Long Short-Term Transformer for Online Action Detection

Anonymous Author(s)
Affiliation
Address
email

Abstract

In this paper, we present Long Short-term TRansformer (LSTR), a new temporal modeling algorithm for online action detection, by employing a long- and short-term memories mechanism that is able to model prolonged sequence data. It consists of a LSTR encoder that is capable of dynamically exploiting coarse-scale historical information from an extensively long time window (*e.g.*, 2048 long-range frames of up to 8 minutes), together with a LSTR decoder that focuses on a short time window (*e.g.*, 32 short-range frames of 8 seconds) to model the fine-scale characterization of the ongoing event. Compared to prior work, LSTR provides an effective and efficient mean to model long videos with less heuristic algorithm design. LSTR achieves significantly improved results on standard online action detection benchmarks over the existing state-of-the-art approaches. Extensive empirical analysis validates the setup of the long- and short-term memories and the design choices of LSTR.

1 Introduction

Given an incoming stream of video frames, online action detection [12] is concerned with the task of classifying what is happening at each frame without seeing the future. It is a causal reasoning problem different from offline action recognition where the entire video snippets are present. In this paper, we tackle the online action detection problem by presenting a new temporal modeling algorithm that is able to capture temporal correlations on prolonged sequences up to 8 minutes long, while retaining fine granularity of the event in the representation. This is achieved by modeling activities at different temporal scales that is necessary to capture the variety of events, some occurring in brief bursts, other manifest in slowly varying trends.

Specifically, we propose a new model, named *Long Short-term TRansformer (LSTR)*, to jointly model the long- and short-term temporal dependencies. The LSTR has two advantages over previous work. 1) Storing history in the "plaintext" manner avoids the difficulty of learning with recurrent models [16, 44, 26] (no backpropagatoin through time, BPTT, needed); the model can directly attend to any useful frames in the long-term memory. 2) Separating long- and short-term memories allows us to simultaneously conduct focused modeling of short-term information while extracting well-informed compressed knowledge from the long-term history. This means we can compress the long-term history without worrying about losing important historical information for making classification at the current time.

As shown in Fig. 1, we explicitly divide the entire history into the long- and short-term memories and build our model with an encoder-decoder architecture. Specifically, the *LSTR encoder* compresses and abstracts the long-term memory into a fixed length of latent representations, and the *LSTR decoder* uses a short window of transient frames that perform the self-attention operation in addition to the cross-attention to the exploited token embeddings from the LSTR encoder. In the LSTR encoder,

Submitted to 35th Conference on Neural Information Processing Systems (NeurIPS 2021). Do not distribute.

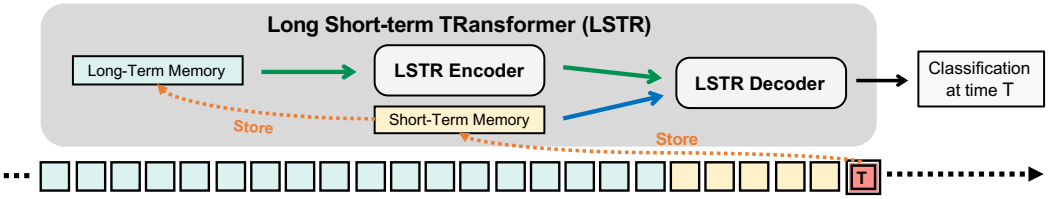

Figure 1: **Overview of our Long Short-term TRansformer (LSTR)**. Given a live streaming video, LSTR sequentially identifies the actions happening in each incoming frame by using an encoder-decoder architecture, without future context. The dashed brown arrows indicate the data flow of the long- and short-term memories following the first-in-first-out (FIFO) logic. (Best viewed in color.)

an extended temporal support becomes beneficial in dealing with untrimmed, streaming videos by devising two-stage memory compression, which is shown to be computationally efficient in both training and inference. Our overall long short-term Transformer architecture gives rise to an effective and efficient representation for modeling prolonged time sequence data.

To evaluate the effectiveness of our model, we validate LSTR on three standard datasets (THUMOS'14 [27], TVSeries [12], and HACS Segment [66]), which have distinct natures of video lengths (from a few seconds to tens of minutes). Experimental results show that LSTR significantly outperforms all state-of-the-art methods for online action detection, and the ablation studies further demonstrate LSTR's superior performance for modeling long video sequences.

2 Related Work

Online Action Detection. Temporal action localization targets detecting the temporal boundaries of action instances after observing the entire video [48, 62, 21, 46, 68, 3, 34, 35, 33]. Emerging real-world applications, such as robotic perception, require causal processing, where only past and current frames are used for detecting actions of the present. There has been a wealthy body of recent work on this online action detection problem [12]. RED [20] uses a reinforcement loss to encourage recognizing actions as early as possible. TRN [63] models greater temporal context by simultaneously performing online action detection and anticipation. IDN [17] learns discriminative features and accumulates only relevant information for the present. LAP-Net [42] proposes an adaptive sampling strategy to obtain optimal features. PKD [67] transfers knowledge from offline to online models using curriculum learning. As with early action detection [25, 36, 65], Shou *et al*. [47] focus on online detection of action start (ODAS). StartNet [22] decomposes ODAS into two stages and learns with policy gradient and WOAD [23] uses weakly-supervised learning with video-level labels.

Temporal/Sequence Modeling. Modeling temporal information, especially from long-term context, is important but challenging for many video understanding tasks [58, 41, 59]. Prior work on action recognition mostly relies on heuristic sub-sampling (typically 3 to 7 video frames) for more feasible training [64, 55, 19, 37, 50] or recurrent neural networks [26, 8] that maintain the entire history in a summary state, at the cost of losing track of specifics for the individual time steps in the past. 3D CNNs [6, 53, 60] are also widely used to perform spatio-temporal feature modeling. However, they are limited by the receptive field hence unable to model long-range temporal context. Wu *et al*. [58] propose long-term feature bank for generic video understanding, which stores various supportive information, such as object and scene features, as context references. These features are usually pre-computed independently, having even no sense of position or order information, thus cannot provide an effective representation of the full context. On the other hand, most of above work does not explicitly integrate the long- and short-term features, but usually combine them with simple mechanisms, such as pooling and concatenation. Recent findings in cognitive science for the underlying mechanisms of working memory and attention [40, 10, 7, 31] have also shed light on the design principle for modeling long-term information with attention [57, 58]. Comparing with RNNs, the attention mechanism is able to maintain long-term information at fast inference speed [54].

Transformers for Action Understanding. Transformers have achieved breakthrough success in NLP [43, 13] and are adopted in computer vision for image recognition [15, 51] and object detection [5]. Recent research papers exploit Transformers for temporal modeling in videos, such as action recognition [39, 45, 32, 2, 1] and temporal action localization [38, 49], and achieve state-of-the-art

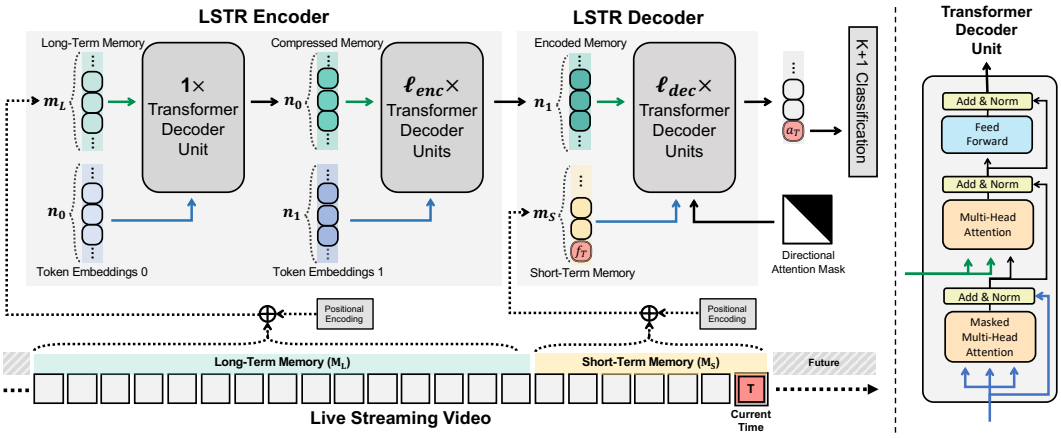

Figure 2: **Visualization of our Long Short-Term Transformer (LSTR)**, which is formulated in an encoder-decoder manner. Specifically, the LSTR encoder encodes the long-term memory $\mathbf{M}_L$ of size m_L to compressed latent features, and the LSTR decoder takes this encoded memory and references related information with the short-term memory $\mathbf{M}_S$ of size m_S for action recognition of the current. The LSTR encoder and decoder are built with Transformer decoder units [54], each taking input tokens (dark green arrows) and output tokens (dark blue arrows) as inputs. During inference, LSTR processes every incoming frame in an online manner, absent future context. (Best viewed in color.)

results in the common benchmarks. However, most of these work only performs temporal modeling with relatively short video clips due to their computational demanding and limited modeling capability. Several work [11, 4] focuses on designing transformers to model long-term dependencies, but how to aggregate the long- and short-term information is still not well-explored today [28].

3 Long Short-Term Transformer

Given a live streaming video, our goal is to identify the actions performed in each video frame using only past and current observations. Future information is *not* accessible during inference. Formally, a streaming video at time t is represented by a batch of τ past frames $\mathbf{I}^t = \{I_{t-\tau}, \cdots, I_t\}$, which reads "$I$ up to time t." The online action detection system receives $\mathbf{I}^t$ as inputs, and classifies the action category $\hat{y}_t$ belonging to one of $(K+1)$ classes, $\hat{y}_t \in \{0, 1, \cdots, K\}$, ideally using the posterior probability $P(\hat{y}_t = k|\mathbf{I}^t)$, where $k = 0$ denotes the probability that no event is occurring at frame t. We design our method by assuming that there is a pre-trained feature extractor [55] that processes each video frame I_t into a feature vector $\mathbf{f}_t \in \mathbb{R}^C$ of C dimensions[1]. These vectors form a $(t \times C)$-dimensional temporal sequence that serves as the input of our method.

3.1 Overview

Our method is based on the intuition that frames observed recently provide precise information about the ongoing action instance while frames over an extended period offer contextual references for actions that are potentially happening now. We propose Long Short-term TRansformer (LSTR) in an explicit encoder-decoder manner, as shown in Fig. 2. First, the feature vectors of m_L frames in the distant past are stored in a *long-term memory*. A *short term memory* stores features of m_S recent frames ($m_S \ll m_L$). The LSTR encoder compresses and abstracts features in long-term memory to a encoded latent representation of n_1 vectors. The LSTR decoder then queries the encoded long-term memory with the short-term memory for decoding, leading to the action prediction $\hat{y}_t$ This design follows the line of thought in combining long- and short-term information for action understanding [58, 55, 14], but addresses several key challenges to effectively and efficiently achieve this goal, thanks to the flexibility of modeling brought by the recently emerged Transformers [54].

[1] In practice, some feature extractors [52] take consecutive frames to produce one feature vector. Nonetheless, it is still temporally "centered" on a single frame. Thus we use the single frame notation here for simplicity.

3.2 Long- and Short-Term Memories

We store the streaming input of feature vectors in two consecutive memories. The first memory is called short-term memory, which stores only a small number of frames that are recently observed. We implement it with a first-in-first-out (FIFO) queue of m_S slots. At time T, it stores the feature vectors as $\mathbf{M}_S = \{\mathbf{f}_T, \cdots, \mathbf{f}_{T-m_S+1}\}$. When a frame becomes "older" than m_S time steps, it graduates from $\mathbf{M}_S$ and enters into the long-term memory, which is implemented with another FIFO queue of m_L slots. The long term memory stores $\mathbf{M}_L = \{\mathbf{f}_{T-m_S}, \cdots, \mathbf{f}_{t-m_S-m_L+1}\}$. The long-term memory serves as input memory to the LSTR encoder and the short-term memory serves as the queries for the LSTR decoder. In practice, the long-term memory stores much longer time span than the short-term memory ($m_S \ll m_L$). A typical example of choice is $m_L = 2048$, which represents 512 seconds worth of video contents with 4 frames per second (FPS) sampling rate, and $m_S = 32$ representing 8 seconds. We add a sinusoidal positional encoding $\mathbf{s}$ [54] to each frame feature in the memories *relative* to current time T (*i.e.*, the frame at $T - \tau$ receives a positional embedding of $\mathbf{s}_\tau$).

3.3 LSTR Encoder

The LSTR encoder aims at encoding the long-term memory of m_L feature vectors into a latent representation that LSTR can use for decoding useful temporal context. This task requires large capacity in capturing the relations and temporal context among a span of hundreds or even thousands of frames. Prior work on modeling long-term dependencies for action understanding relies on heuristic temporal sub-sampling [58, 55] or recurrent networks [14] to make more feasible training, at the cost of losing specific information of each time step. Attention-based architectures, such as Transformer [54], have recently been shown promising for similar tasks that require long-range temporal modeling [45]. A straightforward choice for LSTR encoder would be to use the Transformer encoder based on self-attention. However, its time complexity, $O(m_L^2 C)$, grows quadratically with the memory sequence length m_L. This limits our ability to have a long-term memory with sufficient length to cover long videos. Though recent work has been exploring linear complexity Transformer [56], the linear growth still makes having long memory rather computationally heavy. In LSTR, we propose to use a two-stage memory compression based on Transformer decoder units [54] to achieve sub-linear complexity and effective memory encoding.

The Transformer decoder unit [54] takes two sets of inputs. The first set includes a fixed number of n learnable output tokens $\boldsymbol{\lambda} \in \mathbb{R}^{n \times C}$, where C is the embedding dimension. The second set includes another m input tokens $\boldsymbol{\theta} \in \mathbb{R}^{m \times C}$, and m can be a rather large number. It first applies one layer of multi-head self-attention on $\boldsymbol{\lambda}$. The outputs $\boldsymbol{\lambda}'$ are then used as queries in an "QKV cross-attention" operation and the input embeddings $\boldsymbol{\theta}$ serve as key and value. The two steps can be written as

$$\boldsymbol{\lambda}' = \text{SelfAttn}(\boldsymbol{\lambda}) = \text{Softmax}(\frac{\boldsymbol{\lambda} \cdot \boldsymbol{\lambda}^T}{\sqrt{C}})\boldsymbol{\lambda} \;\; \textit{and} \;\; \text{CrossAttn}(\sigma(\boldsymbol{\lambda}'), \boldsymbol{\theta}) = \text{Softmax}(\frac{\sigma(\boldsymbol{\lambda}') \cdot \boldsymbol{\theta}^T}{\sqrt{C}})\boldsymbol{\theta},$$

where $\sigma : \mathbb{R}^{n \times C} \to \mathbb{R}^{n \times C}$ denotes the intermediate layers between the two attention operations. One appealing property of this design is that it transforms the $m \times C$ dimensional input tokens into output tokens of $n \times C$ dimensions in a time complexity of $O(n^2 C + nmC)$. When $n \ll m$, the time complexity becomes linear to m, which makes it an ideal candidate for compressing the long-term memory. This property is also utilized in [28] to efficiently process large volume inputs, such as image pixels and point clouds.

Two-Stage Memory Compression. Stacking multiple Transformer decoder units on the long-term memory, as in [54], can form a memory encoder with linear complexity with respect to the memory size m_L. However, running the encoder at each timestep can still be time consuming. We further reduce the time complexity with a two-stage memory compression design. The first stage has one Transformer decoder unit with n_0 output tokens. Its input tokens are the whole long-term memory of size m_L. The outputs of the first stage are used as the input tokens to the second stage, which has ℓ_{enc} stacked Transformer decoder units and n_1 output tokens. Then, the long-term memory of size $m_L \times C$ is compressed into a latent representation of size $n_1 \times C$, which can then be efficiently queried in the LSTR decoder later. This two-stage memory compression design is illustrated in Fig. 2.

Compared to an $(1 + \ell_{enc})$-layer Transformer encoder with $O(m_L^2(1 + \ell_{enc})C)$ time complexity or stacked Transformer decoder units with n output-tokens having $O(n(n + m_L)(1 + \ell_{enc})C)$ time complexity, the proposed LSTR decoder has complexities of $O(n_0 m_L C + n_0^2 C + (n_1 n_0 + n_1^2)\ell_{enc} C)$. Because both n_0 and n_1 are much smaller than m_L and ℓ_{enc} is usually larger than 1, using the two-stage memory compression in LSTR encoder could be more efficient. In Sec. 3.6, we will demonstrate

that it further enables us to achieve $O(n_0(m_L + C) + n_0^2 C + (n_1 n_0 + n_1^2)\ell_{enc} C)$ time complexity in inference of the encoder on streaming inputs, which becomes sub-linear with respect to m_L. In Sec. 4.5, we empirically found this design also leads to better performance for online action detection

3.4 LSTR Decoder

Short-term memory contains informative features for classifying action on the latest timestep. The LSTR decoder uses the short-term memory as queries to retrieve useful information from the encoded long-term memory produced by the LSTR encoder. The LSTR decoder is formed by stacking ℓ_{dec} layers of Transform decoder units. It takes the outputs of the LSTR encoder as input tokens and the m_S feature vectors in the short-term memory as output tokens. It outputs m_S probability vectors $\{\mathbf{p}_T, \cdots, \mathbf{p}_{T-m_S+1}\} \in [0,1]^{K+1}$, each $\mathbf{p}_t$ representing the predicted probability distribution of K action categories and one "background" class at time t. During inference, we only take the probability vector from the output token corresponding to the current time $\mathbf{p}_T$ for classification result. However, having the additional outputs on the older frames allows the model to leverage more supervision signals during training we will describe below.

3.5 Training LSTR

LSTR can be trained without temporal unrolling and Backpropagation Through Time (BPTT) as in LSTM [26], which is a common property of Transformers [54]. We construct each training sample by randomly sampling an ending time T and filling the two memories by tracing back in time for $m_S + m_L$ frames. We use the empirical cross entropy loss between the predicted probability distribution $\mathbf{p}_T$ at time T and the ground truth action label $y_T \in \{0, 1, \ldots, K\}$ as

$$L(y_T, \mathbf{p}_T; T) = -\sum_{k=0}^{K} \delta(k - y_T) \log p_T^k, \tag{1}$$

where p_T^k is the k-th element of the probability vector $\mathbf{p}_T$, predicted on the latest frame at T. Additionally, we add a *directional attention mask* [54] to the short-term memory so that any frame in the short-term memory can only depend on its previous frames. In this way, we can make prediction on all frames in the short-term memory as if they are the latest ones. Thus we can provide supervision on every frame in the short-term memory, and the complete loss function $\mathcal{L}$ is then

$$\mathcal{L}_T = \sum_{t=T-ms+1}^{T} L(y_t, \mathbf{p}_t; T), \tag{2}$$

where $\mathbf{p}_t$ denotes the prediction from the output token corresponding to time t.

3.6 Online Inference with LSTR

During the online inference, the video frame features are streamed to the model as time passes. Running LSTR's long-term memory encoder from scratch for each frame results in a time complexity of $O(n_0 m_L C + n_0^2 C + (n_1 n_0 + n_1^2)\ell_{enc} C)$. However, at each time step, there is only one new video frame to be updated. We show it is possible to achieve even more efficient online inference by storing the intermediate results. In the Transformer decoder unit of the first memory compression stage, the cross attention is instantiated as

$$\mathrm{CrossAttn}(\mathbf{q}_i, \{\mathbf{f}_{T-\tau} + \mathbf{s}_\tau\}) = \sum_{\tau=m_S}^{m_S+m_L-1} \frac{\exp((\mathbf{f}_{T-\tau} + \mathbf{s}_\tau) \cdot \mathbf{q}_i / \sqrt{C})}{\sum_{\tau=m_S}^{m_S+m_L-1} \exp((\mathbf{f}_{T-\tau} + \mathbf{s}_\tau) \cdot \mathbf{q}_i / \sqrt{C})} \cdot (\mathbf{f}_{T-\tau} + \mathbf{s}_\tau), \tag{3}$$

where the index $\tau = T - t$ is the relative position of a frame t in the long-term memory to the latest time T. This calculation depends on the un-normalized attention weight matrix $\mathbf{A} \in \mathbb{R}^{m_L \times n_0}$, with elements $a_{\tau i} = (\mathbf{f}_{T-\tau} + \mathbf{s}_\tau) \cdot \mathbf{q}_i$. $\mathbf{A}$ can be decomposed into the sum of two matrices $\mathbf{A}^f$ and $\mathbf{A}^s$. We have their elements as $a^f_{\tau i} = \mathbf{f}_{T-\tau} \cdot \mathbf{q}_i$ and $a^s_{\tau i} = \mathbf{s}_\tau \cdot \mathbf{q}_i$. The queries after the first self-attention operation, $\mathbf{Q} = [\mathbf{q}_1, \ldots, \mathbf{q}_{n_0}]$, and position embedding $\mathbf{s}_\tau$ are fixed during inference. Thus the matrix $\mathbf{A}^s$ can be pre-computed and used for every incoming frame. We additionally maintain a FIFO queue of vectors $\mathbf{a}_t = \mathbf{Q}^\top \mathbf{f}_t$ of size m_L. $\mathbf{A}^f$ at any time step T can be obtained by stacking all vectors currently in this queue. Updating this queue at each time step requires $O(n_0 C)$ time complexity for the matrix-vector product. Now we can obtain the matrix $\mathbf{A}$ with only $n_0 \times m_L$ additions by adding $\mathbf{A}^s$ and $\mathbf{A}^f$ together, instead of $n_0 \times m_L \times C$ multiplications and additions using Eq. (3). This means the amortized time complexity of running the LSTR encoder can be further reduced to $O(n_0(m_L + C) + n_0^2 C + (n_0 n_1 + n_1^2)\ell_{enc} C)$ for online inference. Since C is usually above 1024 [24], this is a considerable reduction of runtime cost. We show our runtime in the Appendix.

4 Experiments

4.1 Datasets

We evaluate our model on three publicly-available datasets: **THUMOS'14** [27], **TVSeries** [12] and **HACS Segment** [66]. THUMOS'14 includes over 20 hours of sports video annotated with 20 actions. We follow prior work [63, 17] and train on the validation set (200 untrimmed videos) and evaluate on the test set (213 untrimmed videos). TVSeries contains 27 episodes of 6 popular TV series, totaling 16 hours of video. The dataset is annotated with 30 realistic, everyday actions (*e.g.*, open door). HACS Segment is a large-scale dataset of web videos. It contains 35,300 untrimmed videos over 200 human action classes for training and 5,530 untrimmed videos for validation.

4.2 Settings

Feature Encoding. We follow the experimental settings of state-of-the-art methods [63, 17]. Specifically, we extract video frames at 24 FPS and set the video chunk size to 6. Decisions are made at the chunk level, and thus performance is evaluated every 0.25 seconds. For feature encoding, we adopt two-stream network [55] with ResNet-50 backbone, and use the open-source toolbox [9]. We experiment with feature extractors pretrained on two datasets: ActivityNet and Kinetics. The visual features are extracted at the global pooling layer from the central frame of each chunk, and the motion features are extracted from pre-computed stacked optical flow fields between 6 consecutive frames. The visual and motion features are concatenated along the channel dimension as the final feature $\mathbf{f}$.

Implementation Details. We implemented our proposed model in PyTorch, and performed all experiments on a system with 8 Nvidia V100 graphics cards. To learn model weights, we used the Adam [30] optimizer with weight decay 5×10^{-5}. The learning rate was linearly increased from zero to 5×10^{-5} in the first $2/5$ training iterations and was reduced to zero according to a cosine function. Our models were optimized using batch size of 16, and the training was terminated after 25 epochs.

Evaluation Protocols. We follow prior work and use per-frame **mean average precision (mAP)** to evaluate the performance of online action detection. We also use per-frame **calibrated average precision (cAP)** [12] that was proposed for TVSeries to correct the imbalance between positive and negative samples, $cAP = \sum_k cPrec(k) * I(k)/P$, where $cPrec = TP/(TP + FP/w)$, $I(k)$ is 1 if frame k is a true positive, P is the number of true positives, and w is the negative and positive ratio.

4.3 Comparison with the State-of-the-art Methods

Table 1: **Online action detection results** on THUMOS'14 and TVSeries, comparing LSTR and the state-of-the-art in mAP (%) LSTR outperforms the state-of-the-art methods on THUMOS'14 by 3.7% and 2.6% in mAP and on TVSeries by 3.1% and 2.7% in cAP, by using ActivityNet and Kinetics pretrained features, respectively. *Results are reproduced by us using their papers' default settings.

(a) **Results on ActivityNet pretrained features**

	Input Features	THUMOS'14	TVSeries
		mAP (%)	mcAP (%)
CDC [46]	ActivityNet	44.4	-
RED [20]		45.3	79.2
TRN [63]		47.2	83.7
FATS [29]		51.6	81.7
IDN [17]		50.0	84.7
LAP [42]		53.3	85.3
TFN [18]		55.7	85.0
LFB* [58]		61.6	84.8
LSTR (ours)		**65.3**	**88.1**

(b) **Results on Kinetics pretrained features**

	Input Features	THUMOS'14	TVSeries
		mAP (%)	mcAP (%)
FATS [29]	Kinetics	59.0	84.6
IDN [17]		60.3	86.1
TRN [63]		62.1	86.2
PKD [67]		64.5	86.4
WOAD [23]		67.1	-
LFB* [58]		64.8	85.8
LSTR (ours)		**69.5**	**89.1**

We compare LSTR against the state-of-the-art methods [63, 17, 23] on THUMOS'14, TVSeries, and HACS Segment. Specifically, on THUMOS'14 and TVSeries, we implement LSTR with the long- and short-term memories of 512 and 8 seconds, respectively. On HACS Segment, we reduce the long-term memory to 256 seconds, considering that its videos are strictly shorter than 4 minutes. For LSTR, we implement the two-stage memory compression using two Transformer decoder units. We set the tokens with $n_0 = 16$ and $n_1 = 32$ and the Transformer layers with $\ell_{enc} = 2$ and $\ell_{dec} = 2$.

Table 2: **Online action detection results when only portions of videos are considered** in cAP (%) on TVSeries (*e.g.*, 80%-90% means only frames of this range of action instances were evaluated).

	Input Features	Portion of Video									
		0%-10%	10%-20%	20%-30%	30%-40%	40%-50%	50%-60%	60%-70%	70%-80%	80%-90%	90%-100%
CNN [12]	ActivityNet	61.0	61.0	61.2	61.1	61.2	61.2	61.3	61.5	61.4	61.5
RNN [12]		63.3	64.5	64.5	64.3	65.0	64.7	64.4	64.4	64.4	64.3
TRN [63]		78.8	79.6	80.4	81.0	81.6	81.9	82.3	82.7	82.9	83.3
IDN [17]		80.6	81.1	81.9	82.3	82.6	82.8	82.6	82.9	83.0	83.9
TFN [18]		83.1	84.4	85.4	85.8	87.1	88.4	87.6	87.0	86.7	85.6
LSTR (ours)		**83.6**	**85.0**	**86.3**	**87.0**	**87.8**	**88.5**	**88.6**	**88.9**	**89.0**	**88.9**
IDN [17]	Kinetics	81.7	81.9	83.1	82.9	83.2	83.2	83.2	83.0	83.3	86.6
PKD [67]		82.1	83.5	86.1	87.2	88.3	88.4	89.0	88.7	88.9	87.7
LSTR (ours)		**84.4**	**85.6**	**87.2**	**87.8**	**88.8**	**89.4**	**89.6**	**89.9**	**90.0**	**90.1**

THUMOS'14. We compare LSTR with recent work on THUMOS'14, including methods that use 3D ConvNets [46] and RNNs [61, 17, 23], reinforcement learning [20], and curriculum learning [67]. Table 1 shows that LSTR significantly outperforms the the state-of-the-art methods [58, 23] by 3.7% and 2.4% in terms of mAP using ActivityNet and Kinetics pretrained features, respectively.

TVSeries. Table 1 shows the online action detection results that LSTR outperforms the state-of-the-art methods [18, 67] by 3.1% and 2.7% in terms of cAP using ActivityNet and Kinetics pretrained features, respectively. Following prior work [12], we also investigate LSTR's performance at different action stages by evaluating each decile (ten-percent interval) of the video frames separately. Table 2 shows that LSTR outperforms existing methods at every stage of action instances.

HACS Segment. LSTR achieves 82.6% on HACS Segment in term of mAP using Kinetics pretrained features. Note that HACS Segment is a new large-scale dataset with only a few previous results. LSTR outperforms existing methods RNN [26] (77.6%) by 5.0% and TRN [63] (78.9%) by 3.7%.

4.4 Design Choices of Long- and Short-Term Memories

Table 3: **Results of LSTR using downsampled long-term memory** on THUMOS'14 in mAP (%). In particular, we use long-term memory as 512 seconds and short-term memory as 8 seconds.

Temporal Stride	1	2	4	8	16	32	64	128
LSTR	69.5	69.5	69.5	69.2	68.7	67.3	66.6	65.9

We experiment for design choices of long- and short-term memories. Unless noted otherwise, we use THUMOS'14, which contains various video lengths, and Kinetics pretrained features.

Lengths of long- and short-term memories. We first analyze the effect of different lengths of long-term m_L and short-term m_S memory. In particular, we test $m_S \in \{4, 8, 16\}$ seconds with m_L starting from 0 second (no long-term memory). Note that we choose the max length (1024 seconds for THUMOS'14 and 256 seconds for HACS Segment) to cover length of 98% videos, and do not have proper datasets to test longer m_L. Fig. 3 shows that LSTR is beneficial from larger m_L in most cases. In addition, when m_L is short (≤ 16 in our cases), using larger m_S obtains better results and when m_L is sufficient (≥ 32 in our cases), increasing m_S does not always guarantee better performance.

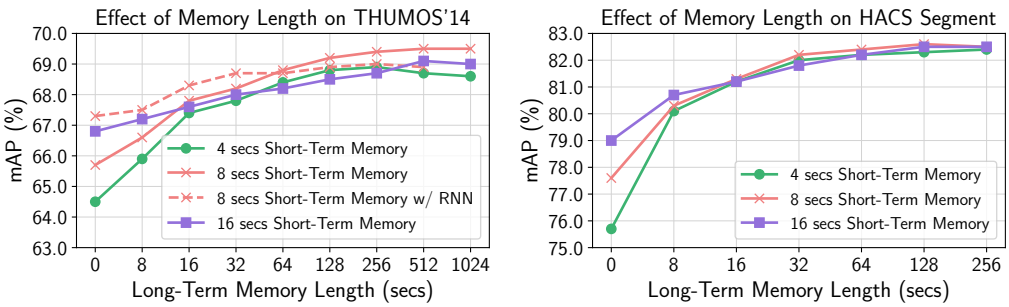

Figure 3: **Effect of using different lengths of long- and short-term memories**.

Can we downsample long-term memory? We implement LSTR with m_S as 8 seconds and m_L as 512 seconds, and test the effect of downsampling long-term memory. Table 3 shows the results that downsampling with strides smaller than 4 does not cause performance drop, but more aggressive strides dramatically decrease the detection accuracy. Note that, when extracting frame features in 4 FPS, both LSTR encoder ($n_0 = 16$) and downsampling with stride 128 compress the long-term memory to 16 features, but LSTR achieves much better performance (69.5% vs. 65.9% in mAP). This demonstrates the effectiveness of our "adaptive compression" comparing to heuristics downsampling.

Can we compensate reduced memory length with RNN? We note that LSTR's performance notably decreases when it can only access to very limited memory (*e.g.*, $m_L + m_S \leq 16$ seconds). Here we test if RNN can be used to compensate the LSTR's reduced memory or even fully replace the LSTR encoder. We implement LSTR using $m_S = 8$ seconds with an extra Gated Recurrent Unit (GRU) [8] (this structure is visualized in the Appendix) to capture all history outside the long- and short-term memory. The dashed line in Fig. 3 shows the results. Plugging-in RNNs indeed improves the performance when m_L is small, but when m_L is large (*i.e.* > 64 seconds), it does not improve the accuracy anymore. Note that RNNs are not used in any other experiments in this paper.

4.5 Design Choices of LSTR

Table 4: **Results of different designs of the LSTR encoder and decoder**. The length of short-term memory m_S is set to 8 seconds. "TR" denotes Transformer. The last row is our best performance.

LSTR Encoder	LSTR Decoder	Length of Long-Term Memory m_L (secs)							
		8	16	32	64	128	256	512	1024
N/A	TR Encoder	65.7	66.8	67.1	67.2	67.3	66.8	66.5	66.2
N/A	TR Decoder	66.5	67.3	67.7	68.1	68.3	67.9	67.0	66.5
TR Encoder	TR Decoder	65.9	66.4	66.7	67.4	67.5	67.2	67.0	66.6
TR Decoder	TR Decoder	66.1	67.1	67.4	68.0	68.5	68.6	68.7	68.7
TR Decoder + TR Encoder	TR Decoder	66.2	67.3	67.6	68.4	68.6	68.8	68.9	69.0
TR Decoder + TR Decoder	*N/A*	64.0	64.7	65.9	66.1	66.5	66.2	65.4	65.2
TR Decoder + TR Decoder	TR Decoder	**66.6**	**67.8**	**68.2**	**68.8**	**69.2**	**69.4**	**69.5**	**69.5**

We continue to explore the design trade-offs of LSTR. Unless noted otherwise, we use short-term memory of 8 seconds, long-term memory of 512 seconds, and Kinetics pretrained features.

Number of layers and tokens. First, we test to use different numbers of token embeddings (*i.e.*, n_0 and n_1) in LSTR encoder. Fig 4 (left) shows that LSTR is quite robust to different choices (the best and worst performance gap is only about 1.5%), but using $n_0 = 16$ and $n_1 = 32$ gets highest accuracy. Second, we experiment for the effect of using different number of Transformer decoder units (*i.e.*, ℓ_{enc} and ℓ_{dec}) in LSTR. As shown in Fig 4 (right), LSTR does not need a large model to get best performance, and in practice, using more layers can easily cause the overfitting problem.

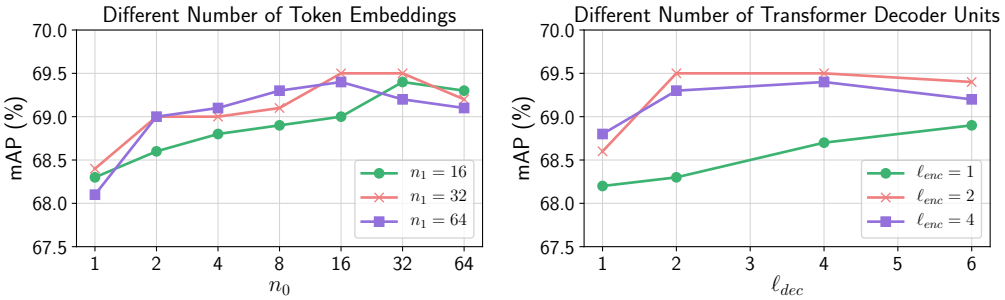

Figure 4: **Left**: Results of different number of token embeddings for our two-stage memory compression. **Right**: Results of different number of Transformer decoder units for ℓ_{enc} and ℓ_{dec}.

Can we unify the temporal modeling using only self-attention models? We test if long-term $\mathbf{M}_L$ and short-term $\mathbf{M}_S$ memory can be learned as a whole using self-attention models. Specifically, we concatenate $\mathbf{M}_L$ and $\mathbf{M}_S$ and feed them into a standard Transformer encoder [54] with similar model size to LSTR. Table 4 (row 7 vs. row 1) shows that LSTR achieves better performance especially when m_L is large (*e.g.* 69.5% vs. 66.5% when $m_L = 512$ and 66.6% vs. 65.7% when $m_L = 8$). This demonstrates the advantages of LSTR for temporal modeling on long- and short-term context.

Can we remove the LSTR encoder? We explore this by directly feeding $\mathbf{M}_L$ into LSTR decoder, and using $\mathbf{M}_S$ as tokens to reference useful information. Table 4 shows that LSTR outperforms this baseline (row 7 vs. row 2), especially when m_L is large. We also compare it with self-attention models (row 1) and observe that although neither of them can effectively model prolonged memory, this baseline outperforms Transformer Encoder. This also demonstrates the effectiveness of our idea of using short-term memory to query related context from long-range context.

Can the LSTR encoder learn effectively using only self-attention? To evaluate the "bottleneck" design in LSTR encoder, we implement a baseline that models $\mathbf{M}_L$ by using standard Transformer Encoder units [54]. Note that it still captures $\mathbf{M}_L$ and $\mathbf{M}_S$ with similar workflow of LSTR, but it does not compress and encode $\mathbf{M}_L$ using learnable tokens. The comparison (row 3 vs. row 7) in Table 4 shows that LSTR outperforms this baseline with all m_L settings. It shows that the performance of this baseline decreases when m_L is getting larger, which also suggests the supreme ability of LSTR for modeling long-range patterns.

How to design the memory compression for the LSTR encoder? First, we test the effectiveness of the single-stage design with $\ell_{enc}+1$ Transformer decoder units. Table 4 (row 4) shows that two-stage memory compression (row 7) is stably better than single-stage, and interestingly, the performance gap gets larger when using larger m_L (0.5% when $m_L = 8$ and 0.8% when $m_L = 512$). Second, we compare cross-attention with self-attention for two-stage compression by replacing the second Transformer decoder units with Transformer encoder. Table 4 shows that LSTR stably outperforms this baseline (row 5) by about 0.5% in mAP. However, its performance is better than models with one-stage compression about 1.3% (vs. row 3) and 0.3% (vs. row 4) on average.

Can we remove the LSTR decoder? We remove the LSTR decoder to evaluate its contribution. Specifically, we feed the entire memory to LSTR encoder and attach a non-linear classifier on its output tokens embeddings. Similar to above experiments, we increase the model size to ensure a fair comparison. Table 4 shows that LSTR outperforms this baseline (row 6) by about 4% on large m_L (*e.g.*, 512 and 1024) and about 2.5% on relative small m_L (*e.g.*, 8 and 16).

4.6 Error Analysis

Table 5: **Action classes with highest and lowest performance** on THUMOS'14 in AP (%).

Action Classes	HammerThrow	PoleVault	LongJump	Diving	BaseballPitch	FrisbeeCatch	Billiards	CricketShot
AP	92.8	89.7	86.9	86.7	55.4	49.4	39.8	38.6

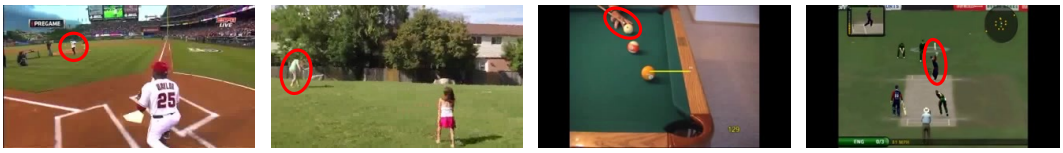

Figure 5: **Failure cases** on THUMOS'14. Action classes from left to right are "BaseballPitch", "FrisbeeCatch", "Billiards", and "CricketShot". Red circle indicates where the action is happening.

In Table 5, we list the action classes from THUMOS'14 [27] where LSTR gets the highest (green) and the lowest (red) frame APs. In Fig. 5 we illustrate sample frames with incorrect predictions. Please see more visualization in the Appendix. We observe that LSTR sees decrease in detection accuracy when the action incurs only tiny motion or the subject is very far away from the camera, but excels at detecting actions with long temporal span and multiple stages, such as "PoleVault" and "Long Jump". This suggests one direction of exploration to extend the temporal modeling capability of LSTR to both the spatial and the temporal domains.

5 Conclusion

We presented Long Short-term TRansformer (LSTR) for online action detection, which divides the historical observations into long- and short-term memories. LSTR compresses the long-term memory into encoded latent features, and then references related temporal context from them with short-term memory. Experimental results demonstrated that our memory design and jointly modeling the long- and short-term temporal information is beneficial for online action detection. Ablation studies showed LSTR's superior ability to deal with prolonged sequences with less heuristic algorithm designs.

References

[1] Anurag Arnab, Mostafa Dehghani, Georg Heigold, Chen Sun, Mario Lučić, and Cordelia Schmid. Vivit: A video vision transformer. *arXiv:2103.15691*, 2021.

[2] Gedas Bertasius, Heng Wang, and Lorenzo Torresani. Is space-time attention all you need for video understanding? *arXiv:2102.05095*, 2021.

[3] Shyamal Buch, Victor Escorcia, Chuanqi Shen, Bernard Ghanem, and Juan Carlos Niebles. SST: Single-stream temporal action proposals. In *CVPR*, 2017.

[4] Mikhail S Burtsev, Yuri Kuratov, Anton Peganov, and Grigory V Sapunov. Memory transformer. *arXiv:2006.11527*, 2020.

[5] Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. End-to-end object detection with transformers. In *ECCV*, 2020.

[6] Joao Carreira and Andrew Zisserman. Quo vadis, action recognition? A new model and the Kinetics dataset. In *CVPR*, 2017.

[7] Marvin M Chun. Visual working memory as visual attention sustained internally over time. *Neuropsychologia*, 2011.

[8] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv:1412.3555*, 2014.

[9] MMAction2 Contributors. Openmmlab's next generation video understanding toolbox and benchmark. `https://github.com/open-mmlab/mmaction2`, 2020.

[10] Nelson Cowan. *Attention and memory: An integrated framework*. Oxford University Press, 1998.

[11] Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc V Le, and Ruslan Salakhutdinov. Transformer-XL: Attentive language models beyond a fixed-length context. *ACL*, 2019.

[12] Roeland De Geest, Efstratios Gavves, Amir Ghodrati, Zhenyang Li, Cees Snoek, and Tinne Tuytelaars. Online action detection. In *ECCV*, 2016.

[13] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. *NAACL*, 2019.

[14] Jeffrey Donahue, Lisa Anne Hendricks, Sergio Guadarrama, Marcus Rohrbach, Subhashini Venugopalan, Kate Saenko, and Trevor Darrell. Long-term recurrent convolutional networks for visual recognition and description. In *CVPR*, 2015.

[15] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *ICLR*, 2021.

[16] Jeffrey L Elman. Finding structure in time. *Cognitive science*, 1990.

[17] Hyunjun Eun, Jinyoung Moon, Jongyoul Park, Chanho Jung, and Changick Kim. Learning to discriminate information for online action detection. In *CVPR*, 2020.

[18] Hyunjun Eun, Jinyoung Moon, Jongyoul Park, Chanho Jung, and Changick Kim. Temporal filtering networks for online action detection. *Pattern Recognition*, 2021.

[19] Christoph Feichtenhofer, Axel Pinz, and Richard P Wildes. Spatiotemporal multiplier networks for video action recognition. In *CVPR*, 2017.

[20] Jiyang Gao, Zhenheng Yang, and Ram Nevatia. RED: Reinforced encoder-decoder networks for action anticipation. In *BMVC*, 2017.

[21] Jiyang Gao, Zhenheng Yang, Chen Sun, Kan Chen, and Ram Nevatia. TURN TAP: Temporal unit regression network for temporal action proposals. *ICCV*, 2017.

[22] Mingfei Gao, Mingze Xu, Larry S Davis, Richard Socher, and Caiming Xiong. Startnet: Online detection of action start in untrimmed videos. *ICCV*, 2019.

[23] Mingfei Gao, Yingbo Zhou, Ran Xu, Richard Socher, and Caiming Xiong. WOAD: Weakly supervised online action detection in untrimmed videos. *CVPR*, 2021.

[24] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016.

[25] Minh Hoai and Fernando De la Torre. Max-margin early event detectors. In *IJCV*, 2014.

[26] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 1997.

[27] Haroon Idrees, Amir R Zamir, Yu-Gang Jiang, Alex Gorban, Ivan Laptev, Rahul Sukthankar, and Mubarak Shah. The THUMOS challenge on action recognition for videos "in the wild". *CVIU*, 2017.

[28] Andrew Jaegle, Felix Gimeno, Andrew Brock, Andrew Zisserman, Oriol Vinyals, and Joao Carreira. Perceiver: General perception with iterative attention. *arXiv:2103.03206*, 2021.

[29] Young Hwi Kim, Seonghyeon Nam, and Seon Joo Kim. Temporally smooth online action detection using cycle-consistent future anticipation. *Pattern Recognition*, 2021.

[30] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *ICLR*, 2015.

[31] Anastasia Kiyonaga and Tobias Egner. Working memory as internal attention: Toward an integrative account of internal and external selection processes. *Psychonomic bulletin & review*, 2013.

[32] Xinyu Li, Yanyi Zhang, Chunhui Liu, Bing Shuai, Yi Zhu, Biagio Brattoli, Hao Chen, Ivan Marsic, and Joseph Tighe. Vidtr: Video transformer without convolutions. *arXiv:2104.11746*, 2021.

[33] Tianwei Lin, Xiao Liu, Xin Li, Errui Ding, and Shilei Wen. BMN: Boundary-matching network for temporal action proposal generation. In *ICCV*, 2019.

[34] Tianwei Lin, Xu Zhao, Haisheng Su, Chongjing Wang, and Ming Yang. BSN: Boundary sensitive network for temporal action proposal generation. In *ECCV*, 2018.

[35] Yuan Liu, Lin Ma, Yifeng Zhang, Wei Liu, and Shih-Fu Chang. Multi-granularity generator for temporal action proposal. In *CVPR*, 2019.

[36] Shugao Ma, Leonid Sigal, and Stan Sclaroff. Learning activity progression in lstms for activity detection and early detection. In *CVPR*, 2016.

[37] Antoine Miech, Ivan Laptev, and Josef Sivic. Learnable pooling with context gating for video classification. *arXiv:1706.06905*, 2017.

[38] Megha Nawhal and Greg Mori. Activity graph transformer for temporal action localization. *arXiv:2101.08540*, 2021.

[39] Daniel Neimark, Omri Bar, Maya Zohar, and Dotan Asselmann. Video transformer network. *arXiv:2102.00719*, 2021.

[40] Klaus Oberauer. Working memory and attention–a conceptual analysis and review. *Journal of cognition*, 2019.

[41] Seoung Wug Oh, Joon-Young Lee, Ning Xu, and Seon Joo Kim. Video object segmentation using space-time memory networks. In *CVPR*, 2019.

[42] Sanqing Qu, Guang Chen, Dan Xu, Jinhu Dong, Fan Lu, and Alois Knoll. LAP-Net: Adaptive features sampling via learning action progression for online action detection. *arXiv:2011.07915*, 2020.

[43] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding by generative pre-training. 2018.

[44] Mike Schuster and Kuldip K Paliwal. Bidirectional recurrent neural networks. *ITSS*, 1997.

[45] Gilad Sharir, Asaf Noy, and Lihi Zelnik-Manor. An image is worth 16x16 words, what is a video worth? *arXiv:2103.13915*, 2021.

[46] Zheng Shou, Jonathan Chan, Alireza Zareian, Kazuyuki Miyazawa, and Shih-Fu Chang. CDC: Convolutional-de-convolutional networks for precise temporal action localization in untrimmed videos. In *CVPR*, 2017.

[47] Zheng Shou, Junting Pan, Jonathan Chan, Kazuyuki Miyazawa, Hassan Mansour, Anthony Vetro, Xavier Giro-i Nieto, and Shih-Fu Chang. Online action detection in untrimmed, streaming videos-modeling and evaluation. In *ECCV*, 2018.

[48] Zheng Shou, Dongang Wang, and Shih-Fu Chang. Temporal action localization in untrimmed videos via multi-stage cnns. In *CVPR*, 2016.

[49] Jing Tan, Jiaqi Tang, Limin Wang, and Gangshan Wu. Relaxed transformer decoders for direct action proposal generation. *arXiv:2102.01894*, 2021.

[50] Yongyi Tang, Xing Zhang, Lin Ma, Jingwen Wang, Shaoxiang Chen, and Yu-Gang Jiang. Non-local netvlad encoding for video classification. In *ECCVW*, 2018.

[51] Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Hervé Jégou. Training data-efficient image transformers & distillation through attention. *arXiv:2012.12877*, 2020.

[52] Du Tran, Lubomir Bourdev, Rob Fergus, Lorenzo Torresani, and Manohar Paluri. Learning spatiotemporal features with 3d convolutional networks. In *ICCV*, 2015.

[53] Du Tran, Heng Wang, Lorenzo Torresani, Jamie Ray, Yann LeCun, and Manohar Paluri. A closer look at spatiotemporal convolutions for action recognition. In *CVPR*, 2018.

[54] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *arXiv:1706.03762*, 2017.

[55] Limin Wang, Yuanjun Xiong, Zhe Wang, Yu Qiao, Dahua Lin, Xiaoou Tang, and Luc Van Gool. Temporal segment networks: Towards good practices for deep action recognition. In *ECCV*, 2016.

[56] Sinong Wang, Belinda Li, Madian Khabsa, Han Fang, and Hao Ma. Linformer: Self-attention with linear complexity. *arXiv:2006.04768*, 2020.

[57] Xiaolong Wang, Ross Girshick, Abhinav Gupta, and Kaiming He. Non-local neural networks. In *CVPR*, 2018.

[58] Chao-Yuan Wu, Christoph Feichtenhofer, Haoqi Fan, Kaiming He, Philipp Krahenbuhl, and Ross Girshick. Long-term feature banks for detailed video understanding. In *CVPR*, 2019.

[59] Chao-Yuan Wu and Philipp Krähenbühl. Towards long-form video understanding. In *CVPR*, 2021.

[60] Saining Xie, Chen Sun, Jonathan Huang, Zhuowen Tu, and Kevin Murphy. Rethinking spatiotemporal feature learning: Speed-accuracy trade-offs in video classification. In *ECCV*, 2018.

[61] Huazhe Xu, Yang Gao, Fisher Yu, and Trevor Darrell. End-to-end learning of driving models from large-scale video datasets. In *CVPR*, 2017.

[62] Huijuan Xu, Abir Das, and Kate Saenko. R-C3D: Region convolutional 3d network for temporal activity detection. In *ICCV*, 2017.

[63] Mingze Xu, Mingfei Gao, Yi-Ting Chen, Larry S Davis, and David J Crandall. Temporal recurrent networks for online action detection. In *ICCV*, 2019.

[64] Joe Yue-Hei Ng, Matthew Hausknecht, Sudheendra Vijayanarasimhan, Oriol Vinyals, Rajat Monga, and George Toderici. Beyond short snippets: Deep networks for video classification. In *CVPR*, 2015.

[65] Bowen Zhang, Hao Chen, Meng Wang, and Yuanjun Xiong. Online action detection in streaming videos with time buffers. *arXiv:2010.03016*, 2020.

[66] Hang Zhao, Antonio Torralba, Lorenzo Torresani, and Zhicheng Yan. Hacs: Human action clips and segments dataset for recognition and temporal localization. In *ICCV*, 2019.

[67] Peisen Zhao, Jiajie Wang, Lingxi Xie, Ya Zhang, Yanfeng Wang, and Qi Tian. Privileged knowledge distillation for online action detection. *arXiv:2011.09158*, 2020.

[68] Yue Zhao, Yuanjun Xiong, Limin Wang, Zhirong Wu, Xiaoou Tang, and Dahua Lin. Temporal action detection with structured segment networks. In *ICCV*, 2017.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] See Sec. 4.6.
   (c) Did you discuss any potential negative societal impacts of your work? [No] We experimented on publicly available datasets. The methods for online action detection can be used in real-world applications, such as patient monitoring systems, interactive home virtual assistants, and collaborative robots. But there still could be unintended uses which is out of our control.
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] All datasets we used are publicly available and we discussed their details in Sec. 4.1. Our code is proprietary, but we tried our best to provide sufficient implementation details to reproduce our model in Sec.4.2.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] The hyperparameters for training are discussed in Sec. 4.2 which are determined by cross-validation on the standard dataset.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] We ran experiments three times with random initialization from independent seeds and reported their averaged numbers in Table 1 and 2.
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] See Sec. 4.2. We also described how to preprocess raw videos in Sec. 4.2.
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes] We cited the datasets we used, THUMOS'14, TVSeries, and HACS Segment, which are all publicly available. We also used the opensource toolkit, MMAction2, for feature extraction and cited their work. See Sec. 4.2.

(b) Did you mention the license of the assets? [No] MMAction2 is released under the Apache License. THUMOS'14 and TVSeries are under the non-commercial use license. HACS Segment dataset is under the BSD 3-Clause License.

(c) Did you include any new assets either in the supplemental material or as a URL? [No]

(d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [No] This is included in the official descriptions of the datasets and code.

(e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [No] We do not have sufficient resource to exhaustively screen offensive content from the dataset we used. We expect, because the datasets include common human activities (*e.g.*, sports), they are unlikely to contain offensive content.

5. If you used crowdsourcing or conducted research with human subjects...

(a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]

(b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]

(c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]