ETAD: Training Action Detection End to End on a Laptop

Shuming Liu[1], Mengmeng Xu[1], Chen Zhao[1], Xu Zhao[2], Bernard Ghanem[1]
[1]King Abdullah University of Science and Technology [2]Shanghai Jiao Tong University
{shuming.liu, mengmeng.xu, chen.zhao, bernard.ghanem}@kaust.edu.sa zhaoxu@sjtu.edu.cn

Abstract

Temporal action detection (TAD) with end-to-end training often suffers from the pain of huge demand for computing resources due to long video duration. In this work, we propose an efficient temporal action detector (ETAD) that can train directly from video frames with extremely low GPU memory consumption. Our main idea is to minimize and balance the heavy computation among features and gradients in each training iteration. We propose to sequentially forward the snippet frame through the video encoder, and backward only a small necessary portion of gradients to update the encoder. To further alleviate the computational redundancy in training, we propose to dynamically sample only a small subset of proposals during training. Various sampling strategies and ratios are studied for both the encoder and detector. ETAD achieves state-of-the-art performance on TAD benchmarks with remarkable efficiency. On ActivityNet-1.3, training ETAD ***in 18 hours*** *can reach* ***38.25% average mAP*** *with only* ***1.3 GB memory*** *per video* ***under end-to-end training****. Code is available at https://github.com/sming256/ETAD.*

1. Introduction

Let us assume a junior researcher, who does not have access to a high-end GPU (*e.g.* NVIDIA A100), starts to work on the temporal action detection (TAD) task, which takes as input a raw video and predicts the period of pre-defined temporal activities [10, 14, 44, 49]. Although great progress has been made in this area, training the whole TAD pipeline is getting computationally heavier and slower, which may discourage the disadvantaged researcher when only limited resources are available. To help with this situation, an efficient end-to-end TAD method with a low-cost requirement (*e.g.* a standard laptop) is in demand.

Most of the current TAD pipeline consists of a video encoder and an action detector. Training them jointly, *i.e. end-to-end training*, has become the recent trend [6, 7, 21, 27]. The advantage of such a paradigm is multi-fold, *e.g.* it allows feature adaptation on the target data domain, enables online frame augmentation to enhance the representation, *etc.* The main challenge of end-to-end training for TAD is the tremendous GPU memory requirement to process a single long untrimmed video (*e.g.* 34 GB for 5 mins video). This is why TAD methods like ASFD [22] resort to downscaling the video frame resolution to 96×96 and sampling a small set of frames (768) during training, while SBP [7] stops a portion of the gradient flow for backpropagation, and TALLFormer [6] caches most of the video features and only updates $15\% - 60\%$ of them. Nonetheless, these methods still need moderate GPU memory (*e.g.* 32 GB) to achieve state-of-the-art detection performance.

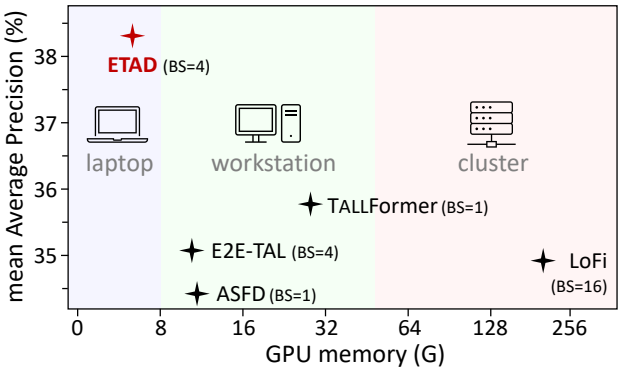

Figure 1. **Compared with recent end-to-end TAD methods, ETAD has very low GPU memory consumption and SOTA performance.** ETAD minimizes and balances the heavy computation among features and gradients. On ActivityNet-1.3, it reaches 38.25% average mAP, 2.65% higher than SOTA end-to-end method TALLFormer [6], while only using 5.2 GB GPU memory (batch size 4) and 18 hours of training.

Our main motivation is to reduce the computation redundancy and leverage minimal GPU memory during end-to-end TAD training. First, although current methods process multiple video snippets in parallel to extract features, our study shows that a sequential process of snippet encoding as well as backpropagation can significantly reduce the peak memory usage without sacrificing any detection performance, and it only moderately increases the train-

ing time. Meanwhile, we observe that not all snippets are needed for updating the video encoder during backpropagation, since most consecutive frames in an untrimmed video are similar in semantics. Second, to guarantee a high recall rate and cover all potential temporal activities, common TAD practice utilizes a dense distribution of action candidates or proposals, such as the proposal map in BMN [24]. We find that such a design choice is not necessary, since most proposals overlap with each other, and they eventually share similar feature representations. Our study shows that sampling only a small portion of these proposals does not affect the detection performance but can improve the training efficiency, and further reduce memory usage.

In this work, we propose an Efficient Temporal Action Detector (ETAD), which provides an end-to-end TAD solution that requires extremely low GPU memory and has affordable training time, as shown in Fig. 1. The success of ETAD is based on a **Sequencialized Gradient Sampling (SGS)** process and an **Action Proposal Sampling (APS)** design. SGS forwards the snippet frames in micro-batches through the video encoder, and selectively backwards only a small portion of gradients, reducing the peak GPU memory usage up to 92%. Additionally, SGS can reduce the delay in synchronizing the encoder input and the detector input, which can result in a training time that is similar to parallelized solutions (only +14%). APS, on the other hand, generates a much smaller but sufficient set of action candidates during training. It shows that only 6% of proposals can still guarantee decent action detection performance and thus remove the training redundancy. Subsequently, our empirical study on sampling strategies in both modules shows that most common strategies, such as label-guided sampling and feature-guided sampling, are not evidently better than the heuristic stochastic sampling.

ETAD achieves state-of-the-art performance on two popular benchmarks in an end-to-end fashion with low memory cost and acceptable time consumption. On ActivityNet-1.3, for example, we train ETAD on a single GPU for 18 hours to reach 38.25% average mAP, while the memory consumption is only 1.3 GB per video. Note that this memory usage is even less than many TAD methods that take as input pre-extracted video features [44, 48]. The main contributions of this work can be summarized as follows:

1. We propose to sequentially backpropagate a small portion of gradients to update the video encoder for end-to-end TAD training. This significantly reduces GPU memory usage without increasing training time much.
2. We adopt various sampling strategies to study the snippet gradient redundancy and action proposal redundancy in the current TAD framework. Surprisingly, using only 6% of proposals and 30% of snippet gradients can guarantee a good detection performance.
3. Extensive experiments show that ETAD reaches state-of-the-art performance on two TAD benchmarks, ActivityNet-1.3 and THUMOS-14. In particular, ETAD achieves 38.25% average mAP on ActivityNet with only 1.3 GB per video in end-to-end training.

2. Related Work

Temporal Action Detection. An action detector can localize action instances directly from videos (*direct*), or merely refine the boundaries of proposals from a proposal-generation network (*refinement*). The *direct* methods usually focus on enhancing the temporal feature representation [29, 44] or improving the proposal evaluation [2, 4, 13, 24, 40, 43]. For example, G-TAD [44] utilizes graph convolutions to model the correlations between video snippets. The *refinement* methods tend to prune off-the-shelf action proposals [10, 11, 28] and provide more accurate boundary predictions [33, 34, 48]. P-GCN [45] is a typical refinement method that exploits proposal-proposal relations to refine predictions of BSN [25]. TCANet [33] uses high-quality proposals generated from BMN [24] and proposes a cascade structure to progressively refine actions. Our proposed ETAD belongs to the family of direct solutions, since it does not rely on any external proposal generation methods, but it can surpass the best refinement method.

End-to-end Solutions in TAD. Recently, more methods study TAD directly from the original video frames to the final proposal predictions, which is referred to as *end-to-end training*. The early work R-C3D [40] encodes the frames with 3D filters and proposes action segments then classifies and refines them. PBRNet [26] and ASFD [21] also train detectors from raw frames, but they suffer from the small batch size and low-resolution frames. E2E-TAL [27] further confirms the benefit of end-to-end training for TAD and studies different design choices. Re^{2}TAL [47] propose to rewires the pretrained video backbones as reversible modules, whose intermediate activations can be cleared from memory during training. Moreover, some works propose ways of pre-training the video encoder by new training tasks to close the gap between action recognition and action detection, *e.g.* TSP [1] and BSP [41]. Differently, our ETAD is able to train the network with high frame resolution, large batch size, and single-stage training.

Sampling in Video Understanding. Although densely sampling snippets over the entire video is effective for understanding short video clips, such an approach is expensive for long untrimmed videos. An alternative way is to summarize the video [15] by selecting only the relevant frames or snippets. For example, SCSampler [18] selects salient clips from video for efficient action recognition. SBP [7] stochastically drops certain backpropagation paths to train the action recognition/detection model memory efficiently. However, the forward path of SBP still requires a

lot of memory for long video input. To reduce the forward computation, TALLFormer [6] first stores the pre-computed video feature in a feature bank and only updates a relatively small portion of features in each iteration. Since features in the bank are not always up-to-date, if the training dataset is too large, this method may fail because the features of the same video between two epochs can be drastically different. Besides, proposal sampling in TAD is an under-explored topic, and most methods [20, 24, 44] exhaustively enumerate the possible locations of activity, leading to redundant computation for highly overlapped proposals.

3. Method

Given an untrimmed video, temporal action detection aims to predict its foreground actions, denoted as $\Psi = \{\varphi_i = (t_s, t_e, c)\}_{i=1}^{M}$, where (t_s, t_e, c) are the start time, end time, and category of the action instance φ_i, respectively. M is the total number of actions.

3.1. Model Architecture

The overall architecture of ETAD is shown in Fig. 2, which illustrates the pipeline of feature extraction and action detection. For *feature extraction*, an off-the-shelf action recognition model, such as TSM [23], R(2+1)D [37], is adapted to encode multiple video snippets to a list of feature vectors. Specifically, each vector is obtained from the feature map before the classification head of the recognition model, with global average pooling applied on the temporal and spatial dimensions. For *action detection*, we adopt a simple yet effective detector to retrieve actions. First, two LSTM layers capture long-range temporal relations to enhance the snippet-level feature representations. Then, two convolution layers are applied to classify the startness and endness of each snippet. Last, a proposal evaluation module refines the candidate proposal boundaries and predicts the proposal confidence. To improve the boundary precision, we stack three proposal evaluation modules with progressively improved IoU thresholds. More details of the detection head can be found in the *supplementary material*.

Based on the simple detection head, we adopt sequentialized gradient sampling and action proposal sampling to alleviate the computation burden, targeting efficient end-to-end TAD training with *minimal* memory usage.

3.2. Preliminary of Sequentialized Gradient

Typically in TAD, the untrimmed long video is represented as $X \in \mathbb{R}^{N\times 3\times T\times H\times W}$, where N snippets (or clips) are sampled from the video, and each snippet x has T frames with spatial resolution $H \times W$. Given an arbitrary video encoder denoted as $\boldsymbol{f_e}$, which can be a CNN-based or a transformer-based action recognition model, snippet x is encoded as a feature vector $f \in \mathbb{R}^C$ by the video encoder. Thus, the feature sequence $F \in \mathbb{R}^{N\times C}$ is extracted from N snippets in parallel. Subsequently, an action detector $\boldsymbol{f_d}$ retrieves action candidates ϕ from this feature sequence. The whole forward process can be denoted as follows:

$$F = [f_1, f_2, ..., f_N] = \boldsymbol{f_e}([x_1, x_2, ..., x_N]) \tag{1}$$

$$\phi = \boldsymbol{f_d}(F) \tag{2}$$

During training, in order to update the parameters $\boldsymbol{\theta}$ of video encoder $\boldsymbol{f_e}$, all the intermediate activations in Eq. (1) must be saved for later gradient backpropagation. Given the loss L, the gradients of $\boldsymbol{\theta}$ are computed by:

$$\Delta\boldsymbol{\theta} = \frac{\partial L}{\partial \boldsymbol{\theta}} = \frac{\partial L}{\partial F} \cdot \frac{\partial F}{\partial \boldsymbol{\theta}} \tag{3}$$

Compared with standard action recognition, the existence of snippet dimension in X causes a tremendous computation in $\boldsymbol{f_e}$, which increases linearly with the number of snippets N. For example, when using 128 snippets, 224×224 resolution, 16 videos in a batch, even a light weight action recognition model (TSM [23]) needs more than 500GB memory, which is infeasible to perform end-to-end training on most platforms. **However**, the computation graph of each snippet in $\boldsymbol{f_e}$ is essentially independent of others. Precisely, as long as no batch-level parameters need to update, such as batch normalization, we can apply associative law to Eq. (1) to get $f_i = \boldsymbol{f_e}(x_i)$, then the parallel computation in Eq. (3) can be decoupled as:

$$\Delta\boldsymbol{\theta} = \frac{\partial L}{\partial [f_1, ..., f_N]} \frac{\partial [f_1, ..., f_N]}{\partial \boldsymbol{\theta}} = \sum_{i=1}^{N} \Delta f_i \frac{\partial f_i}{\partial \boldsymbol{\theta}} \tag{4}$$

3.3. Sequentialized Gradient Sampling

Eq. (4) suggests us that the computation of the partial derivative of snippet feature F to encoder parameter $\boldsymbol{\theta}$ can be obtained from two different ways, *i.e.* parallelly compute the derivative from the N snippets in one-step, or divide N snippets into multiple **micro-batches** (a micro-batch has K snippets, $K << N$), and then ***sequentially*** compute the gradient within a micro-batch by N/K iterations and accumulate the gradients. Based on the above derivation, we propose **Sequentialized Gradient Sampling (SGS)** for efficient end-to-end TAD training, which is consist of three stages as illustrated in Fig. 2.

1. *Sequentialized video encoding.* We temporally split the video X into multiple micro-batches. Each micro-batch has K snippets. We run forward passes on the encoder for N/K times without saving any intermediate activations, and concatenate the output features in the temporal dimension as F.

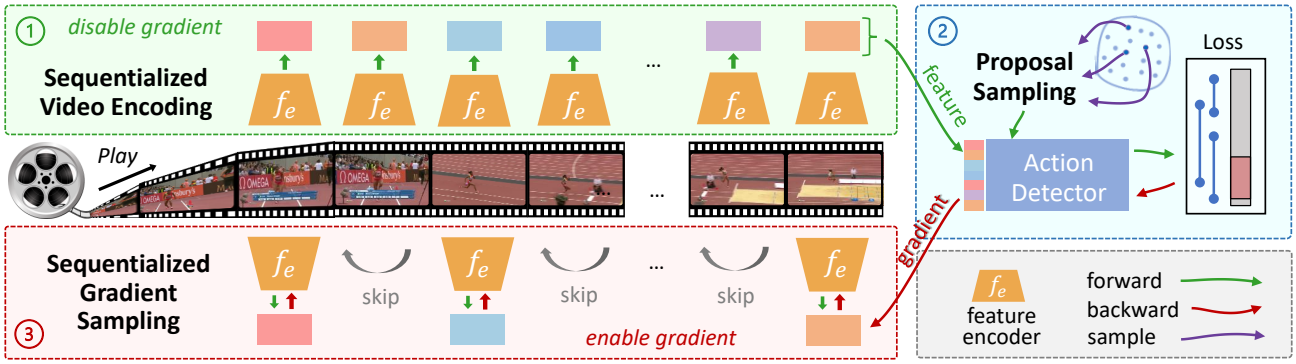

Figure 2. **The training pipeline of ETAD can be divided into three stages: sequentialized video encoding, action detector learning, and sequentialized gradient updating.** The sequential video decoding in stage 1 and sequential gradient updating in stage 3 visibly alleviate the whole network's GPU memory consumption. The action proposal sampling in stage 2 and gradient sampling in stage 3 further cut down the computation redundancy and reduce the training time.

2. *Action detector learning.* We use F to train the detector for one iteration and backpropagate the gradients to the concatenated features F. We collect the feature gradients $\Delta[f_1, \cdots, f_N]$, and free all the cache in GPU memory.
3. *Gradient sampling and sequentialized updating.* We sample γ portion from N snippets and sequentially update the encoder with their gradients. In each step, we use a micro-batch to compute the gradients of encoder parameters and accumulate all the gradients for the later parameter update.

The key to achieving efficient end-to-end training by SGS is to sequentially process a small micro-batch data in each iteration during stage 1 and stage 3, and only backward a small portion of gradients during stage 3. As a comparison, in traditional end-to-end training, all N snippets' intermediate activations in the video encoder in stage 1 are reserved for later backpropagation, which takes over 95% of total GPU memory. Instead, our SGS operates on a small data volume for video encoding during each iteration, and the peak memory usage is only $\frac{K}{N}$ of the traditional end-to-end setting. Since the micro-batch data is related to K instead of N, such memory usage can be constant and independent of the video length. In the extreme case, if $K = 1$, no matter how long the video is given, the maximum memory usage by SGS is aligned with the memory usage as designed in action recognition task. Another advantage offered by SGS is its high GPU utilization. Unlike traditional parallel design that requires all snippets to be ready for the video encoder, which can result in high latency such as frame decoding and data augmentation, SGS can efficiently utilize GPUs without such requirements.

Moreover, although extra forward computation is involved in stage 3, using gradient sampling in SGS can address this deficiency and reduce the overall computation to be less than the original end-to-end training (see Tab. 5). In the experiments, we find that such sampling won't affect the TAD performance (see Sect. 4.3). This is because the consecutive video frames in the untrimmed video are usually similar in appearance and semantics, and the feature vectors of corresponding snippets may share similar representations. Thus, the gradients of encoders on such snippets are also similar. Moreover, as mentioned by [6], since the video encoder is already pre-trained on a large-scale action recognition dataset, thus it evolves more slowly than other modules in the network with a smaller learning rate, leading to relatively small gradient values. Based on such insight, our SGS which only backpropagates a small ratio of snippets would still guarantee high TAD performance.

Regards to time efficiency, although the sequential processing breaks down the parallel design, the total training time using SGS is only 114% than original end-to-end training, but it requires less than 1/25 memory of the default setting (see Tab. 3). Besides, SGS can be complementary of other memory-efficient techniques, such as activation checkpointing [5], mixed-precision training [31], *etc.* Noted that our SGS is agnostic of encoder architecture and thus can incorporate any of the common encoders in its framework.

In summary, our proposed SGS approach can effectively achieve efficiency in **memory usage**, **overall computation**, and **training time** under end-to-end training. The pseudo-code of our SGS algorithm can be found in the *supplementary material*.

3.4. Action Proposal Sampling

Beyond SGS, we also study proposal sampling, which aims to reduce the redundant action proposals in the action detector. In the current two-stage TAD methods (*i.e.* methods use RoI alignment or similar to explicitly extract

proposal features), a dense candidate proposal set is needed for proposal refinement. For example, BMN [24] and G-TAD [44] enumerate all possible combinations of start and end locations to generate candidate proposals to deal with the large action length variation. Mathematically, given the number of snippets N, there will be $C_N^2 = N \cdot (N-1)/2$ proposals, which has the quadratic complexity with respective to N. However, due to the dense enumeration, most of these proposals overlap with each other. Thus, a large portion of the extracted proposal features is similar or duplicated. Moreover, the proposal evaluation module in TAD usually refines each proposal's start and end boundary [26], so it is unnecessary to consider proposals that are temporally close.

To reduce such redundancy while preserving high detection performance, we propose to replace the densely sampled proposals with a subset produced by an efficient sampling approach, called **Action Proposal Sampling (APS)**, as illustrated in Fig. 2. Our experiments suggest that with a proper sampling strategy (see Sect. 3.5), using only 6% proposals can provide a similar detection performance to the full setup, but it saves more than 90% of the detector's computation. By combining APS with SGS, our method can achieve extremely memory-efficient end-to-end training with state-of-the-art detection performance.

3.5. Sampling Strategy

We further study the possible sampling strategies in both SGS and APS. Three types of sampling strategies are proposed and compared: *heuristic sampling*, *feature-guided sampling*, and *label-guided sampling*.

Heuristic Sampling includes three strategies: random, grid, and block. They are similar to the samplings in MAE [12]. The *random* strategy simply samples snippets or proposals randomly following a uniform distribution. The *grid* strategy samples the snippets with a pre-defined temporal stride (along the temporal dimension) or grid (on a proposal map), as shown in Fig. 3. The *block* strategy samples consecutive snippets or a block of proposals in the proposal map. This strategy essentially evaluates the model in a trimmed clip of the video.

Feature-guided Sampling are based on the data distribution in the feature space. Farthest Point Sampling (FPS, [9]) selects the new snippet/proposal which has the farthest distance, where a distance is defined as the euclidean distance between two snippet features or proposal features. FPS can provide the most distinguished samples of the candidates since the selected samples are more variant in the embedding space. We also implement the Determinantal Point Process (DPP) to enforce diversity during training. We take the cosine similarity as the kernel function and update the determinant in every training epoch. When a sampling ratio is given, we can directly apply kDPP [19] because the target has a fixed size. Please refer to the *supplementary material* for more discussions.

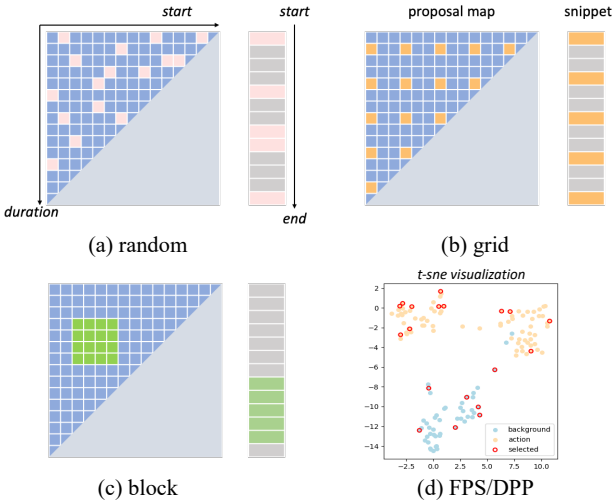

Figure 3. **Different sampling strategies: *random*, *grid*, *block*, *FPS*, *DPP*.** *Random* strategy samples certain proposals/snippets from a uniform distribution, *grid* strategy samples with a fixed temporal stride, and *block* strategy samples a consecutive area in proposal map/snippet sequence. Besides, *FPS* (farthest point sampling) chooses the new snippet/proposal which has the farthest distance to the selected samples. *DPP* (determinantal point process) selects the data from the feature embedding space to enforce the sample diversity.

Label-guided Sampling uses ground truth supervision during action proposal sampling. IoU-balanced sampling [32] guarantees the selected proposals have nearly the same number in different IoU thresholds, such as $\{0 \sim 0.3, 0.3 \sim 0.7, 0.7 \sim 1\}$. Similarly, scale-balanced sampling maintains the equivalence of proposal numbers around different action scales: small (scale<0.3), middle (0.3<scale<0.7), and large (scale>0.7).

Our experiment shows that random sampling, grid sampling, and DPP-based sampling all work well (see Tab. 4). Using a rather small sampling rate, *e.g.* 30% at SGS and 6% at the APS, can provide a decent TAD performance. Such small sampling ratios can greatly reduce computation in both the video encoder and the action detector while keeping the SOTA performance.

4. Experiments

4.1. Implementation details

Datasets and evaluation metrics. ActivityNet-1.3 [14] is a large-scale video understanding dataset, consisting of 19,994 videos annotated for the temporal action detection task. The dataset is divided into train, validation, and test sets with a ratio of 2:1:1. THUMOS-14 [16] contains 200

Table 1. **Action localization results on the validation set of ActivityNet-1.3**, measured by mAP (%) at different tIoU thresholds and the average mAP. E2E means the method is under end-to-end training. Mem. is the GPU memory usage (GB) *per video*.

Method	Video Encoder	E2E	Flow	0.5	0.75	0.95	Average	Mem.	Pub.
RTD-Action [36]	I3D	✗	✓	47.21	30.68	8.61	30.83	-	ICCV2021
P-GCN [45]	I3D	✗	✓	48.26	33.16	3.27	31.11	-	ICCV2019
BMN [24]	TSN	✗	✓	50.07	34.78	8.29	33.85	-	ICCV2019
VSGN [48]	TSN	✗	✓	52.38	36.01	8.37	35.07	1.6	ICCV2021
G-TAD [44]	R(2+1)D-34 (TSP)	✗	✗	51.26	37.12	9.29	35.81	0.7	CVPR2020
CSA [35]	R(2+1)D-34 (TSP)	✗	✗	52.64	37.75	7.94	36.25	-	ICCV2021
ActionFormer [46]	R(2+1)D-34 (TSP)	✗	✗	54.70	37.80	8.40	36.60	-	ECCV2022
RCL [39]	R(2+1)D-34 (TSP)	✗	✗	55.15	39.02	8.27	37.65	-	CVPR2022
R-C3D [40]	C3D	✓	✗	-	-	-	26.80	-	ICCV2017
AFSD [21]	I3D	✓	✓	52.40	35.30	6.50	34.40	12	CVPR2021
LoFi [42]	TSM-ResNet50	✓	✗	50.91	35.86	8.79	34.96	29	NeurIPS2021
PBRNet [26]	I3D	✓	✓	53.96	34.97	8.98	35.01	-	AAAI2020
E2E-TAL [27]	SlowFast-ResNet50	✓	✗	50.47	35.99	**10.83**	35.10	3	CVPR2022
TALLFormer [6]	Video Swin-B	✓	✗	54.10	36.20	7.90	35.60	29	ECCV2022
ETAD	TSM-ResNet50	✓	✗	53.79	37.59	10.56	36.79	1.7	CVPRW2023
ETAD	R(2+1)D-34 (TSP)	✓	✗	**55.49**	**39.32**	10.57	**38.25**	**1.3**	CVPRW2023

annotated untrimmed videos in the validation set and 213 videos in the test set. We also evaluate our methods on the HACS dataset [49] and achieve state-of-the-art performance (see *supplementary material*). Mean Average Precision (mAP) at certain IoU thresholds and average mAP are reported as the main evaluation metrics. On ActivityNet-1.3, the IoU thresholds are chosen from 0.5 to 0.95 with 10 steps. On THUMOS-14, the thresholds are chosen from $\{0.3, 0.4, 0.5, 0.6, 0.7\}$.

Implementation Details. Our method is implemented with PyTorch 1.12, CUDA 11.1, and mmaction2 [8] on 1 Tesla V100 GPU by default. TSM [23] and R(2+1)D [37] are adopted as our video encoder for end-to-end training on ActivityNet-1.3, while two stream I3D [3] is adopted as the encoder on THUMOS-14. We fix the weights of the first two stages of the video encoder and freeze all batch normalization layers. For TSM, the image resolution is set to 224×224 with clip length 8, which is the same as in [42]. For R(2+1)D, the image resolution is set to 112×112, and the clip length is set to 16, following [1]. We adopt random cropping as data augmentation. Note that the TSM model is only pretrained on Kinetics-400 [17] and not finetuned on the target datasets, *i.e.* ActivityNet-1.3, or THUMOS-14. The R(2+1)D model is pretrained on the ActivityNet dataset by [1]. We use a batch size of 4 and the AdamW optimizer [30] with weight decay of 10^{-4}. The learning rate is set to 10^{-3} for the action detector and $10^{-6}/10^{-7}$ for TSM/R(2+1)D. The micro-batch size K in SGS is set to 4 by default. The sampling ratios are 30% and 6% in SGS and APS, respectively. The total training epoch is set to 6 and the learning rate decays by 0.1 after 5 epochs. Following [24, 33], we apply the video-level classification scores from [50] on ActivityNet-1.3 and [38] on THUMOS-14.

4.2. Comparison with State-of-the-Art Methods

ActivityNet-1.3. Tab. 1 compares ETAD with other state-of-the-art methods on ActivityNet-1.3. Under end-to-end training, ETAD achieves 38.25% average mAP with only 1.3 GB memory (per video), outperforming other state-of-the-art end-to-end training methods both on efficiency and efficacy by a large margin. Compared with LoFi [42] which also uses TSM-ResNet50 as the video encoder, ETAD achieves +1.83 average mAP gain with only 15% GPU budget. Interestingly, the memory usage of end-to-end-based ETAD is even smaller than feature-based VSGN [48], suggesting that ETAD is extremely memory-efficient. When the batch size is 4, ETAD's total memory usage is still lower than 8 GB, which can be easily trained on a RTX2080.

THUMOS-14. We also show the advantage of our method on THUMOS-14 in Tab. 2, which reaches the comparable performance with other end-to-end methods, such as ASFD [21], E2E-TAL [27]. Particularly, ETAD is better on high IoU thresholds, indicating the high precision of the generated action boundaries. Furthermore, our SGS can enable end-to-end training of SOTA feature-based TAD methods, *e.g.* ActionFormer [46]. As shown in Tab. 2 (bottom block), end-to-end training consistently boosts the mAP under all IoU thresholds, while only costing 10.4 GB memory with the heavy Swin transformer backbone.

Table 2. **Action localization results on test set of THUMOS14**, measured by mAP (%) at different tIoU thresholds. † means the reproduced results with Video Swin-T and only RGB modality.

Method	0.3	0.4	0.5	0.6	0.7
BMN [24]	56.0	47.4	38.8	29.7	20.5
G-TAD [44]	57.3	51.3	43.0	32.6	22.8
TCANet [33]	60.6	53.2	44.6	36.8	26.7
VSGN [48]	66.7	60.4	52.4	41.0	30.4
AFSD [21]	67.3	62.4	55.5	43.7	31.1
E2E-TAL [27]	69.4	64.3	56.0	46.4	34.9
ETAD	**69.63**	**64.47**	**56.17**	**47.18**	**35.89**
ActionFormer†	69.63	62.63	51.26	38.29	21.10
$\cdots$+**ETAD**	**72.82**	**66.95**	**57.28**	**44.51**	**28.75**

4.3. Ablation Study

In this section, we conduct the ablation studies on ActivityNet-1.3 to verify the effectiveness of each design.

Up to 94% dense action proposals are redundant for action detection. To prepare an efficient and powerful action detector for end-to-end training, we first operate on pre-extracted video features to verify the effectiveness of APS. Fig. 4 shows that the performance of the detector saturates from a small proposal sampling ratio. When the sampling ratio is under 4%, the mAP starts to drop visibly. This result confirms our assumption that dense enumerated proposals are redundant for action detection. Using 6% sampling, ETAD successfully results in the same detection performance as using a complete proposal set. With pre-extracted frozen features, it speeds up the training 7.5x faster (from 45 mins to 6 mins), and cuts down 92% of memory usage (from 16 GB to 1.2 GB).

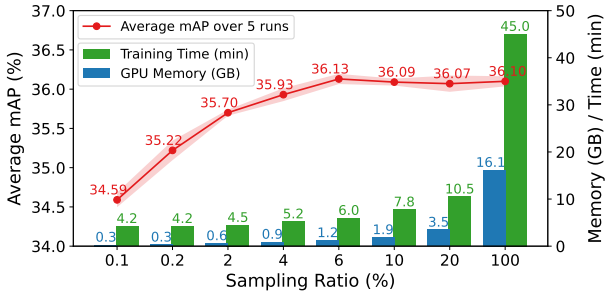

Figure 4. **Using only 6% proposals are sufficient for action detection.** For proposal sampling, we use pre-extracted TSM features with different sampling ratios and report the mAP, GPU memory, and training time. Random sampling is adopted.

End-to-end training can improve TAD performance, but it is memory-consuming. Based on the efficient action detector with APS, we extract the snippet feature with a learnable video encoder and jointly optimize it with the action detector. As shown in Tab. 3 (first row), end-to-end training can bring a significant performance gain from 36.13 to 36.85, which also proves the importance of end-to-end TAD training. However, this naive end-to-end training requires 137 GB memory, which is infeasible on most platforms.

Table 3. **Sequential backpropagation can greatly reduce the peak GPU memory while requiring more training time. Combined with gradient sampling, ETAD can achieve efficient and effective end-to-end training.** Seq. means adopting sequentialized backpropagation. Ratio is the gradient sampling ratio. K stands for the micro-batch size in SGS.

Seq.	Ratio	K	FLOPs	Mem.(GB)	Time	mAP
✗	100%	-	100%	137	100%	36.85
✓	100%	8	150%	10.3	180%	36.85
		4	150%	6.6	190%	
		2	150%	4.7	194%	
		1	150%	3.8	264%	
✓	50%	4	100%	6.6	137%	36.83
	40%	4	90%	6.6	121%	36.82
	30%	**4**	**80%**	**6.6**	**114%**	**36.79**
	20%	4	70%	6.6	101%	36.75
	10%	4	60%	6.6	91%	36.68

Sequential backpropagation is memory-efficient for end-to-end training, but it is also time-consuming. To further alleviate the memory limitations of end-to-end training, we apply sequential backpropagation on the naive end-to-end training, as shown in Tab. 3 (middle block). In experiments, we also find such an implementation has the same detection performance as the naive one, which also verifies the equality discussed in Sect. 3.3. Thus, we only compare the peak GPU memory and training time, which shows that adopting sequential backpropagation in end-to-end training can greatly reduce the GPU memory consumption from 103 GB to 3.8 GB. Unfortunately, the training time is also increased by 2.6× larger. Besides, since we need to recompute the activations during backpropagation, the number of FLOPs is also increased.

Gradient sampling can effectively save training time, without sacrificing detection performance. To further reduce the training time, gradient sampling is combined with sequential backpropagation, known as our complete SGS approach. As shown in Tab. 3 (bottom), gradient sampling with a ratio larger than 30% can still maintain nearly the same detection performance, which proves the existence of snippet-level learning redundancy. Such a scenario also happens in THUMOS-14 dataset (see *supplementary material*). In the meantime, the training time is evidently decreased from 190% to 114%, which is almost the same as the naive end-to-end training. These results verify that SGS can be served as an effective tool for memory-efficient end-to-end TAD training.

Table 4. **Effect of different sampling strategies.** We apply the TSM-R50 as the video encoder and report the mAP on ActivityNet. Frozen backbone is used in APS and end-to-end training is used in SGS, thus the later results are expected to be higher.

Sampling Type	Sampling Strategy	APS	SGS
heuristic	random	**36.13**	**36.79**
	grid	36.04	36.77
	block	32.97	36.74
feature-guided	FPS	33.59	36.61
	DPP	**36.16**	**36.78**
label-guided	IoU-balanced	34.84	N.A.
	Scale-balanced	35.10	N.A.

Heuristic sampling strategy is recommended. As shown in Tab. 4, from the APS column for proposal sampling, we find that random sampling, grid sampling, and DPP work well. While block sampling and label-guided sampling both show certain downgrades in performance because they change the proposal distribution and thus can not guarantee the variety of proposals. From the SGS column for gradient sampling, all experiments outperform the pre-extracted feature baseline in APS (36.13%). Considering the detection performance and computation complexity of different sampling strategies, we recommend adopting heuristic samplings such as random or grid strategies in APS and SGS. More discussions can be found in *supplementary material*.

4.4. Further Discussions

Compared with other end-to-end strategies, SGS shows both memory-efficiency and performance superiority. We compare SGS with other end-to-end TAD strategies in Tab. 5. From the aspect of memory usage, SGS leverages only 1.7 GB memory per video to train the model in an end-to-end fashion, which is much lower than other methods. From the aspect of detection performance, SGS reaches almost the same mAP as in naive end-to-end training, and beats other end-to-end strategies. For example, though TALLFormer [6] uses less forward computation by adopting the feature bank technique, the method may face the risk of failure if the training dataset is large, where the features of the same snippet between two epochs can be drastically different. Therefore, we insist to adopt the full forward propagation for all the snippets, and backward the gradient sequentially and selectively.

SGS is complementary to other memory-saving techniques. For example, activation checkpointing [5] also saves part of intermediate activations and does forward recomputation during backpropagation, but it operates on the model's different layers. Mixed-precision technique [31] adaptively combines half-precision computation to save memory and speed up the training. The gradient accumulation sums the gradient over multiple batches to implicitly change the batch size to save memory. For comparison, our sequential gradient sampling process focuses on reducing the complexity over **temporal dimensions**, instead of batch dimensions or depth dimensions. As shown in Tab. 6, SGS is complementary to aforementioned memory-saving techniques. It allows the detector to accommodate higher resolution frames, larger batch sizes, and a deeper backbone.

Table 5. **Comparison of Sequentialized Gradient Sampling with other end-to-end training strategies in TAD.** We set the sampling rate to 30% and use ETAD detector in all experiments. Computation in forward/backward stands for the theoretical computation cost of the video encoder during each propagation. GPU memory is reported with TSM-ResNet50 backbone.

Methods	Forward	Backward	Mem.	mAP
Pre-extracted Feature	0%	0%	1.2	36.13
Multi-stage Training [44]	60%	60%	44	36.36
Feature Bank [6]	30%	30%	44	36.54
SGS (ours)	**130%**	**30%**	**6.6**	**36.79**
Naive End-to-End	100%	100%	137	36.85

Table 6. **Combination of SGS with other memory-saving methods.** We report the memory usgae with batch size 4.

Method	mAP	Mem.(GB)
Naive End-to-End	36.85	137
SGS	36.79	6.6
Checkpoint	36.50	72
Checkpoint + SGS	36.63	4.8
Mxied Precision	36.49	67
Mxied Precision + SGS	36.70	4.8
Checkpoint + Mxied Precision + SGS	36.71	4.0

5. Conclusion

In this paper, we propose an end-to-end training method for the temporal action detector with extremely low GPU memory consumption. The training pipeline of ETAD contains sequentialized video encoding, action detector learning, and sequentialized gradient updating. ETAD achieves state-of-the-art action detection performance on multiple benchmarks. The proposed sequentialized gradient sampling method makes end-to-end training tractable in real-world applications, and the empirical results of different sampling strategies can shed light on how to effectively reduce computations in video localization problems. We hope this work will encourage the community to carry out more research on end-to-end training in various untrimmed video understanding tasks, such as video language grounding and video captioning.

References

[1] Humam Alwassel, Silvio Giancola, and Bernard Ghanem. TSP: Temporally-sensitive pretraining of video encoders for localization tasks. In *Int. Conf. Comput. Vis. Worksh.*, 2021. 2, 6

[2] Yueran Bai, Yingying Wang, Yunhai Tong, Yang Yang, Qiyue Liu, and Junhui Liu. Boundary content graph neural network for temporal action proposal generation. In *ECCV*, 2020. 2

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

[4] Yu-Wei Chao, Sudheendra Vijayanarasimhan, Bryan Seybold, David A. Ross, Jia Deng, and Rahul Sukthankar. Rethinking the faster R-CNN architecture for temporal action localization. In *CVPR*, 2018. 2

[5] Tianqi Chen, Bing Xu, Chiyuan Zhang, and Carlos Guestrin. Training deep nets with sublinear memory cost. *arXiv preprint arXiv:1604.06174*, 2016. 4, 8

[6] Feng Cheng and Gedas Bertasius. Tallformer: Temporal action localization with long-memory transformer. In *ECCV*, 2022. 1, 3, 4, 6, 8

[7] Feng Cheng, Mingze Xu, Yuanjun Xiong, Hao Chen, Xinyu Li, Wei Li, and Wei Xia. Stochastic backpropagation: A memory efficient strategy for training video models. *arXiv preprint arXiv:2203.16755*, 2022. 1, 2

[8] MMAction2 Contributors. Openmmlab's next generation video understanding toolbox and benchmark. github.com/open-mmlab/mmaction2, 2020. 6

[9] Yuval Eldar, Michael Lindenbaum, Moshe Porat, and Yehoshua Y Zeevi. The farthest point strategy for progressive image sampling. *IEEE Transactions on Image Processing*, 6(9):1305–1315, 1997. 5

[10] Victor Escorcia, Fabian Caba Heilbron, Juan Carlos Niebles, and Bernard Ghanem. DAPs: Deep action proposals for action understanding. In *ECCV*, 2016. 1, 2

[11] Jiyang Gao, Kan Chen, and Ramakant Nevatia. CTAP: Complementary temporal action proposal generation. In *ECCV*, 2018. 2

[12] Kaiming He, Xinlei Chen, Saining Xie, Yanghao Li, Piotr Dollár, and Ross Girshick. Masked autoencoders are scalable vision learners. In *CVPR*, 2022. 5

[13] Fabian Caba Heilbron, Wayner Barrios, Victor Escorcia, and Bernard Ghanem. SCC: Semantic context cascade for efficient action detection. In *CVPR*, 2017. 2

[14] Fabian Caba Heilbron, Victor Escorcia, Bernard Ghanem, and Juan Carlos Niebles. ActivityNet: A large-scale video benchmark for human activity understanding. In *CVPR*, 2015. 1, 5

[15] Cheng Huang and Hongmei Wang. A novel key-frames selection framework for comprehensive video summarization. *IEEE Transactions on Circuits and Systems for Video Technology*, 30(2):577–589, 2019. 2

[16] YG Jiang, J Liu, A Roshan Zamir, G Toderici, I Laptev, M Shah, and R Sukthankar. Thumos challenge: Action recognition with a large number of classes, 2014. 5

[17] Will Kay, Joao Carreira, Karen Simonyan, Brian Zhang, Chloe Hillier, Sudheendra Vijayanarasimhan, Fabio Viola, Tim Green, Trevor Back, Paul Natsev, et al. The kinetics human action video dataset. *arXiv preprint arXiv:1705.06950*, 2017. 6

[18] Bruno Korbar, Du Tran, and Lorenzo Torresani. Scsampler: Sampling salient clips from video for efficient action recognition. In *ICCV*, 2019. 2

[19] Alex Kulesza and Ben Taskar. k-dpps: Fixed-size determinantal point processes. In *ICML*, 2011. 5

[20] Chuming Lin, Jian Li, Yabiao Wang, Ying Tai, Donghao Luo, Zhipeng Cui, Chengjie Wang, Jilin Li, Feiyue Huang, and Rongrong Ji. Fast learning of temporal action proposal via dense boundary generator. In *AAAI*, 2020. 3

[21] Chuming Lin, Chengming Xu, Donghao Luo, Yabiao Wang, Ying Tai, Chengjie Wang, Jilin Li, Feiyue Huang, and Yanwei Fu. Learning salient boundary feature for anchor-free temporal action localization. In *CVPR*, 2021. 1, 2, 6, 7

[22] Chuming Lin, Chengming Xu, Donghao Luo, Yabiao Wang, Ying Tai, Chengjie Wang, Jilin Li, Feiyue Huang, and Yanwei Fu. Learning salient boundary feature for anchor-free temporal action localization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3320–3329, June 2021. 1

[23] Ji Lin, Chuang Gan, and Song Han. TSM: Temporal shift module for efficient video understanding. In *ICCV*, 2019. 3, 6

[24] Tianwei Lin, Xiao Liu, Xin Li, Errui Ding, and Shilei Wen. BMN: boundary-matching network for temporal action proposal generation. In *ICCV*, 2019. 2, 3, 5, 6, 7

[25] Tianwei Lin, Xu Zhao, Haisheng Su, Chongjing Wang, and Ming Yang. BSN: Boundary sensitive network for temporal action proposal generation. In *ECCV*, 2018. 2

[26] Qinying Liu and Zilei Wang. Progressive boundary refinement network for temporal action detection. In *AAAI*, 2020. 2, 5, 6

[27] Xiaolong Liu, Song Bai, and Xiang Bai. An empirical study of end-to-end temporal action detection. In *CVPR*, 2022. 1, 2, 6, 7

[28] Yuan Liu, Lin Ma, Yifeng Zhang, Wei Liu, and Shih-Fu Chang. Multi-granularity generator for temporal action proposal. In *CVPR*, 2019. 2

[29] Fuchen Long, Ting Yao, Zhaofan Qiu, Xinmei Tian, Jiebo Luo, and Tao Mei. Gaussian temporal awareness networks for action localization. In *CVPR*, 2019. 2

[30] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *ICLR*, 2019. 6

[31] Paulius Micikevicius, Sharan Narang, Jonah Alben, Gregory Diamos, Erich Elsen, David Garcia, Boris Ginsburg, Michael Houston, Oleksii Kuchaiev, Ganesh Venkatesh, et al. Mixed precision training. *arXiv preprint arXiv:1710.03740*, 2017. 4, 8

[32] Jiangmiao Pang, Kai Chen, Jianping Shi, Huajun Feng, Wanli Ouyang, and Dahua Lin. Libra r-cnn: Towards balanced learning for object detection. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 821–830, 2019. 5

[33] Zhiwu Qing, Haisheng Su, Weihao Gan, Dongliang Wang, Wei Wu, Xiang Wang, Yu Qiao, Junjie Yan, Changxin Gao, and Nong Sang. Temporal context aggregation network for temporal action proposal refinement. In *CVPR*, 2021. 2, 6, 7

[34] Zheng Shou, Dongang Wang, and Shih-Fu Chang. Temporal action localization in untrimmed videos via multi-stage CNNs. In *CVPR*, 2016. 2

[35] Deepak Sridhar, Niamul Quader, Srikanth Muralidharan, Yaoxin Li, Peng Dai, and Juwei Lu. Class semantics-based attention for action detection. In *CVPR*, 2021. 6

[36] Jing Tan, Jiaqi Tang, Limin Wang, and Gangshan Wu. Relaxed transformer decoders for direct action proposal generation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 13526–13535, 2021. 6

[37] 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. 3, 6

[38] Limin Wang, Yuanjun Xiong, Dahua Lin, and Luc Van Gool. Untrimmednets for weakly supervised action recognition and detection. In *CVPR*, 2017. 6

[39] Qiang Wang, Yanhao Zhang, Yun Zheng, and Pan Pan. Rcl: Recurrent continuous localization for temporal action detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 13566–13575, 2022. 6

[40] Huijuan Xu, Abir Das, and Kate Saenko. R-C3D: Region convolutional 3D network for temporal activity detection. In *ICCV*, 2017. 2, 6

[41] Mengmeng Xu, Juan-Manuel Pérez-Rúa, Victor Escorcia, Brais Martinez, Xiatian Zhu, Bernard Ghanem, and Tao Xiang. Boundary-sensitive pre-training for temporal localization in videos. In *ICCV*, 2021. 2

[42] Mengmeng Xu, Juan Manuel Perez Rua, Xiatian Zhu, Bernard Ghanem, and Brais Martinez. Low-fidelity video encoder optimization for temporal action localization. In *NeurIPS*, 2021. 6

[43] Mengmeng Xu, Mattia Soldan, Jialin Gao, Shuming Liu, Juan-Manuel Pérez-Rúa, and Bernard Ghanem. Boundary-denoising for video activity localization. *arXiv preprint arXiv:2304.02934*, 2023. 2

[44] Mengmeng Xu, Chen Zhao, David S Rojas, Ali Thabet, and Bernard Ghanem. G-TAD: Sub-graph localization for temporal action detection. In *CVPR*, 2020. 1, 2, 3, 5, 6, 7, 8

[45] Runhao Zeng, Wenbing Huang, Mingkui Tan, Yu Rong, Peilin Zhao, Junzhou Huang, and Chuang Gan. Graph convolutional networks for temporal action localization. In *ICCV*, 2019. 2, 6

[46] Chenlin Zhang, Jianxin Wu, and Yin Li. Actionformer: Localizing moments of actions with transformers. In *ECCV*, 2022. 6

[47] Chen Zhao, Shuming Liu, Karttikeya Mangalam, and Bernard Ghanem. Reˆ 2tal: Rewiring pretrained video backbones for reversible temporal action localization. In *CVPR*, 2023. 2

[48] Chen Zhao, Ali K Thabet, and Bernard Ghanem. Video self-stitching graph network for temporal action localization. In *ICCV*, 2021. 2, 6, 7

[49] Hang Zhao, Zhicheng Yan, Lorenzo Torresani, and Antonio Torralba. HACS: Human action clips and segments dataset for recognition and temporal localization. *ICCV*, 2019. 1, 6

[50] Y Zhao, B Zhang, Z Wu, S Yang, L Zhou, S Yan, L Wang, Y Xiong, D Lin, Y Qiao, et al. Cuhk & ethz & siat submission to activitynet challenge 2017. *CVPR ActivityNet Workshop*, 2017. 6