaFrame, we observe that the best model of AdaFrame trained with a smaller K performs better or on par with that of AdaFrame trained with more time steps (a larger K) but is forced to make early predictions (smaller μ). For instance, AdaFrame-3 with $\mu = 0.7$ uses 25.1 GFLOPs on average on FCVID to obtain a mAP of 76.5 percent, outperforming AdaFrame-5 with $\mu = 0.5$, which uses 31.6 GFLOPs to achieve a mAP of 76.6 percent. This might due to the mismatch between training and testing—during training a large K allows the model to consider more by viewing more frames before making predictions. Although we can adjust the computation with different thresholds during inference, AdaFrame-10 is not fully optimized with extremely constrained information as AdaFrame-3 for recognition. This suggests it is important to apply different models conditioned on computational needs.

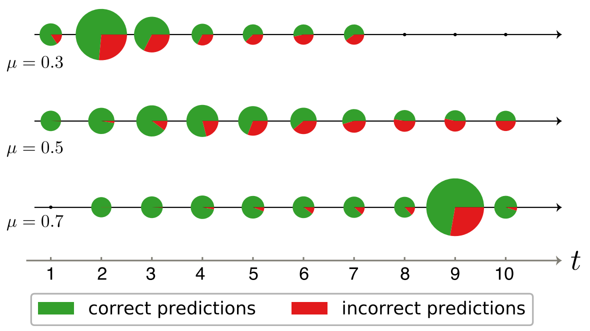


Fig. 4. *Data flow through AdaFrame over time*. Each circle indicates, by size, the fraction of samples that exit the model at the corresponding time step.

Analyses of Learned Policies. We now analyze learned frame usage policies by taking a trained AdaFrame-10 model and adjusting the threshold in order to meet different computational requirements. Fig. 4 illustrates, for different thresholds, how many samples exit the model at each time step, and the prediction accuracy of these samples. We observe that prediction accuracy decreases over time, i.e. higher accuracy in early early stages, pushing hard recognition decisions downstream by using more frames. In addition, when a larger μ is used to increase computational budget, more predictions are made at later stages.

Furthermore, we study whether computation required to make predictions for different categories is different. Therefore, we visualize in Fig. 5, the percentage of samples that exit the model at each time step using a subset of classes in FCVID. We can see that, for relatively simple classes like objects (e.g., "cow" and "giraffe") and static scenes ("Eiffel tower" and "cathedral exterior"), AdaFrame produces predictions for most of the videos in the first three steps. On the other hand, for some complicated cooking classes (e.g., "making ice cream" and "making hot tarts"), AdaFrame emits predictions in the middle of the entire time horizon. In addition, more frames are needed by AdaFrame to differentiate similar categories like "dining at restaurant" and "dining at home". We further demonstrate in Fig. 6 sampled videos that use different numbers of frames at testing time.

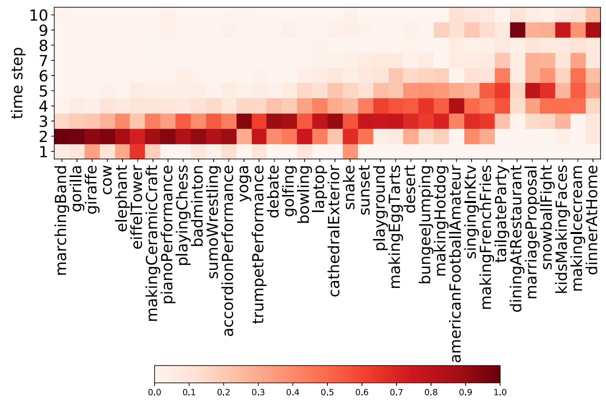


Fig. 5. *Learned inference policies for different classes over time.* Each square, by density, represents the percentage of samples that are classified at the corresponding time step from a certain class in FCVID.

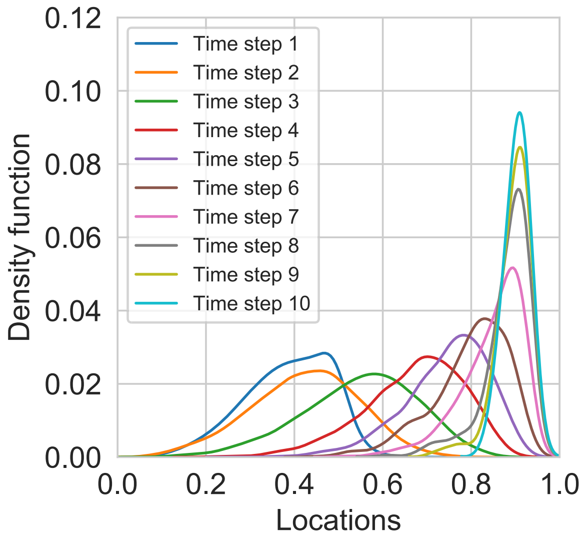


Fig. 7. *Density estimate function of locations used by AdaFrame at each time step.* The kernel density estimate function (Y-axis) provides an approximation of probability density function.

We observe frame usage differs not only across different categories but also within the same class (see the top two rows of Fig. 6). This results from large intra-class variations among different samples. For instance, for the "making cookies" category, AdaFrame requires four frames to produce correct predictions when there are severe camera motions and cluttered backgrounds in the video.

Moreover, we also investigate where the model jumps at each step. Using AdaFrame-10 with $\mu = 0.7$, we observe that the agent jumps back at least once 42.8 percent of videos on FCVID to extract information from previous time steps. This suggests AdaFrame is indeed flexible and bidirectional when searching over time. We also visualize the density estimate function of locations selected by AdaFrame-10 at each time step in Fig. 7. We can see that, for the 2-4 steps, AdaFrame samples through the entire video, possibly due to the prediction confidence is not very high.

4.2.2 Extensions to Modern Networks

We now study whether the framework can be used with more powerful yet computationally more expensive backbone networks. In particular, we adopt (1) a DPN-107 model [19] trained using temporal segment networks [4], which is a state-of-the-art framework based on 2D CNNs. During training, it learns to aggregate information from uniformly sampled frames with a Softmax function, and at test time the trained 2D CNN is used similarly as vanilla 2D networks; (2) a SlowFast-8×8 network [5], which samples 8 frames from a total of 64 frames with a stride of 8 and stacks them as inputs to 3D CNNs with two pathways. We compute features from the penultimate layer of the two backbones, (2,688-D for DPN and 2,304-D for SlowFast) as inputs to AdaFrame to learn frame usage conditioned on videos.

The results are summarized in Table 2. We can see from the table that AdaFrame achieves similar or better performance compared to their counterparts with much less computational cost for both backbones. In particular, on FCVID, AdaFrame-5-DPN achieves the same accuracy as DPN-10 while only requiring 56.5 percent less computation (79.82 versus 183.40 GFLOPs). And more computation can be saved with the SlowFast backbone, for example, AdaFrame-5-SlowFast produces a mAP of 83.0 percent with

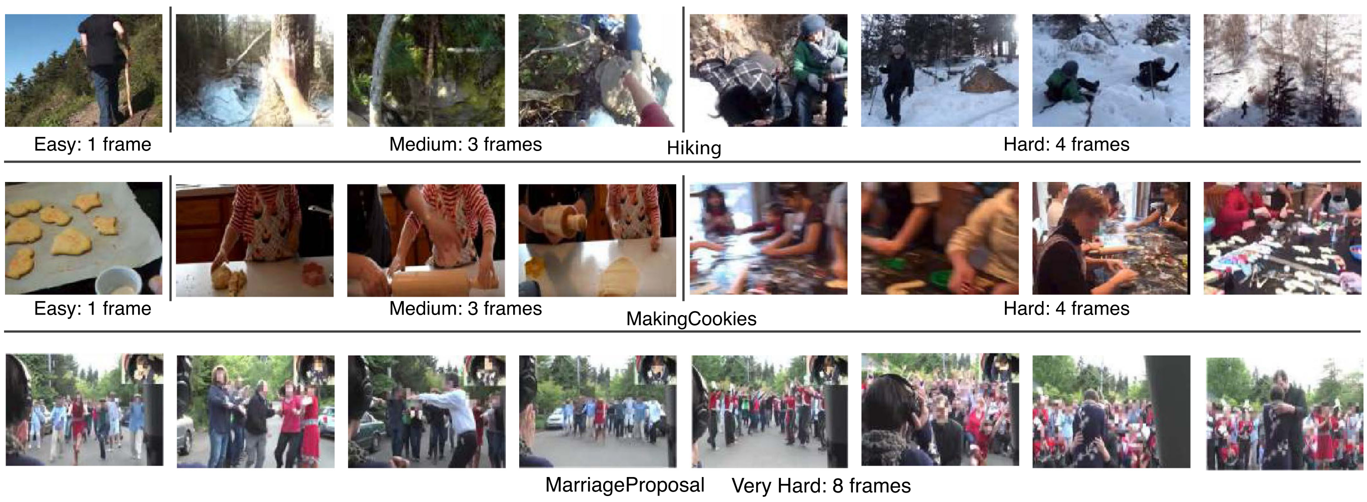


Fig. 6. *Sampled videos from the validation set of* FCVID *that use different number of frames for testing.* Frame usage is different not only among different categories but also within the same class (e.g., "making cookies" and "hiking").

TABLE 2
Results of Using DPN (Top) and SlowFast (Bottom) as the Backbone of AdaFrame on FCVID and ActivityNet

		FCVID				ActivityNet			
		μ	mAP	GFLOPs	# Frames	μ	mAP	GFLOPs	# Frames
DPN	DPN-3	—	77.5	55.02 ± 0.00	3.00 ± 0.00	—	74.6	55.02 ± 0.00	3.00 ± 0.00
	AdaFrame-3	0.2	75.2	37.35 ± 8.10	1.90 ± 0.41	0.2	74.4	44.63 ± 10.96	2.27 ± 0.56
	AdaFrame-3	0.5	78.5	46.40 ± 10.24	2.30 ± 0.52	0.5	76.1	52.89 ± 9.01	2.69 ± 0.46
	AdaFrame-3	0.7	79.4	54.46 ± 8.35	2.77 ± 0.43	0.7	76.7	56.82 ± 6.12	2.89 ± 0.31
	DPN-5	—	80.1	91.70 ± 0.00	5.00 ± 0.00	—	79.8	91.70 ± 0.00	5.00 ± 0.00
	AdaFrame-5	0.2	78.6	42.47 ± 11.86	2.16 ± 0.60	0.2	76.4	48.76 ± 15.03	2.48 ± 0.76
	AdaFrame-5	0.5	80.7	59.18 ± 16.60	3.01 ± 0.84	0.5	78.2	60.75 ± 16.32	3.09 ± 0.83
	AdaFrame-5	0.7	81.6	79.82 ± 22.21	4.06 ± 1.13	0.7	80.4	79.43 ± 21.82	4.04 ± 1.11
	DPN-10	—	81.6	183.40 ± 0.00	10.00 ± 0.00	—	82.3	183.40 ± 0.00	10.00 ± 0.00
	AdaFrame-10	0.2	78.7	48.36 ± 20.45	2.46 ± 1.04	0.2	77.9	53.67 ± 21.52	2.73 ± 1.09
	AdaFrame-10	0.5	80.8	76.48 ± 35.39	3.89 ± 1.80	0.5	79.6	77.46 ± 33.37	3.94 ± 1.70
	AdaFrame-10	0.7	82.3	134.87 ± 62.90	6.86 ± 3.20	0.75	82.4	134.28 ± 60.42	6.83 ± 3.08
SlowFast	SlowFast-3	—	78.8	197.13 ± 0.00	3.00 ± 0.00	—	77.7	197.13 ± 0.00	3.00 ± 0.00
	AdaFrame-3	0.2	70.6	96.52 ± 45.02	1.44 ± 0.67	0.2	75.6	134.73 ± 23.61	2.01 ± 0.35
	AdaFrame-3	0.5	78.8	136.10 ± 71.05	2.03 ± 1.06	0.5	78.0	169.59 ± 34.06	2.53 ± 0.51
	AdaFrame-3	0.7	80.2	146.80 ± 35.68	2.19 ± 0.53	0.7	79.5	194.39 ± 19.44	2.90 ± 0.29
	SlowFast-5	—	81.5	328.55 ± 0.00	5.00 ± 0.00	—	82.0	328.55 ± 0.00	5.00 ± 0.00
	AdaFrame-5	0.2	81.2	158.86 ± 59.34	2.37 ± 0.89	0.2	77.5	148.81 ± 42.03	2.22 ± 0.63
	AdaFrame-5	0.5	82.3	202.43 ± 63.03	3.02 ± 0.94	0.5	80.3	194.39 ± 61.87	2.90 ± 0.92
	AdaFrame-5	0.7	83.0	229.91 ± 55.65	3.43 ± 0.83	0.7	82.3	262.76 ± 83.20	3.92 ± 1.24
	SlowFast-10	—	83.0	657.10 ± 0.00	10.00 ± 0.00	—	84.2	657.10 ± 0.00	10.00 ± 0.00
	AdaFrame-10	0.2	77.8	178.97 ± 121.32	2.67 ± 1.81	0.2	75.4	138.75 ± 26.16	2.07 ± 0.39
	AdaFrame-10	0.5	81.8	305.66 ± 140.76	4.56 ± 2.10	0.5	79.6	202.43 ± 83.48	3.02 ± 1.25
	AdaFrame-10	0.7	83.5	544.95 ± 187.08	8.13 ± 2.79	0.75	84.0	364.64 ± 176.17	5.44 ± 2.63

We report variants of AdaFrame with both backbones to accommodate different computational budget.

only 35 percent of computation needed for SlowFast-10. We also observe computational savings with both backbones on ActivityNet. For instance, 26.8 and 44.5 percent computation can be saved using AdaFrame-10 with DPN and SlowFast, respectively, compared to their counterparts. This verifies that AdaFrame is a generic framework compatible with modern powerful architectures for video recognition.

Comparisons to State-of-the-Art Methods on ActivityNet. We also compare to state-of-the-art methods that design effective backbones for video recognition on ActivityNet. Table 3 summarizes the results. We can see that AdaFrame achieves comparable or better results compared with state-of-the-art methods while requiring few frames for inference.

4.3 Discussions

In this section, we conduct a set of experiments to justify our design choices of AdaFrame using the ResNet-101 model.

Global Memory. We conduct an ablation study to investigate the effectiveness of the global memory using different number of frames. The results are summarized in Table 4. We can see that, with global context information provided by the memory, AdaFrame achieves better performance than the non-memory counterpart; using 16 frames in the memory provides the best trade-off between computational overhead and accuracy. In addition, we also use audio features in the memory. We observe that using 16 audio frames, in the form of log-mel spectrograms, achieves slightly better performance compared

TABLE 3
State-of-the-Art Results With Different Architectures on ActivityNet

	# Frames†	2D/3D	mAP
IDT [49]	–	–	68.7
C3D [1]	10×16	3D	67.7
P3D [50]	20×16	3D	78.9
RRA [51]	*	2D	83.4
MARL [25]	25×1	2D	83.8
AdaFrame-10 (DPN)	6.83×1	2D	82.4
AdaFrame-10 (SlowFast)	5.44×8	3D	84.0

**RRA uses all frames decoded at 4fps; † A×B, represents uniformly sampled A snippets for each video, and there are B frames in each snippet.*

TABLE 4
Results of AdaFrame Using Different Global Memories on FCVID

Global Memory		Inference	
# Frames	Overhead	mAP	# Frames
0	0	77.9	8.40
V12	0.98	79.2	8.53
V32	2.61	80.2	8.24
V16	1.32	**80.2**	**8.21**
A16	1.14	78.1	8.88

Here, V denotes visual features and A denotes audio features.

TABLE 5
Results of AdaFrame With Different Reward Functions on FCVID

Reward function	mAP	# Frames
Prediction Reward	78.7	8.34
Prediction Transition Reward	78.9	8.31
Ours	**80.2**	**8.21**

The number frames used on average and the corresponding mAP.

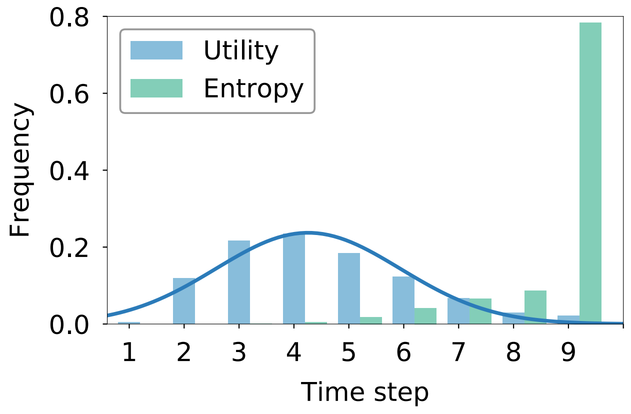


Fig. 8. Fraction of samples classified over time using utility and entropy.

to the model without using any memory, but it is not as good as visual global memory. This might due to the limited performance of audio features for video classification [15].

Reward Function. We introduce a reward function that encourages the model to be more confident when viewing more frames, as a proxy to measure whether the transition from the last time step is useful. We also compare with the following alternative reward functions: (1) Prediction Reward, which adopts the prediction score of the ground-truth class p_t^{gt} as reward; (2) Prediction Transition Reward, which uses the confidence margin $p_t^{gt} - p_{t-1}^{gt}$ as reward. Table 5 presents the results. We observe that our reward function and Prediction Transition Reward, both of which are margin-based, achieve better results than Prediction Reward that simply uses predictions of the current step. This confirms that encouraging the model to be more confident when more frames are observed provides useful feedback about whether selected frames are informative. In addition, our reward function, which introduces a margin between scores of the ground-truth class and other classes, also outperforms the Prediction Transition Reward.

Stop Criterion. To achieve adaptive inference, we use the outputs from the utility network, estimating the advantage of seeing more frames in the future, to determine whether to exit the model. Alternatively, one can compute the entropy of prediction scores, which measures the confidence of classifiers, to decide whether to stop inference. We also experimented with entropy as stop signals, but we found that predictions over time are not as smooth as estimated utilities, leading to high entropies in early stages and very low entropies in the end. As a result, adaptive inference cannot be achieved with entropy using different thresholds. Unlike entropies, utilities are encoded with future information since they are designed explicitly to estimate expected future rewards, resulting in smooth transitions over time. Fig. 8 visualizes the percentage of samples (frequency) that are classified over time using both utility and entropy. We can see the sample distribution using entropy over time is skewed.

TABLE 6
Comparisons Between AdaFrame and Alternative Methods in Terms of mAP, GFLOPs, Runtime, and Number of Parameters on FCVID

Method	GFLOPs	Runtime (seconds/video)	# Parameters (Million)	mAP
AvgPooling-25	195.5	0.566	44.6	80.0
LSTM-25	196	0.641	93.5	79.8
AdaFrame-10	75.1	0.416	98.2	80.2

TABLE 7
Comparisons of Different Approaches on Kinetics

	# Frames	GFLOPs	Top-1 Acc.
AvgPooling	10	657	75.2
AvgPooling-G	10	657.8	75.2
LSTM	10	657.2	75.1
LSTM-G	10	670	75.3
SCSampling	10	667.2	74.8
AdaFrame-10	6.19 ± 1.59	415 ± 107	75.4

Here, we use a single center crop for inference.

Runtime and Number of Parameters. We also report the runtime and the number of parameters of AdaFrame and compare with AvgPooling and LSTM. The results are summarized in Table 6. We measure runtime by seconds per video on a server with 2 Intel Xeon Processor E5-2689 v4 CPUs and 512G memory using a single NVIDIA V100 GPU. Since the number of frames selected by AdaFrame is instance-specific, we use a batch size of 1 during inference. Note that AvgPooling-25 can be much faster than LSTM-based approaches if 25 frames are run in parallel. A potential future direction is to explore alternative agent models to learn frame usage policy in one shot rather than making sequential decisions with LSTMs. In addition, compared to AvgPooling, LSTM-based approaches contain more parameters due to the operations in LSTM cells are dense matrix multiplications; recurrent networks can be further pruned to reduce footprint [52], [53].

4.4 Results on Trimmed Videos

While we mainly focus on untrimmed videos with a large number of redundant frames to save computation, we also evaluate our approach on the widely used Kinetics-400 dataset [45], which contains YouTube clips with an average duration of 10 seconds belonging to 400 categories. Following [5], we use $\sim 240K$ videos for training and $\sim 20K$ videos for evaluation. We adopt a SlowFast-8×8 network [5] as the backbone of AdaFrame, and compute global memory using spatially and temporally downsampled frames with MobileNetv2. We use a single center crop for inference and report frames used, GFLOPs and Top-1 accuracy in Table 7. We can see that AdaFrame can reduce the number of frames used from 10 to 6.19 on average, achieving the best trade-off between accuracy and computational cost. For SCSampling, we do not observe performance gain over the original model as in [9]; the reason might be our sampler is based on downsampled images and is weaker than the sampler in [9], which uses full resolution images and audio information as inputs.

5 Conclusion

We presented AdaFrame, a generic conditional computation framework, which adaptively allocates computational resources conditioned on input videos. In particular, AdaFrame learns frame usage policy on a per-video basis in order to select a small set of frames to make correct predictions to decrease computational cost. It contains a memory-augmented LSTM, in which a global memory module is attached to provide global context information. Optimized with policy search methods in reinforcement learning paradigm, AdaFrame learns to select which frames to use and predict future utilities. During inference, the predicted utility is used to decide whether to exit the model for adaptive inference. We conducted extensive experiments on two large-scale benchmarks, FCVID and ActivityNet, and we demonstrate qualitatively and quantitatively that AdaFrame can learn useful frame usage policy conditioned on inputs for fast video recognition. Future directions for research include to how to design dynamic inference strategies that can accurately predict videos containing multiple labels (sub-actions) at different temporal steps as well as how to explore alternative lightweight agents to replace LSTMs to learn frame usage policy.

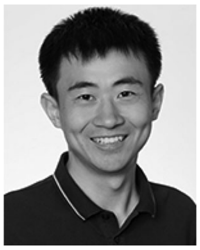
Acknowledgments

LSD is partially supported by the Intelligence Advanced Research Projects Activity (IARPA) via Department of Interior/Interior Business Center (DOI/IBC) contract number D17PC00345. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes not withstanding any copyright annotation thereon. Disclaimer: The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied of IARPA, DOI/IBC or the U.S. Government.

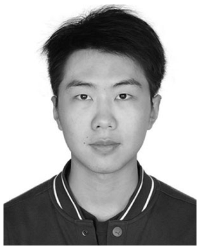
References

[1] D. Tran, L. Bourdev, R. Fergus, L. Torresani, and M. Paluri, “Learning spatiotemporal features with 3d convolutional networks,” in *Proc. IEEE Int. Conf. Comput. Vis.*, 2015, pp. 4489–4497.
[2] X. Wang, R. Girshick, A. Gupta, and K. He, “Non-local neural networks,” in *Proc. Conf. Comput. Vis. Pattern Recognit.*, 2018, pp. 7794–7803.
[3] K. Simonyan and A. Zisserman, “Two-stream convolutional networks for action recognition in videos,” in *Proc. 27th Int. Conf. Neural Inf. Process. Syst.*, 2014, pp. 568–576.
[4] L. Wang *et al.*, “Temporal segment networks for action recognition in videos,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 41, no. 11, pp. 2740–2755, Nov. 2019.
[5] C. Feichtenhofer, H. Fan, J. Malik, and K. He, “Slowfast networks for video recognition,” in *Proc. Int. Conf. Comput. Vis.*, 2019, pp. 6201–6210.
[6] C.-Y. Wu, M. Zaheer, H. Hu, R. Manmatha, A. J. Smola, and P. Krähenbühl, “Compressed video action recognition,” in *Proc. Conf. Comput. Vis. Pattern Recognit.*, 2018, pp. 6026–6035.
[7] B. Zhang, L. Wang, Z. Wang, Y. Qiao, and H. Wang, “Real-time action recognition with enhanced motion vector CNNs,” in *Proc. Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 2718–2726.
[8] M. Zolfaghari, K. Singh, and T. Brox, “Eco: Efficient convolutional network for online video understanding,” in *Proc. Eur. Conf. Comput. Vis.*, 2018, pp. 695–712.
[9] B. Korbar, D. Tran, and L. Torresani, “Scsampler: Sampling salient clips from video for efficient action recognition,” in *Proc. Int. Conf. Comput. Vis.*, 2019, pp. 6231–6241.
[10] Z. Wu, C. Xiong, C.-Y. Ma, R. Socher, and L. S. Davis, “Adaframe: Adaptive frame selection for fast video recognition,” in *Proc. Conf. Comput. Vis. Pattern Recognit.*, 2019, pp. 1278–1287.
[11] J. Y.-H. Ng, M. Hausknecht, S. Vijayanarasimhan, O. Vinyals, R. Monga, and G. Toderici, “Beyond short snippets: Deep networks for video classification,” in *Proc. Conf. Comput. Vis. Pattern Recognit.*, 2015, pp. 4694–4702.
[12] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in *Proc. Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 770–778.
[13] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi, “Inception-v4, inception-resnet and the impact of residual connections on learning,” in *Proc. 31st AAAI Conf. Artif. Intell.*, 2017, pp. 4278–4284.
[14] J. Carreira and A. Zisserman, “Quo vadis, action recognition? a new model and the kinetics dataset,” in *Proc. Conf. Comput. Vis. Pattern Recognit.*, 2017, pp. 4724–4733.
[15] Y.-G. Jiang, Z. Wu, J. Wang, X. Xue, and S.-F. Chang, “Exploiting feature and class relationships in video categorization with regularized deep neural networks,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 40, no. 2, pp. 352–364, Feb. 2018.
[16] F. C. Heilbron, V. Escorcia, B. Ghanem, and J. C. Niebles, “Activitynet: A large-scale video benchmark for human activity understanding,” in *Proc. Conf. Comput. Vis. Pattern Recognit.*, 2015, pp. 961–970.
[17] S. Yeung, O. Russakovsky, G. Mori, and L. Fei-Fei, “End-to-end learning of action detection from frame glimpses in videos,” in *Proc. Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 2678–2687.
[18] H. Fan, Z. Xu, L. Zhu, C. Yan, J. Ge, and Y. Yang, “Watching a small portion could be as good as watching all: Towards efficient video classification,” in *Proc. 27th Int. Joint Conf. Artif. Intell.*, 2018, pp. 705–711.
[19] Y. Chen, J. Li, H. Xiao, X. Jin, S. Yan, and J. Feng, “Dual path networks,” in *Proc. Adv. Conf. Neural Inf. Process. Syst.*, 2017, pp. 4467–4475.
[20] S. Xie, C. Sun, J. Huang, Z. Tu, and K. Murphy, “Rethinking spatiotemporal feature learning: Speed-accuracy trade-offs in video classification,” in *Proc. Eur. Conf. Comput. Vis.*, 2018, pp. 305–321.
[21] S. Ji, W. Xu, M. Yang, and K. Yu, “3D convolutional neural networks for human action recognition,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 35, no. 1, pp. 221–231, Jan. 2013.
[22] J. Lin, C. Gan, and S. Han, “TSM: Temporal shift module for efficient video understanding,” in *Proc. Int. Conf. Comput. Vis.*, 2019, 7082–7092.
[23] Y.-C. Su and K. Grauman, “Leaving some stones unturned: Dynamic feature prioritization for activity detection in streaming video,” in *Proc. Eur. Conf. Comput. Vis.*, 2016, pp. 783–800.
[24] S. Bhardwaj, M. Srinivasan, and M. M. Khapra, “Efficient video classification using fewer frames,” in *Proc. Conf. Comput. Vis. Pattern Recognit.*, 2019, pp. 354–363.
[25] W. Wu, D. He, X. Tan, S. Chen, and S. Wen, “Multi-agent reinforcement learning based frame sampling for effective untrimmed video recognition,” in *Proc. Int. Conf. Comput. Vis.*, 2019, pp. 6222–6231.
[26] Z. Liu *et al.*, “Teinet: Towards an efficient architecture for video recognition,” in *Proc. AAAI Conf. Artif. Intell.*, 2020, pp. 11669–11676.
[27] R. Gao, T.-H. Oh, K. Grauman, and L. Torresani, “Listen to look: Action recognition by previewing audio,” in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2020, pp. 10457–10467.
[28] Z. Shou *et al.*, “DMC-net: Generating discriminative motion cues for fast compressed video action recognition,” in *Proc. Conf. Comput. Vis. Pattern Recognit.*, 2019, pp. 1268–1277.
[29] X. Zhu, Y. Xiong, J. Dai, L. Yuan, and Y. Wei, “Deep feature flow for video recognition,” in *Proc. Conf. Comput. Vis. Pattern Recognit.*, 2017, pp. 4141–4150.
[30] B. Pan, W. Lin, X. Fang, C. Huang, B. Zhou, and C. Lu, “Recurrent residual module for fast inference in videos,” in *Proc. Conf. Comput. Vis. Pattern Recognit.*, 2018, pp. 1536–1545.
[31] P. Viola and M. J. Jones, “Robust real-time face detection,” *Int. J. Comput. Vis.*, vol. 57, pp. 137–154, 2004.
[32] S. Teerapittayanon, B. McDanel, and H. Kung, “Branchynet: Fast inference via early exiting from deep neural networks,” in *Proc. 23rd Int. Conf. Pattern Recognit.*, 2016, pp. 2464–2469.
[33] G. Huang, D. Chen, T. Li, F. Wu, L. van der Maaten, and K. Q. Weinberger, “Multi-scale dense convolutional networks for efficient prediction,” in *Proc. 6th Int. Conf. Learn. Representations*, 2018. [Online]. Available: https://openreview.net/forum?id=Hk2aImxAb

[34] M. McGill and P. Perona, “Deciding how to decide: Dynamic routing in artificial neural networks,” in *Proc. 34th Int. Conf. Mach. Learn.*, 2017, pp 2363–2372.
[35] M. Figurnov *et al.*, “Spatially adaptive computation time for residual networks,” in *Proc. Conf. Comput. Vis. Pattern Recognit.*, 2017, pp. 1790–1799.
[36] X. Wang, F. Yu, Z.-Y. Dou, and J. E. Gonzalez, “Skipnet: Learning dynamic routing in convolutional networks,” in *Proc. Eur. Conf. Comput. Vis.*, 2018, pp. 420–436.
[37] A. Veit and S. Belongie, “Convolutional networks with adaptive inference graphs,” in *Proc. Eur. Conf. Comput. Vis.*, 2018, pp. 3–18.
[38] Z. Wu *et al.*, “Blockdrop: Dynamic inference paths in residual networks,” in *Proc. Conf. Comput. Vis. Pattern Recognit.*, 2018, pp. 8817–8826.
[39] J. Collins, J. Sohl-Dickstein, and D. Sussillo, “Capacity and trainability in recurrent neural networks,” in *Proc. 5th Int. Conf. Learn. Representations*, 2017. [Online]. Available: https://openreview.net/forum?id=BydARw9ex
[40] D. Yogatama *et al.*, “Memory architectures in recurrent neural network language models,” in *Proc. 6th Int. Conf. Learn. Representations*, 2018. [Online]. Available: https://openreview.net/forum?id=SkFqf0lAZ
[41] A. Vaswani *et al.*, “Attention is all you need,” in *Proc. Advances Neural In. Process. Syst.*, 2017, pp. 5998–6008.
[42] V. Mnih, N. Heess, A. Graves, and K. Kavukcuoglu, “Recurrent models of visual attention,” in *Proc. 27th Int. Conf. Neural Inf. Process. Syst.*, 2014, pp. 2204–2212.
[43] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: MIT Press, 1998.
[44] K. Soomro, A. R. Zamir, and M. Shah, “UCF101: A dataset of 101 human actions classes from videos in the wild,” *CoRR*, vol. abs/1212.0402, 2012. [Online]. Available: http://arxiv.org/abs/1212.0402
[45] W. Kay *et al.*, “The kinetics human action video dataset,” *CoRR*, vol. abs/1705.06950, 2017. [Online]. Available: http://arxiv.org/abs/1705.06950
[46] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, “MobileNetV2: Inverted residuals and linear bottlenecks,” in *Proc. Conf. Comput. Vis. Pattern Recognit.*, 2018, pp. 4510–4520.
[47] A. G. Howard *et al.*, “MobileNets: Efficient convolutional neural networks for mobile vision applications,” *CoRR*, vol. abs/1704.04861, 2017. [Online]. Available: http://arxiv.org/abs/1704.04861
[48] J. F. Gemmeke *et al.*, “Audio set: An ontology and human-labeled dataset for audio events,” in *Proc. IEEE Int. Conf. Acoust. Speech Signal Process.*, 2017, pp. 776–780.
[49] H. Wang and C. Schmid, “Action recognition with improved trajectories,” in *Proc. Int. Conf. Comput. Vis.*, 2013, pp. 3551–3558.
[50] Z. Qiu, T. Yao, and T. Mei, “Deep quantization: Encoding convolutional activations with deep generative model,” in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2017, pp. 4085–4094. [Online]. Available: http://doi.ieeecomputersociety.org/10.1109/CVPR.2017.435
[51] C. Zhu *et al.*, “Fine-grained video categorization with redundancy reduction attention,” in *Proc. Eur. Conf. Comput. Vis.*, 2018, pp. 136–152.
[52] S. Zhang and B. C. Stadie, “One-shot pruning of recurrent neural networks by jacobian spectrum evaluation,” in *Proc. 8th Int. Conf. Learn. Representations*, 2020. [Online]. Available: https://openreview.net/forum?id=r1e9GCNKvH
[53] S. Narang, G. Diamos, S. Sengupta, and E. Elsen, “Exploring sparsity in recurrent neural networks,” in *Proc. 5th Int. Conf. Learn. Representations*, 2017. [Online]. Available: https://openreview.net/forum?id=BylSPv9gx



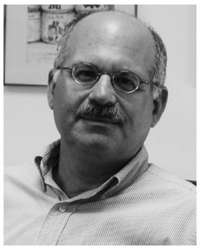
Zuxuan Wu received the PhD degree in computer science from the University of Maryland, College Park, in 2020. His research interests include computer vision and deep learning. His research has been recognized by a 2018 Snap Inc. fellowship and a 2019 Microsoft Research fellowship.



Hengduo Li received the BS degree from Fudan University, Shanghai, China, in 2018. He is currently working toward the PhD degree at the Department of Computer Science, University of Maryland, College Park, MD.His research interests include deep learning and computer vision, including object detection and video understanding.

Caiming Xiong received the BS and MS degrees in computer science from the Huazhong University of Science and Technology (HUST), in 2005 and 2007, respectively, and the PhD degree at the Department of Computer Science and Engineering, University at Buffalo, SUNY, in 2014. Currently, he is the senior director of AI Research at Salesforce. During his career, he has co-authored more than 80 papers in deep learning, natural language processing, computer vision, and theoretical machine learning. He has won an Outstanding Paper Award at the Annual Meeting of the Association for Computational Linguistics (ACL) in 2019, and a Best Paper Award from the European Conference on Data Mining (ECDM) in 2012.

Yu-Gang Jiang received the PhD degree in computer science from the City University of Hong Kong, in 2009 and worked as a postdoctoral research scientist with Columbia University, New York during 2009-2011. He is currently a professor and dean with the School of Computer Science, Fudan University, Shanghai, China. His research interests include the areas of multimedia, computer vision and AI security. His work has led to many awards, including the inaugural ACM China Rising Star Award, 2015 ACM SIGMM Rising Star Award, and the Research Award for excellent young scholars from NSF China.



Larry S. Davis (Fellow, IEEE) received the BA degree from Colgate University, in 1970, and the MS and PhD degrees in computer science from the University of Maryland, in 1974 and 1976, respectively. From 1977 to 1981, he was an assistant professor with the Department of Computer Science, University of Texas, Austin. He returned to the University of Maryland as an associate professor, in 1981. From 1985 to 1994, he was the director of the University of Maryland Institute for Advanced Computer Studies. From 1999 to 2012, he was the chair of the Computer Science Department in the institute. He is currently a professor with the institute and with the Computer Science Department. He is a fellow of the ACM, and the IAPR.