Stochastic Backpropagation:
A Memory Efficient Strategy for Training Video Models

Feng Cheng[1*] Mingze Xu[2†] Yuanjun Xiong[2] Hao Chen[2] Xinyu Li[2] Wei Li[2] Wei Xia[2]
[1]UNC Chapel Hill [2]AWS AI Labs
fengchan@cs.unc.edu, {xumingze,yuanjx,hxen,xxnl,wayl,wxia}@amazon.com

Abstract

We propose a memory efficient method, named Stochastic Backpropagation (SBP), for training deep neural networks on videos. It is based on the finding that gradients from incomplete execution for backpropagation can still effectively train the models with minimal accuracy loss, which attributes to the high redundancy of video. SBP keeps all forward paths but randomly and independently removes the backward paths for each network layer in each training step. It reduces the GPU memory cost by eliminating the need to cache activation values corresponding to the dropped backward paths, whose amount can be controlled by an adjustable keep-ratio. Experiments show that SBP can be applied to a wide range of models for video tasks, leading to up to 80.0% GPU memory saving and 10% training speedup with less than 1% accuracy drop on action recognition and temporal action detection.

1. Introduction

One of the common challenge in training video understanding models is the limited availability of GPU memory. Although video models, such as those for action recognition [3, 42, 45, 49] and temporal action detection [6, 13], are known to benefit from capturing context features over the entire span of a video [49, 50], it is not always feasible to end-to-end train them while taking as many input frames (*e.g.*, typically hundreds) as they need. For example, when training a temporal action detector using ResNet-50 [21] as feature extractor, 128 frames as inputs, and batch size 4, the feature extractor itself costs *over* 40 GB memory that could exceed the limit of most modern GPUs.

On the other hand, video data is highly redundant, which suggests opportunity for optimization. We observe that in video models, the activation maps and gradients of different frames are similar at the bottom layers but become more and

*The work was done during an Amazon internship.
†Corresponding Author.

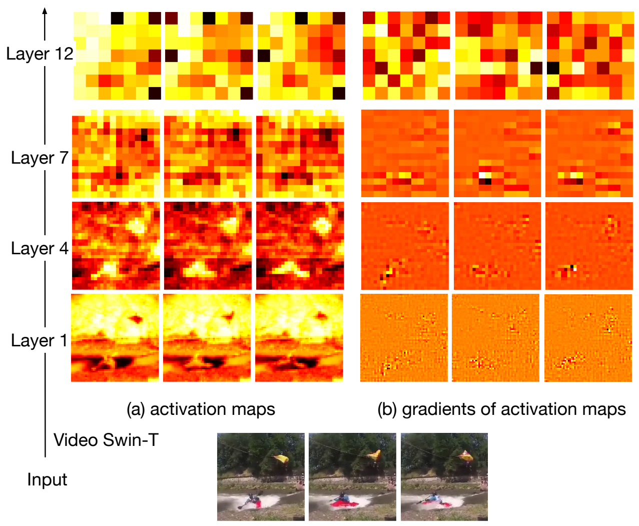


Figure 1. The activations and gradients are obtained from different layers (*e.g.*, layer 1, 4, 7, and 12) of Video Swin-T [33]. The activations and gradients between frames are similar at the bottom layers and are quite different at the top layers, which matches our assumption that there is much redundancy between the frames at the bottom layers but less at the top layers.

more different at the top layers. An example of Video Swin-T [33] on Kinetics-400 [7] dataset is shown in Fig. 1. During model training, the tiny difference in the seemly similar features (*i.e.*, fine-grained information) of the bottom layers is crucial for producing the difference in the top layers, however, they may not make much difference in terms of updating the parameters of these bottom layers. Thus, we hypothesize that complete computation is necessary for the activation maps (forward paths) to extract important semantic information but could be *unnecessary* for the gradients (backward paths).

In this paper, we propose a memory saving technique, named *Stochastic Backpropagation (SBP)*. Different from previous work that jointly removes the forward and backward paths [1], SBP randomly drops a proportion of the in-

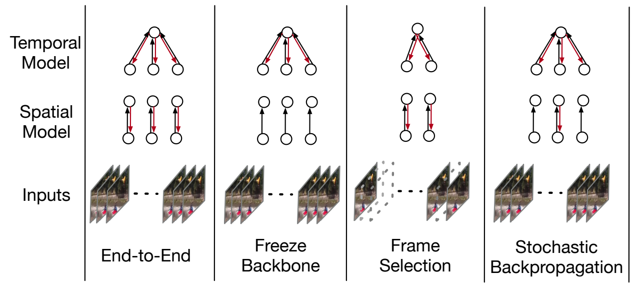


Figure 2. Comparison of different memory saving techniques. Different from freezing backbone or frame selection, Stochastic Backpropagation randomly removes the backward paths when training the spatial model while keep them for the temporal model.

dividual backward paths in backpropagation while keeping all the forward paths. Furthermore, we found that this random removal of backward paths can happen in a layer-wise manner, where we remove immediately connected paths of certain randomly sampled neurons in each layer. Note that, due to the chained dependency in the backpropagation algorithm, this would lead to incompletely computed gradients for all network parameters. However, we empirically found that gradients from this incomplete computation could still be sufficient for effectively updating the network parameters, as long as the overall computation graph is preserved. This makes SBP easy to be implemented without the need to account for the change of dependency across multiple layers. The memory saving of SBP is from the avoidance of caching the corresponding activation maps on removed computation paths. Since the memory can be safely re-used in backpropagation, the backward process uses almost no (one layer at most) additional memory[1], which makes the activation caching the major source of memory cost during training. Thus, the saving amount solely depends on the keep-ratio and can be significant when the keep-ratio is low.

SBP is a general technique that can be applied to a wide range of video tasks and models. Experimental results show that training with SBP uses only $0.2\times\sim0.5\times$ the GPU memory and speedup $1.1\times\sim1.2\times$ the training with less than 1% loss of accuracy on Kinetics-400 [7] and Epic-Kitchen-55 [12] for action recognition and less than 1% loss of mAP on THUMOS'14 [22] for temporal action detection.

2. Related Work

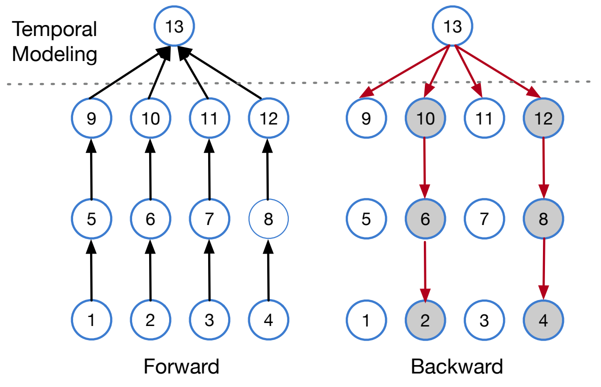
Video Understanding. Action recognition [7,42,45,46,49] aims to classify the activities of one or more agents in videos of various types and applications, from privacy-sensitive [53] and surveillance cameras [43] to egocentric videos [12, 28, 34]. Given an untrimmed video, temporal action localization estimates the start and end times of each action instance. One common practice [4,8,10,17,30,31,55] is to first generate action proposals, then identify their action classes and temporal boundaries. On the other hand, the "bottom-up" fashion [40, 57] first makes frame-level dense predictions and groups them as action instances. Online action detection [13,16,18,19,39,41,52,54,56], however, recognizes actions as soon as each video frame arrives without accessing any future information. Due to the high cost of GPU memory, most above offline and online methods cannot be trained in end-to-end manner and thus are built upon the extracted features from raw videos. Transformers recently achieve convincing results in video tasks, such as action recognition [2,3,27,33] and detection [37,44,54]. However, the high computational cost and memory demands also limit most of them to be only trained on short video clips. A few methods [5, 11] focus on designing Transformers to model long-form inputs, but mainly for text.

Memory Saving Techniques. The trade-off between memory and accuracy has been a standing topic in deep learning research. Gradient checkpoint [9] and accumulation, can save a large amount memory ($\sim$50%) but is likely to slow down the training process. Sparse Network [14] is applied to image recognition models but can only save the memory theoretically. Sideways [35, 36] reduce the memory cost by overwriting activations whenever new ones become available but can only be applied to causal models. Regarding video specific methods, a popular paradigm for training temporal action detectors is to build the model upon pre-extracted features for temporal modeling and reasoning ("Freeze Backbone" in Fig. 2). Two recent methods, AFSD [29] and DaoTAD [48], are end-to-end trained (DaoTAD also freezes the first two stages of their backbone), but they need to reduce the spatial resolution (*i.e.* 96×96 in AFSD and 112×112 in DaoTAD). However, freezing the backbone or reducing the input size can decrease the accuracy and is not an optimal solution. Since video usually contains high redundancy between adjacent frames, several work also tries to reduce the number of input frames, including frame dropout [1], early pooling [32] and frame selections [20] ("Frame Selection" in Fig. 2). However, aggressively subsampling of input frames will unavoidably lose important fine-scale information in the temporal domain, thus have a negative impact on the final accuracy [50, 54]. In addition, these methods cannot be applied to tasks that require per-frame predictions (*e.g.*, online action detection) or involve high-motion (*e.g.*, lip-reading). The comparisons between SBP and other memory saving techniques are shown in Fig. 2

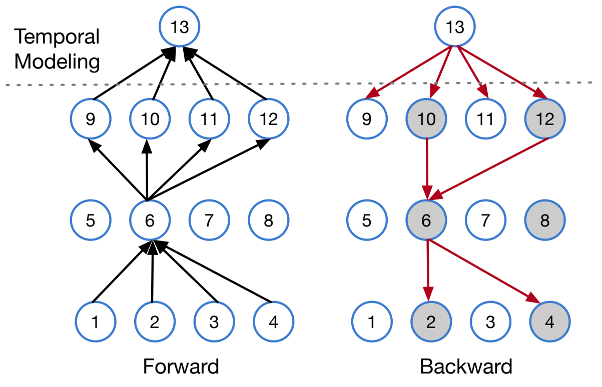
3. Stochastic Backpropagation

We propose a memory efficient strategy, *Stochastic Backpropagation (SBP)*, for training video models. Due to

[1]This mechanism is already encapsulated in most modern deep learning frameworks, such as PyTorch and Tensorflow.



(a) SBP on a tree model.



(b) SBP on a graph model.

Figure 3. Illustration of SBP on tree and graph models. In tree model (a), each node is only connected to the nodes with the same location in adjacent layers. In graph model (b), each node is potentially connected to all the nodes in adjacent layers (only connections of Node-6 are plotted for better visualization). During backward, the gradients will only propagate between the sampled nodes (in color gray).

the high-redundancy at the bottom layers, SBP randomly drops a proportion of their gradients during backpropagation. This can save the memory since we no longer need to keep track of some of the activations that are used to calculate the gradients. In this section, we first introduce the general idea of how SBP works in tree and graph models that are widely used in video tasks, then explain the concrete implementations for these two types of models.

3.1. Overview

When a model is defined, its backward computational graph is a Directed Acyclic Graph (DAG), $G(V, E)$, in which the vertices V are the activations cached for gradients calculation and edges E are the operations to calculate the gradients. As the model is a stack of layers, we also organize V in a layer-wise fashion, that is, $V = \{V^{l_0}, V^{l_1}, V^{l_2}, \cdots, V^{l_n}\}$, in which V^{l_i} is the output of layer l_i and V^{l_0} is the model input. We view V^{l_i} as a set of feature vectors. For example, if $V^{l_i} \in \mathbb{R}^{C\times D\times H\times W}$, we view V^{l_i} as $D \times H \times W$ vertices, where each vertex (we call it as *node* in this paper) is a feature vector $\in \mathbb{R}^C$.

As shown in Fig. 3, we generally divide the entire model into two parts: 1) spatial model (below dashed line) and 2) temporal model (above dashed line). The top several layers of the model can be viewed as the temporal model as these layers mainly learn global semantic information from all the frames. All the other bottom layers are viewed as the spatial model that mainly learns the spatial and local context information. SBP is only applied to the spatial model.

The tree-model is a special type of models with tree-like DAGs, whose children node is usually only connected with one father node. It is one of the simplest model structures but is widely adopted in long-term video tasks. To better illustrate SBP, we first introduce SBP on tree-models and then extend it to more general graph-models.

3.1.1 SBP on Tree-Models

In the tree-models, each node is only connected to the nodes with the same location at the neighboring layers (as shown in Fig. 3a). In long-term video tasks, most methods [52, 54, 55] use a spatial backbone, such as ResNet [21], to extract frame features. Such models can be viewed as tree-models as the frames are independently processed with each other.

Fig. 3a shows how SBP is done on tree-models. The several top layers are used to do temporal modeling, capturing temporal information from multiple frames. As there are more redundancy at the bottom layers than the top layers, we only drop the gradients at the bottom layers and keep all the gradients at the top layers. During normal end-to-end training, most memory is consumed at the bottom layers due to the much higher feature dimensions. SBP drops gradients at the bottom layers and eliminates the necessity of tracking the corresponding intermediate nodes, which can save lots of memory.

3.1.2 SBP on Graph-Models

In a more general graph-model, each node is potentially connected to all the nodes at the neighboring layers (as shown in Fig. 3b). Vision Transformers [3, 15, 33] are one type of fully-connected graph-models. Fig. 3b shows how SBP is done on graph-models. During forward, each node will aggregate data from all the nodes in the previous layer (only the connections of Node-6 is plotted for better visualization). During backward, similar to tree-models, the gradients are only propagated between the sampled nodes.

In tree-models, the gradients of sampled nodes are exactly the same as training using all the nodes. For example, in Fig. 3a the gradients we calculated for Node-6 are the same no matter Node-9,11,5,7 are sampled or not. But in general graph-models, as all the top nodes will propagate gradients to the bottom nodes, the gradients' calculation for

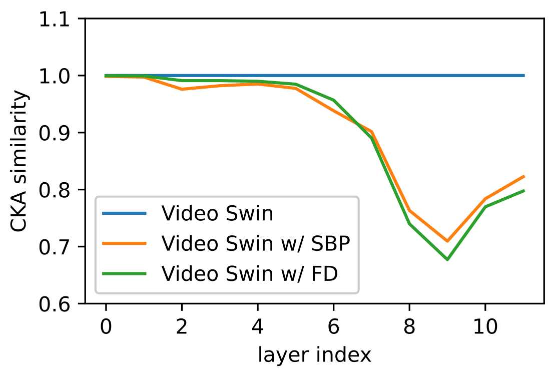


Figure 4. The CKA similarity [26] between SBP, Frame Dropout (FD) [1], and End-to-End training of the activations at the same layer on Video Swin-T [33]. The similarities at bottom layers are quite high, which indicates drop gradients at bottom layers will not hurt the model due to the high-redundancy. w/ SBP has higher similarity than w/ FD at the top layers, which shows the importance of keeping all gradients of top layers.

the sampled nodes are not exact and will be more and more inaccurate from top to bottom. However, this will not be a problem as there are more and more redundancy from top to bottom layers. Keeping the sampled nodes the same across layers will also alleviate this problem. We set the a uniform keep-ratio of SBP for all layers to simplify parameter identification. But we do note that a systematic strategy can be developed for setting layer-specific keep-ratios.

Why can we effectively learn with partial gradients? We believe it is because we remove the redundancy between frames but keep all important information. As shown in Fig. 1, as both the activation maps and the gradients of these activation maps are very similar between the frames at the bottom layers, dropping the gradients of some frames will not hurt the model too much. Fig. 4 also supports this point. The CKA similarity [26] $\in [0, 1]$ are used to measure the similarity of the representations between two neural networks. Frame Dropout (FD) [1] is trained using only $1/4$ of the frames as in the baseline, which is equivalent to dropping $3/4$ of the gradients in all the layers. The curve of FD is the CKA similarity between the model trained using 1/4 of the frames and using all the frames (baseline). The similarity is very high (>0.9) for the first 6 layers, which means dropping the gradients will not degrade model's performance dramatically.

Comparison with frame selection methods. The frame selection methods [20, 25] use a light-weight network to sample the frames from the video and then feed the sampled frames to the target model. Suppose we use the same frame sampler, the differences between SBP and frame selection methods are whether to preserve forward-path for the unsampled frames. We argue that keeping all the forward-paths is important. The top layers that do temporal modeling learn global temporal information from all the frames and thus the learned features contain less redundancy as shown in Fig. 1. Dropping the forward-path as in frame-selection methods will impact the temporal modeling as the less redundant temporal model can only learn from a sampled set of frames. As shown in Fig. 4, keeping the gradients (with SBP) at the top layers has higher CKA similarity than dropping the gradients (with FD).

Comparison with Gradient Dropout [47]. Both SBP and Gradient Dropout drop the gradients during backward. Our SBP aims to save the memory by dropping a large proportion (*i.e.*, 75%) of the backward paths while Gradient Dropout aims to regularize the training by randomly zero-out a small proportion (10%~20%) of gradients. We explore the application of gradient dropout in a new direction and then design a strategy to drop the backward paths that can achieve practical memory savings.

3.2. Instantiations

Tree-models are typically adopted in long-term video tasks that take hundreds of frames as inputs. For short-term video tasks, such as action recognition, the fully-connected graph-models (*e.g.*, Video Transformers [2,3,27, 33]) achieve better accuracy. In this section, we instantiate on these two types of models.

3.2.1 Spatial-then-Temporal (StT) Models

For long-term tasks such as online action detection and offline action detection, most existing methods are Spatial-then-Temporal models (StT models). StT models are instances of tree-models (Fig. 3a). They first use a spatial model (2D- or 3D-CNN using sliding windows) to extract the spatial features for each frame / chunk and then feed the extracted features to a temporal model (RNN, Transformer, *etc.*) for temporal modeling.

Denote the spatial model as f_s and temporal model as f_t, the input video $x \in \mathbb{R}^{C\times D\times H\times W}$ is a clip of D frames with spatial resolution $H \times W$. We omit the batch dimension for simplicity. The spatial model takes input of size $\mathbb{R}^{C\times K\times H\times W}$, in which K is the chunk size. For example, 2D-CNN takes a single frame as input ($K = 1$) and 3D-CNN takes a chunk of K frames as input. So the forward pass of this model would be:

$$x_i = x[:, i*K : (i+1)*K], i \in \{0, \cdots, n\} \tag{1}$$
$$h = \{f_s(x_0), f_s(x_1), \cdots, f_s(x_n)\}, n = T/K - 1 \tag{2}$$
$$y = f_t(h) \tag{3}$$

In this model, the spatial model mainly extracts the spatial and local temporal features from the frames and temporal model gathers the global temporal semantic features from the output of spatial model. As temporal model contains less redundancy, SBP is only applied to the spatial model.

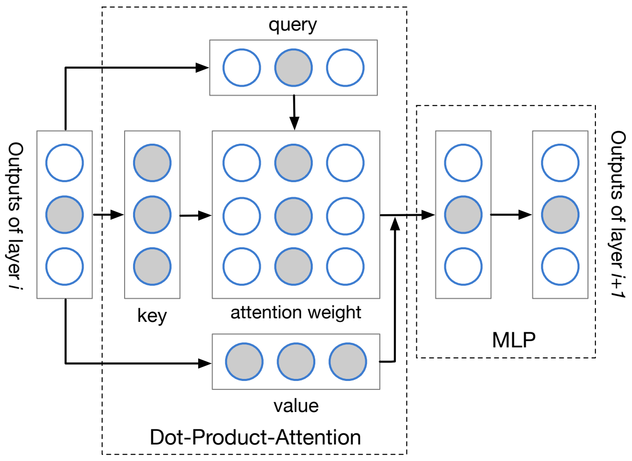


Figure 5. SBP on a Dot-Product Attention layer. In this example, the keep-ratio is $1/3$. Only sampled nodes (in color gray) are cached for gradient calculations in model training.

SBP can be implemented by dividing the input frames into two sets using a predefined sampling function: Set-1 $\{x_{i_0}, x_{i_1}, ..., x_{i_u}\}$ in which the gradients are kept and Set-2 $\{x_{j_0}, x_{j_1}, ..., x_{j_v}\}$ in which the gradients are dropped. During forward, we track all the activations for Set-1 and only track the extracted features for Set-2. Then we concatenate the extracted features and do fully forward and backward on the temporal model f_t.

3.2.2 Video Transformers

For short-term video tasks, such as action recognition, video Transformers such as TimeSformer [3] and Video Swin Transformer [33] achieve superior accuracy. Here we instantiate SBP on Video Transformers.

In the tree-models, the input node at one temporal location is independent of the nodes at other temporal locations in the spatial model, and thus we can easily apply SBP on it. However, in video Transformers, the nodes at different temporal locations are dependent on each other due to the dot-product attention layer. We address this issue by applying SBP layer-wise instead of model-wise in tree-models.

Fig. 5 shows how SBP is applied to a Transformer layer. The same set of nodes are sampled (gray nodes in Fig. 5) for all the layers. In the Dot-Product-Attention, all the key and value nodes are related to each of the query nodes. Given the gradients of the sampled nodes in the output, in order to calculate the gradients of the corresponding nodes in query accurately, all the key and value nodes are sampled. Sampling all the key and value nodes will only increase the memory consumption slightly as the attention weights comprise most of the GPU memory usage in Dot-Product-Attention. Suppose the shape of query, key and value is $\mathbb{R}^{(h\times d)\times n}$ and the shape of attention weights is $\mathbb{R}^{h\times n\times n}$, in which h, d and n are the number of heads, the dimension of each head and the number of tokens respectively. In video Transformers, the number of tokens n are much larger than the head dimension d (*i.e.*, in video Swin Transformer, $n = 392$ and $d = 32$) and thus attention weights comprise most of the GPU memory.

In Video Transformers, we consider the several top most layers (3 layers in this paper) served as "temporal model" that have fully backward paths and apply SBP only to the bottom layers (spatial model).

Algorithm 1 Pytorch-like pseudocode of SBP for forward and backward on an arbitrary operation f.

```
# f: an arbitrary operation
# idx: sampled indices where gradients are kept

# Forward function
def FORWARD(ctx, inputs, idx):
   # forward without gradient calculation
   with torch.no_grad():
      y = f(inputs)
   # cache sampled nodes
   ctx.save_for_backward(inputs[idx])
   # cache sampled index
   ctx.idx = idx
   return y

# Backward function
def BACKWARD(ctx, dy):
   dy = dy[ctx.idx]
   nodes = ctx.saved_tensors
   with torch.enable_grad():
      # reforward with gradient calculation
      nodes = nodes.detach().requires_grad_(True)
      y = f(nodes)
   d_nodes = torch.autograd.grad(y, nodes, dy)
   # gradients w.r.t inputs
   d_inputs = zeros.fill(ctx.idx, d_nodes)
   return d_inputs
```

3.3. Efficient Implementation

Alg. 1 shows the implementation for an arbitrary operation f, including the spatial model in StT, and the dot-product attention and multilayer perception (MLP) in Video Transformers. During forward, the output is produced without tracking the activations. Only the sampled inputs that need to calculate the gradients are saved for backward. During backward, SBP does the re-forward and backward on the sampled inputs to calculate the gradients. This can take full use of the "autograd engine" in the deep learning frameworks and is efficient with little overhead.

3.4. Space and Time Complexity

For the StT model, M_s is the memory needed by the spatial model and M_c is the memory needed by the temporal model. Thus its space ratio is $(rM_s + M_c)/(M_s + M_c)$, which is close to r since M_s is typically $5\times$ to $10\times$ larger than M_c [52, 54, 55]. For Video Transformers, d and n are the head dimensions in multi-head attention, number of tokens, respectively. The activation maps needed for gra-

Model	Training	K400 (%)		Epic-55 (%)		Mem (GB/gpu)
		Top-1	Top-5	Top-1	Top-5	
Swin-T	E2E	78.8	93.6	46.0	81.5	15.2
	FD	76.3	92.4	38.9	79.9	**2.9**
	SBP	**78.2**	**93.1**	**45.2**	**81.7**	4.4
Swin-B	E2E	82.7	95.5	49.4	83.2	32.5
	FD	80.8	94.9	44.8	82.5	**6.3**
	SBP	**81.9**	**95.2**	**47.3**	**83.6**	8.6

Table 1. Results of action recognition using Video Swin-T and -B. Both SBP and Frame Dropout (FD) use the keep-ratio at 0.25 here.

dient calculation in a Transformer layer includes 2× inputs ($2hdn$), QKV vectors (each with hdn), the attention weights ($2hnn$ before and after the softmax) and the MLP[2] ($10hdn$), thus requires the memory of $15hdn + 2hnn$ in total. SBP (as shown in Fig. 5) reduces the memory of query to $hdnr$, attention weights to $2hnnr$ and MLP to $10hdnr$. So the space ratio is

$$\frac{4hdn + 11hdnr + 2hnnr}{15hdn + 2hnn} = \frac{r(11\frac{d}{n} + 2) + 4\frac{d}{n}}{(11\frac{d}{n} + 2) + 4\frac{d}{n}} \quad (4)$$

For both the models, the space-ratio is almost proportional to the sampling rate r and thus the memory reduction of SBP can be significant if r is small (*i.e.*, 0.25). This is because during training, the most memory-consuming part is to cache the activations for gradient calculation, but with SBP, we only need to cache the sampled activations.

Regarding the time ratio, suppose the forward and backward time is equivalent, the overall ratio is $\frac{1+2r}{2}$ as we perform $(1 + r)\times$ the forward and $r\times$ the backward.

4. Experiments on Action Recognition

We evaluate SBP by applying it to state-of-the-art action recognition models, Video Swin [38], and conduct experiments on Kinetics-400 (K400) [24] and EPIC-Kitchens-55 (Epic-55) [12] datasets. K400 contains ∼240K training and 20K validation videos with 400 actions. Each video is trimmed to 10 seconds. Epic-55 includes egocentric videos of unscripted activities (mostly cooking) recorded in kitchen environments. Segments of each video are labeled with one verb and one noun. We follow prior work [50] to use their train and validation splits, and report the Top-1 and Top-5 accuracy on verb classification.

4.1. Implementation Details

We train Video Swin-T and -B [38] as backbones with batch size 4 on each GPU. We use the learning rate of $2.5e$-4 and $7.5e$-5 for Swin-T and -B respectively and train the models for 30 epochs. Other hyperparameters are kept the same as the original paper [38]. During inference, we follow prior work [50] by averaging the predictions of 4 × 3 views (*i.e.*, 4 uniform temporal cropping and 3 spatial cropping from top left, center, and bottom right) for K400 and using only central crop for Epic-55. For SBP, we use the uniform random sampler along the temporal dimension such that the nodes corresponding to one frame are either all sampled or dropped. The sampled nodes are kept the same across all layers. SBP is applied on the first 8 layers for Video Swin-T with layers {2,2,6,2}, and is applied on the first 18 layers for Video Swin-B with layers {2,2,18,2}.

[2]It consists of norm (hdn) → fc ($4hdn$) → activation ($4hdn$) → fc (hdn), where the hidden size of fc may vary but is $4hd$ in most vision Transformers [3, 33].

Model	Mem (GB/gpu)				Speed (s/itr)			
	without		with		without		with	
	E2E	SBP	E2E	SBP	E2E	SBP	E2E	SBP
Swin-T	15.2	4.4	5.8	**3.2**	0.22	**0.21**	0.29	0.26
Swin-B	32.5	8.6	8.0	**4.6**	0.43	**0.41**	0.57	0.52

Table 2. Benchmark of the memory usage and training speed of SBP with and without gradient checkpoint. The keep ratio is 0.25 for SBP and batch size is 4 per GPU. The memory is measured using *torch.cuda.max_memory_allocated()*. The speed is measured on a single Nvidia A100 graphics card.

Training	Sampling Dimension	K400 (%)		Epic-55 (%)	
		Top-1	Top-5	Top-1	Top-5
SBP	temporal	78.2	**93.1**	45.2	**81.7**
	spatial & temporal	**78.3**	93.0	**45.4**	81.3

Table 3. Temporal (frame-wise) and spatial-temporal sampling (3D-checkerboard) achieve comparable results. The model is Video Swin-T and the keep-ratio of SBP is 0.25.

4.2. Evaluation Results

Comparison to prior work. In Table 1, we compare the accuracy and memory usage of SBP (w/ keep-ratio as 0.25) to existing training strategies using Video Swin on K400 and Epic-55. Results show that SBP uses only 25% of memory as end-to-end (E2E) training uses, with only ∼0.7% Top-1 accuracy drop on K400 and ∼1.0% on Epic-55. Frame Dropout (FD) [1] is a simple frame selection method that only processes a subset of uniformly sampled frames. Different from SBP that only drops the backward paths, FD removes both forward and backward paths for the dropped frames. Table 1 shows that, although FD costs the least memory, it leads to more significant decrease of accuracy, especially on temporal-heavy dataset, Epic-55. This demonstrates the importance of keeping all the forward paths during training and inference.

Can SBP work with other efficient training strategies? We validate if SBP is complementary with other memory saving methods. As gradient checkpoint [9] is widely used to save memory, we report the memory and speed of SBP when integrated with gradient checkpoint. As shown

Method	Training	mAP (%)	GPU Mem (#GPUs × GB)
TRN [52]	Feat. Extract.	55.3	1 × 12
	E2E	56.8	8 × 14.6
	SBP (0.25)	56.7	8 × 5.6
	SBP (0.125)	**56.9**	8 × 4.3
LSTR [54]	Feat. Extract.	56.8	1 × 5
	E2E	59.2	8 × 32
	SBP (0.25)	59.1	8 × 12.2
	SBP (0.125)	**59.5**	8 × 7

Table 4. Comparison of SBP, using a fixed feature extractor (Feat. Extract.), and end-to-end (E2E) training on TRN and LSTR.

in Table 2, the memory consumption of SBP without checkpoint is similar to the standard end-to-end training with checkpoint, but is ∼30% faster. When applying gradient checkpoint to SBP, additional ∼40% the memory can be saved comparing to using SBP alone. Furthermore, we test to incorporate Mixed Precision (w/ opt_level as O1) into SBP. The results show that this can lead to ∼50% more savings of GPU memory on Swin-T without accuracy loss.

Can we drop gradients in both spatial and temporal domains? Till now, nodes corresponding to one frame are either all kept or dropped. To verify if SBP can be applied to both spatial and temporal dimensions, we implement the spatial-temporal gradient drop with a 3D-checkerboard pattern. From Table 3, temporal sampling (frame-wise) and spatial-temporal (3D-checkerboard) sampling achieve comparable accuracy, which shows SBP also works well along spatial sampling. This result indicates the potential to apply the proposed method to image domains.

5. Experiments on Temporal Action Detection

To validate SBP's efficiency for long-term video tasks, we build it upon multiple state-of-the-art methods [52, 54, 55] for temporal action detection in untrimmed videos. The experiments are conducted on THUMOS'14 [22] dataset, which contains about 20 hours of video with 20 sports actions. Since its training set contains only trimmed videos that cannot be used to train temporal action detection models, we follow prior work [52,54] and train on the validation set (200 videos) and evaluate on the test set (213 videos). Most experiments are conducted on 8 Tesla 16GB V100, except for the end-to-end training that requires more memory, the models are trained on 8 Tesla 32GB V100.

5.1. Online Action Detection

Given a live video stream, online action detection tries to detect the actions that are occurring in each frame without seeing the future. Existing methods are formulated in the spatial-then-temporal (StT) paradigm, where the spatial model is pretrained to extract features from the video frames, and the temporal model operates on the feature sequence to capture long-range context. We adopt two state-of-the-art methods (*i.e.*, TRN [52] and LSTR [54]) and conduct experiments using RGB as inputs[3]. In particular, TRN is built upon recurrent networks for temporal modeling and LSTR uses multiple cascaded Transformers to capture both long- and short-term dependencies.

5.1.1 Implementation Details

We follow the common setting of existing methods [52,54]. Specifically, the spatial model is ResNet50 pretrained on K400. The model is trained with batch size 16 and the initial learning rate is $1e$-6 and $5e$-5 for TRN and LSTR respectively. Note that both TRN and LSTR freeze their spatial model and only train the temporal model in the original papers. TRN takes 64 frames as inputs and LSTR takes 160 frames (*i.e.*, long-term 128 and short-term 32) as inputs. When converting it to end-to-end training using SBP, we set the learning rate of spatial model ∼10× smaller than the initial learning rate (this setting is validated in the later section). As both TRN and LSTR are StT models, we use the implementation as described in Sec. 3.2. Per-frame mean average precision (mAP) is adopted to evaluate the performance of online action detection.

5.1.2 Evaluation Results

The results are shown in Table 4. First, freezing the backbone leads to 1.5% and 2.4% mAP drop for TRN and LSTR respectively. Second, by applying SBP with keep-ratio of 0.125, we can train the model with 22% of the memory with similar mAPs compared with end-to-end training. Third, one interesting observation is that SBP with lower keep-ratio r=0.125 is even slightly higher than the end-to-end training, which may be the reason that SBP can be served as a regularization to ease overfitting on THUMOS dataset that is relatively small [47]. Fourth, the improvement on LSTR is larger than that on TRN, which indicates a stronger temporal model can benefit more from the end-to-end training.

5.1.3 Ablation Studies

How to sample nodes? One key aspect of SBP is how to sample the output nodes of the top most layer that SBP is applied to (named as *candidate nodes* for abbreviation). We examine the three sampling methods as below. (1) *Uniform Random Sampler* evenly divides the candidates nodes into chunks along temporal dimension. The size of each chunk is $1/r$. We randomly sample 1 node inside each chunk. (2) *Diverse-Feature Sampler* first sorts the candidate nodes according to the magnitude and then performs uniform random sampling on the sorted node list. (3) *Diverse-Grad*

[3]In the original papers, they ensemble RGB and optical flow as inputs. Without loss of generality, we only use RGB in our experiments.

Training	Keep-ratio	mAP (%)						GPU Mem (#GPUs × GB)
		0.3	0.4	0.5	0.6	0.7	AVG	
Feat. Extract.	-	43.15	35.84	28.36	19.89	11.96	27.84	**8×3**
End-to-End	-	46.14	**39.86**	**32.02**	**23.29**	**14.56**	**31.17**	8×26
SBP	0.5	45.36	**38.72**	**31.46**	22.74	14.01	**30.46**	8×12
	0.25	45.12	37.80	30.44	**22.81**	**14.52**	30.14	8×6
	0.125	**46.40**	38.50	29.95	20.37	11.82	29.41	**8×5**

Table 5. Results of temporal action localization on THUMOS'14. We only use RGB frames as inputs for simplicity. Note that the end-to-end training is almost impossible for many research groups but SBP with keep ratio 0.5 can be trained on mainstream GPUs (12 GB).

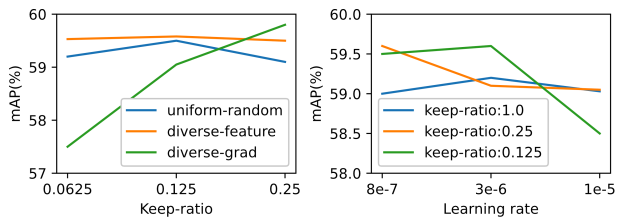


Figure 6. The influence to the performance in mAP of using different sampling methods and initial learning rates for spatial model. The initial learning rate for temporal model is 5e-5.

Sampler sorts the gradients of the candidate nodes according to the magnitude and then performs uniform random sampling on the sorted node list. From the results in Fig. 6, we can observe that: 1) the diverse-feature sampler achieves overall the best mAP; 2) the mAP of these three samplers are quite similar, which indicates our method is robust to the samplers. As the uniform random sampler achieves comparable mAP with others but is much simpler to implement (preferred from the perspective of Occam's razor), we adopt it as our default sampler for the following experiments.

How to set learning rate for the spatial model? We do some empirical studies to determine the best learning rate for training StT models with SBP. The results are shown in Fig. 6. We observe that: 1) the best learning rate for spatial model is about $10\times$ smaller than the temporal model for all the keep-ratios; 2) training with SBP, the spatial model is quite robust to the learning rate as long as they are relative small. Therefore, unless noted otherwise, we set the learning rate of spatial model as $0.1\times$ of the temporal model.

5.2. Temporal Action Localization

Temporal Action localization aims to detect the temporal boundaries for every action instance in untrimmed videos. Most existing methods [17,30,55] follow the StT paradigm. We take the state-of-the-art method, G-TAD [55] as baseline and conduct experiments on THUMOS'14 [23].

5.2.1 Implementation Details

We use the same settings as in the original paper [55]: the spatial model is ResNet50 pretrained on K400 and the learning rate for temporal model is $4e$-5. When integrating it with SBP, we unfreeze the spatial model and set the learning rate for spatial model to $5e$-5. We take mean Average Precision (mAP) at certain IoUs as the evaluation metric.

5.2.2 Evaluation Results

The results are as shown in Tab 5. Freezing the backbone leads to 3.3% drop in average mAP. SBP is ~1% lower than end-to-end training but can still lead to ~2% improvement comparing to using frozen feature extractor. Training with SBP uses only 12 GB, 6 GB and 5 GB on each GPU compared with 26GB using end-to-end training for keep-ratio of 0.5, 0.25 and 0.125 respectively. Note the amount of memory (8 GPUs, each with 26GB) for end-to-end training may not be available for many research groups. But applying SBP with keep-ratio 0.25 only gets 1% loss of mAP, the model can be trained on 4 GPUs with 12 GB memory (*i.e.*, $4\times$ GTX 1080Ti), which is much more feasible.

6. Discussion

We presented a simple yet effective technique Stochastic Backpropagation (SBP) that can reduce a large portion of GPU memory usage for training video models. However, there are still several limitations of our current work. First, SBP is a new prototype to show that memory saving can be achieved by dropping the backward paths, but as byproduct, it still causes slightly loss of accuracy. Future explorations on strategies of using more adaptive layer-wise keep-ratios and sampling methods can be studied to reduce the loss of accuracy or even improve the accuracy due to the regularization effect of gradient dropout [47]. Second, SBP brings only a small amount of training speedup ($\sim 1.1\times$). However, how to integrate it with other efficient model training strategies, such as Multigrid [51], to speed up the training is still not well explored.

Potential Negative Impact. We study the research problem of efficient training for video models, which can be used in real-world video understanding applications, such as patient or elderly health monitoring, augmented and virtual reality, and collaborative robots. But there still could be unintended uses, and we advocate responsible usage complying applicable laws and regulations.

References

[1] Hassan Akbari, Linagzhe Yuan, Rui Qian, Wei-Hong Chuang, Shih-Fu Chang, Yin Cui, and Boqing Gong. Vatt: Transformers for multimodal self-supervised learning from raw video, audio and text. *NeurIPS*, 2021. 1, 2, 4, 6

[2] Anurag Arnab, Mostafa Dehghani, Georg Heigold, Chen Sun, Mario Lučić, and Cordelia Schmid. Vivit: A video vision transformer. *ICCV*, 2021. 2, 4

[3] Gedas Bertasius, Heng Wang, and Lorenzo Torresani. Is space-time attention all you need for video understanding? *ICML*, 2021. 1, 2, 3, 4, 5, 6

[4] Shyamal Buch, Victor Escorcia, Chuanqi Shen, Bernard Ghanem, and Juan Carlos Niebles. SST: Single-stream temporal action proposals. In *CVPR*, 2017. 2

[5] Mikhail S Burtsev, Yuri Kuratov, Anton Peganov, and Grigory V Sapunov. Memory transformer. *arXiv:2006.11527*, 2020. 2

[6] 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

[7] Joao Carreira and Andrew Zisserman. Quo vadis, action recognition? a new model and the kinetics dataset. In *CVPR*, 2017. 1, 2

[8] 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

[9] Tianqi Chen, Bing Xu, Chiyuan Zhang, and Carlos Guestrin. Training deep nets with sublinear memory cost. *arXiv:1604.06174*, 2016. 2, 6

[10] Xiyang Dai, Bharat Singh, Guyue Zhang, Larry S Davis, and Yan Qiu Chen. Temporal context network for activity localization in videos. In *ICCV*, 2017. 2

[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. 2

[12] Dima Damen, Hazel Doughty, Giovanni Maria Farinella, Sanja Fidler, Antonino Furnari, Evangelos Kazakos, Davide Moltisanti, Jonathan Munro, Toby Perrett, Will Price, et al. Scaling egocentric vision: The epic-kitchens dataset. In *ECCV*, 2018. 2, 6

[13] Roeland De Geest, Efstratios Gavves, Amir Ghodrati, Zhenyang Li, Cees Snoek, and Tinne Tuytelaars. Online action detection. In *ECCV*, 2016. 1, 2

[14] Tim Dettmers and Luke Zettlemoyer. Sparse networks from scratch: Faster training without losing performance. *arXiv:1907.04840*, 2019. 2

[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. *arXiv:2010.11929*, 2020. 3

[16] Hyunjun Eun, Jinyoung Moon, Jongyoul Park, Chanho Jung, and Changick Kim. Learning to discriminate information for online action detection. In *CVPR*, 2020. 2

[17] Jiyang Gao, Zhenheng Yang, Chen Sun, Kan Chen, and Ram Nevatia. TURN TAP: Temporal unit regression network for temporal action proposals. *ICCV*, 2017. 2, 8

[18] Mingfei Gao, Mingze Xu, Larry S Davis, Richard Socher, and Caiming Xiong. Startnet: Online detection of action start in untrimmed videos. *ICCV*, 2019. 2

[19] Mingfei Gao, Yingbo Zhou, Ran Xu, Richard Socher, and Caiming Xiong. WOAD: Weakly supervised online action detection in untrimmed videos. In *CVPR*, 2021. 2

[20] Shreyank N Gowda, Marcus Rohrbach, and Laura Sevilla-Lara. Smart frame selection for action recognition. *arXiv:2012.10671*, 2020. 2, 4

[21] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016. 1, 3

[22] 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. 2, 7

[23] 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. 8

[24] 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:1705.06950*, 2017. 6

[25] Bruno Korbar, Du Tran, and Lorenzo Torresani. Scsampler: Sampling salient clips from video for efficient action recognition. In *ICCV*, 2019. 4

[26] Simon Kornblith, Mohammad Norouzi, Honglak Lee, and Geoffrey Hinton. Similarity of neural network representations revisited. In *ICML*, 2019. 4

[27] 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. 2, 4

[28] Yin Li, Zhefan Ye, and James M Rehg. Delving into egocentric actions. In *CVPR*, 2015. 2

[29] 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. 2

[30] Tianwei Lin, Xiao Liu, Xin Li, Errui Ding, and Shilei Wen. Bmn: Boundary-matching network for temporal action proposal generation. In *CVPR*, 2019. 2, 8

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

[32] Xin Liu, Silvia L Pintea, Fatemeh Karimi Nejadasl, Olaf Booij, and Jan C van Gemert. No frame left behind: Full video action recognition. In *CVPR*, 2021. 2

[33] Ze Liu, Jia Ning, Yue Cao, Yixuan Wei, Zheng Zhang, Stephen Lin, and Han Hu. Video swin transformer. *arXiv:2106.13230*, 2021. 1, 2, 3, 4, 5, 6

[34] Minghuang Ma, Haoqi Fan, and Kris M Kitani. Going deeper into first-person activity recognition. In *CVPR*, 2016. 2

[35] Mateusz Malinowski, Grzegorz Swirszcz, Joao Carreira, and Viorica Patraucean. Sideways: Depth-parallel training of video models. In *CVPR*, 2020. 2

[36] Mateusz Malinowski, Dimitrios Vytiniotis, Grzegorz Swirszcz, Viorica Patraucean, and Joao Carreira. Gradient forward-propagation for large-scale temporal video modelling. In *CVPR*, 2021. 2

[37] Megha Nawhal and Greg Mori. Activity graph transformer for temporal action localization. *arXiv:2101.08540*, 2021. 2

[38] Daniel Neimark, Omri Bar, Maya Zohar, and Dotan Asselmann. Video transformer network. *arXiv:2102.00719*, 2021. 6

[39] 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. 2

[40] 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. 2

[41] 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. 2

[42] Karen Simonyan and Andrew Zisserman. Two-stream convolutional networks for action recognition in videos. *NeurIPS*, 2014. 1, 2

[43] Waqas Sultani, Chen Chen, and Mubarak Shah. Real-world anomaly detection in surveillance videos. In *CVPR*, 2018. 2

[44] Jing Tan, Jiaqi Tang, Limin Wang, and Gangshan Wu. Relaxed transformer decoders for direct action proposal generation. *arXiv:2102.01894*, 2021. 2

[45] Du Tran, Lubomir Bourdev, Rob Fergus, Lorenzo Torresani, and Manohar Paluri. Learning spatiotemporal features with 3d convolutional networks. In *ICCV*, 2015. 1, 2

[46] 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. 2

[47] Hung-Yu Tseng, Yi-Wen Chen, Yi-Hsuan Tsai, Sifei Liu, Yen-Yu Lin, and Ming-Hsuan Yang. Regularizing meta-learning via gradient dropout. In *ACCV*, 2020. 4, 7, 8

[48] Chenhao Wang, Hongxiang Cai, Yuxin Zou, and Yichao Xiong. Rgb stream is enough for temporal action detection. *arXiv:2107.04362*, 2021. 2

[49] 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. 1, 2

[50] 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. 1, 2, 6

[51] Chao-Yuan Wu, Ross Girshick, Kaiming He, Christoph Feichtenhofer, and Philipp Krahenbuhl. A multigrid method for efficiently training video models. In *CVPR*, 2020. 8

[52] Mingze Xu, Mingfei Gao, Yi-Ting Chen, Larry S Davis, and David J Crandall. Temporal recurrent networks for online action detection. In *ICCV*, 2019. 2, 3, 5, 7

[53] Mingze Xu, Aidean Sharghi, Xin Chen, and David J Crandall. Fully-coupled two-stream spatiotemporal networks for extremely low resolution action recognition. In *WACV*, 2018. 2

[54] Mingze Xu, Yuanjun Xiong, Hao Chen, Xinyu Li, Wei Xia, Zhuowen Tu, and Stefano Soatto. Long short-term transformer for online action detection. *NeurIPS*, 2021. 2, 3, 5, 7

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

[56] Peisen Zhao, Jiajie Wang, Lingxi Xie, Ya Zhang, Yanfeng Wang, and Qi Tian. Privileged knowledge distillation for online action detection. *arXiv:2011.09158*, 2020. 2

[57] Yue Zhao, Yuanjun Xiong, Limin Wang, Zhirong Wu, Xiaoou Tang, and Dahua Lin. Temporal action detection with structured segment networks. In *ICCV*, 2017. 2