TSM: Temporal Shift Module for Efficient and Scalable Video Understanding on Edge Devices

TSM: Temporal Shift Module for Efficient and Scalable Video Understanding on Edge Devices

Ji Lin, Chuang Gan, Kuan Wang and Song Han

Abstract—The explosive growth in video streaming requires video understanding at high accuracy and low computation cost. Conventional 2D CNNs are computationally cheap but cannot capture temporal relationships; 3D CNN based methods can achieve good performance but are computationally intensive. In this paper, we propose a generic and effective Temporal Shift Module (TSM) that enjoys both high efficiency and high performance. The key idea of TSM is to shift part of the channels along the temporal dimension, thus facilitate information exchanged among neighboring frames. It can be inserted into 2D CNNs to achieve temporal modeling at zero computation and zero parameters. TSM offers several unique advantages. Firstly, TSM has high performance; it ranks the first on the Something-Something leaderboard upon submission. Secondly, TSM has high efficiency; it achieves a high frame rate of 74fps and 29fps for online video recognition on Jetson Nano and Galaxy Note8. Thirdly, TSM has higher scalability compared to 3D networks, enabling large-scale Kinetics training on 1,536 GPUs in 15 minutes. Lastly, TSM enables action concepts learning, which 2D networks cannot model; we visualize the category attention map and find that spatial-temporal action detector emerges during the training of classification tasks. The code is publicly available at https://github.com/mit-han-lab/temporal-shift-module.

Index Terms—Temporal Shift Module, Video Recognition, Video Object Detection, Distributed Training, Edge Device, Network Dissection.

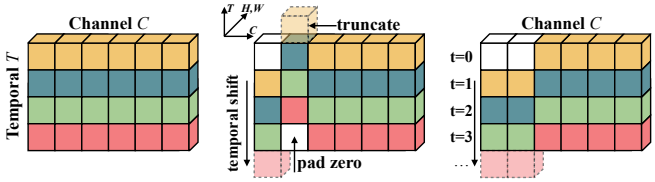
1 Introduction

Hardware-efficient video understanding is an important step towards real-world deployment, both on the cloud and on the edge. For example, there are over 10^5 hours of videos uploaded to YouTube every day to be processed for recommendation and ads ranking; tera-bytes of sensitive videos in hospitals need to be processed locally on edge devices to protect privacy. All these industry applications require both accurate and efficient video understanding.

Deep learning has become the standard for video understanding over the years [4], [52], [58], [59], [63], [68], [72]. One key difference between video recognition and image recognition is the need for *temporal modeling*. For example, to distinguish between opening and closing a box, reversing the order will give opposite results, so temporal modeling is critical. Existing efficient video understanding approaches directly use 2D CNN [28], [46], [58], [68]. However, 2D CNN on individual frames could not capture the temporal information very well. 3D CNNs [4], [52] can jointly learn spatial and temporal features but the computation cost is large, making the deployment on edge devices difficult; it cannot be applied to real-time online video recognition. There are works to trade off between temporal modeling and computation, such as post-hoc fusion [8], [12], [16], [68] and mid-level temporal fusion [54], [63], [72]. Such methods sacrifice the low-level temporal modeling for efficiency, but much of the useful information is lost during the feature extraction before the temporal fusion happens.

In this paper, we propose a new perspective for efficient temporal modeling in video understanding by proposing a novel Temporal Shift Module (TSM). Concretely, an activation in a video model can be represented as $A \in \mathbb{R}^{N \times C \times T \times H \times W}$, where N is the batch size, C is the number of channels, T is the temporal dimension, H and W are the spatial resolutions. Traditional 2D CNNs operate independently over the dimension T; thus no temporal modeling takes effects (Figure 1a). In contrast, our Temporal Shift Module (TSM) shifts the channels along the temporal dimension, both forward and backward. As shown in Figure 1b, the information from neighboring frames is mingled with the current frame after shifting. Our intuition is: the convolution operation consists of *shift* and *multiply-accumulate*. We *shift* in the time dimension by ± 1 and fold the *multiply-accumulate* from time dimension to channel dimension. For real-time online video understanding, future frames can't get shifted to the present, so we use a uni-directional TSM (Figure 1c) to perform online video understanding.

Despite the zero-computation nature of the shift operation, we empirically find that simply adopting the spatial shift strategy [61]



(a) The original tensor without shift. **(b)** Offline temporal shift (bi-direction). **(c)** Online temporal shift (uni-direction).

Fig. 1. Temporal Shift Module (TSM) performs efficient temporal modeling by moving the feature map along the temporal dimension. It is computationally free on top of a 2D convolution, but achieves strong temporal modeling ability. TSM efficiently supports both **offline** and **online** video recognition. Bi-directional TSM mingles both past and future frames with the current frame, which is suitable for high-throughput offline video recognition. Uni-directional TSM mingles only the past frame with the current frame, which is suitable for low-latency online video recognition.

- *J. Lin, K. Wang, S. Han are with the Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology. E-mail: {jilin, kuanwang, songhan}@mit.edu*
- *C. Gan is with MIT-IBM Watson AI Lab. E-mail: ganchuang@csail.mit.edu*

used in image classifications introduces two major issues for video understanding: (1) it is *not efficient*: shift operation is conceptually zero FLOP but incurs data movement. The additional cost of data movement is non-negligible and will result in latency increase. This phenomenon has been exacerbated in the video networks since they usually have a large memory consumption (5D activation). (2) It is *not accurate*: shifting too many channels in a network will significantly hurt the spatial modeling ability and result in performance degradation. To tackle the problems, we make two technical contributions. (1) We use a *temporal partial shift* strategy: instead of shifting all the channels, we shift only a small portion of the channels for efficient temporal fusion. Such strategy significantly cuts down the data movement cost (Figure 2a). (2) We insert TSM inside *residual branch* rather than outside so that the activation of the current frame is preserved, which does not harm the spatial feature learning capability of the 2D CNN backbone.

To verify the effectiveness of TSM, we carried out comprehensive experiments: (1) we show that TSM consistently improve the video recognition performance compared to 2D model without incurring extra computation or parameters; it also achieves state-of-the-art performance on multiple action recognition dataset; (2) TSM has much better accuracy-computation trade-off compared to prior works. The contributions of our paper are summarized as follows:

- We provide a new perspective for efficient video model design by temporal shift, which is computationally free but has strong spatio-temporal modeling ability.
- We observed that naive shift cannot achieve high efficiency or high performance. We then proposed two technical modifications *partial shift* and *residual shift* to realize a high efficiency model design.
- We propose *bi-directional TSM* for *offline* video understanding that achieves state-of-the-art performance. It ranks the first on Something-Something leaderboard upon publication.
- We propose *uni-directional TSM* for *online* real-time video recognition with strong temporal modeling capacity at low latency on edge devices.
- With the efficient design of TSM, we scale up the training of video network to 1,536 GPUs, and finish the training on Kinetics dataset in 15 minutes, without losing accuracy. To the best of our knowledge, we are the first to systematically investigate the large-scale training on video recognition.
- We provide an in-depth analysis to understand the learned knowledge inside the TSM module, and find that spatial-temporal action detector automatically emerges during training using only classification labels.

2 Related Work

In this section, we briefly review four related topics: 1) Deep Video Recognition, 2) Temporal Modeling, and 3) Efficient Neural Networks.

2.1 Deep Video Recognition

2.1.1 2D CNN.

Using the 2D CNN is a straightforward way to conduct video recognition [2], [11], [12], [14], [28], [46], [58]. For example, Simonyan *et al*. [46] designed a two-stream CNN for RGB input (spatial stream) and optical flow [65] input (temporal stream) respectively. Temporal Segment Networks (TSN) [58] extracted averaged features from strided sampled frames. Such methods are more efficient compared to 3D counterparts but cannot infer the temporal order or more complicated temporal relationships.

2.1.2 3D CNN.

3D convolutional neural networks can jointly learn spatio-temporal features. Tran *et al*. [52] proposed a 3D CNN based on VGG models, named C3D, to learn spatio-temporal features from a frame sequence. Carreira and Zisserman [4] proposed to inflate all the 2D convolution filters in an Inception V1 model [50] into 3D convolutions. However, 3D CNNs are computationally heavy, making the deployment difficult. They also have more parameters than 2D counterparts, thus are more prone to over-fitting. On the other hand, our TSM has the same spatial-temporal modeling ability as 3D CNN while enjoying the same computation and parameters as the 2D CNNs.

2.1.3 Trade-offs.

There have been attempts to trade off expressiveness and computation costs. Lee *et al*. [31] proposed a motion filter to generate spatio-temporal features from 2D CNN. Tran *et al*. [54] and Xie *et al*. [63] proposed to study mixed 2D and 3D networks, either first using 3D and later 2D (bottom-heavy) or first 2D and later 3D (top-heavy) architecture. ECO [72] also uses a similar top-heavy architecture to achieve a very efficient framework. Another way to save computation is to decompose the 3D convolution into a 2D spatial convolution and a 1D temporal convolution [39], [49], [54]. For mixed 2D-3D CNNs, they still need to remove low-level temporal modeling or high-level temporal modeling. Compared to decomposed convolutions, our method completely removes the computation cost of temporal modeling has enjoys better hardware efficiency.

2.2 Temporal Modeling

A direct way for temporal modeling is to use 3D CNN based methods as discussed above. Wang *et al*. [59] proposed a spatial-temporal non-local module to capture long-range dependencies. Wang *et al*. [60] proposed to represent videos as space-time region graphs. An alternative way to model the temporal relationships is to use 2D CNN + post-hoc fusion [8], [12], [16], [68]. Some works use LSTM [24] to aggregate the 2D CNN features [8], [13], [15], [48], [64]. Attention mechanism also proves to be effective for temporal modeling [32], [37], [44]. Zhou *et al*. [68] proposed Temporal Relation Network to learn and reason about temporal dependencies. The former category is computational heavy, while the latter cannot capture the useful low-level information that is lost during feature extraction. Our method offers an efficient solution at the cost of 2D CNNs, while enabling both low-level and high-level temporal modeling, just like 3D-CNN based methods.

2.3 Efficient Neural Networks

The efficiency of 2D CNN has been extensively studied. Some works focused on designing an efficient model [25], [26], [42], [66]. Recently neural architecture search [36], [73], [74] has been introduced to find an efficient architecture automatically [3], [51]. Another way is to prune, quantize and compress an existing model for efficient deployment [19], [20], [23], [34], [57], [70]. Address shift, which is a hardware-friendly primitive, has also been exploited for compact 2D CNN design on image recognition tasks [61], [67]. Nevertheless, we observe that directly adopting the shift operation on video recognition task neither maintains efficiency nor accuracy, due to the complexity of the video data.

3 Temporal Shift Module (TSM)

We first explain the intuition behind TSM: data movement and computation can be separated in a convolution. However, we observe that such naive shift operation neither achieves high efficiency nor high performance. To tackle the problem, we propose two techniques minimizing the data movement and increasing the model capacity, which leads to the efficient TSM module.

3.1 Intuition

Let us first consider a normal convolution operation. For brevity, we used a 1-D convolution with the kernel size of 3 as an example. Suppose the weight of the convolution is $W = (w_1, w_2, w_3)$, and the input X is a 1-D vector with infinite length. The convolution operator $Y = \text{Conv}(W, X)$ can be written as: $Y_i = w_1 X_{i-1} + w_2 X_i + w_3 X_{i+1}$. We can decouple the operation of convolution into two steps: *shift* and *multiply-accumulate*: we shift the input X by $-1, 0, +1$ and multiply by w_1, w_2, w_3 respectively, which sum up to be Y. Formally, the *shift* operation is:

$$X_i^{-1} = X_{i-1}, \quad X_i^0 = X_i, \quad X_i^{+1} = X_{i+1} \tag{1}$$

and the *multiply-accumulate* operation is:

$$Y = w_1 X^{-1} + w_2 X^0 + w_3 X^{+1} \tag{2}$$

The first step *shift* can be conducted without any multiplication. While the second step is more computationally expensive, our Temporal Shift module *merges* the *multiply-accumulate* into the following 2D convolution, so it introduces no extra cost compared to 2D CNN based models.

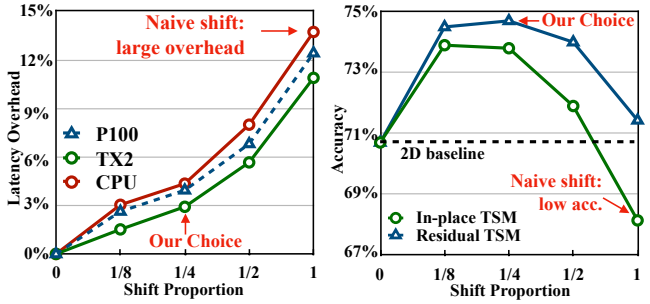
The proposed Temporal Shift module is described in Figure 1. In Figure 1a, we describe a tensor with C channels and T frames. The features at different time stamps are denoted as different colors in each row. Along the temporal dimension, we shift part of the channels by -1, another part by $+1$, leaving the rest un-shifted (Figure 1b). For online video recognition setting, we also provide an online version of TSM (Figure 1c). In the online setting, we cannot access future frames, therefore, we only shift from past frames to future frames in a uni-directional fashion.

3.2 Naive Shift Does Not Work

Despite the simple philosophy behind the proposed module, we find that directly applying the spatial shift strategy [61] to the temporal dimension cannot provide high performance nor efficiency. To be specific, if we shift all or most of the channels, it brings two disasters: **(1) Worse efficiency due to large data movement**. The shift operation enjoys no computation, but it involves data movement. Data movement increases the memory footprint and inference latency on hardware. Worse still, such effect is exacerbated in the video understanding networks due to large activation size (5D tensor). When using the naive shift strategy shifting every map, we observe a 13.7% increase in CPU latency and 12.4% increase in GPU latency, making the overall inference slow. **(2) Performance degradation due to worse spatial modeling ability.** By shifting part of the channels to neighboring frames, the information contained in the channels is no longer accessible for the current frame, which may harm the spatial modeling ability of the 2D CNN backbone. We observe a 2.6% accuracy drop when using the naive shift implementation compared to the 2D CNN baseline (TSN).

3.3 Module Design

To tackle the two problem from naive shift implementation, we discuss two technical contributions.



(a) Overhead *vs*. proportion. **(b)** Residual *vs*. in-place.

Fig. 2. (a) Latency overhead of TSM due to data movement. **(b)** Residual TSM achieve better performance than in-place shift. We choose 1/4 proportion residual shift as our default setting. It achieves higher accuracy with a negligible overhead.

3.3.1 *Reducing Data Movement.*

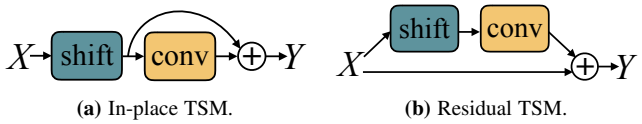
To study the effect of data movement, we first measured the inference latency of TSM models and 2D baseline on different hardware devices. We shifted different proportion of the channels and measured the latency. We measured models with ResNet-50 backbone and 8-frame input using no shift (2D baseline), partial shift ($1/8, 1/4, 1/2$) and all shift (shift all the channels). The timing was measure on server GPU (NVIDIA Tesla P100), mobile GPU (NVIDIA Jetson TX2) and CPU (Intel Xeon E5-2690). We report the average latency from 1000 runs after 200 warm-up runs. We show the overhead of the shift operation as the percentage of the original 2D CNN inference time in 2a. We observe the same overhead trend for different devices. If we shift all the channels, the latency overhead takes up to **13.7%** of the inference time on CPU, which is definitely **non-negligible** during inference. On the other hand, if we only shift a small proportion of the channels, *e.g.*, $1/8$, we can limit the latency overhead to **only 3%**. Therefore, we use *partial shift* strategy in our TSM implementation to significantly bring down the memory movement cost.

3.3.2 *Keeping Spatial Feature Learning Capacity.*

We need to balance the model capacity for spatial feature learning and temporal feature learning. A straight-forward way to apply TSM is to insert it before each convolutional layer or residual block, as illustrated in Figure 3a. We call such implementation *in-place shift*. It harms the spatial feature learning capability of the backbone model, especially when we shift a large amount of channels, since the information stored in the shifted channels is lost for the current frame.

To address such issue, we propose a variant of the shift module. Instead of inserting it in-place, we put the TSM *inside* the residual branch in a residual block. We denote such version of shift as *residual shift* as shown in 3b. Residual shift can address the degraded spatial feature learning problem, as all the information in the original activation is still accessible after temporal shift through identity mapping.

To verify our assumption, we compared the performance of in-place shift and residual shift on Kinetics [29] dataset. We studied the experiments under different shift proportion setting. The results are shown in 2b. We can see that residual shift achieves better performance than in-place shift for all shift proportion. Even we shift all the channels to neighboring frames, due to the shortcut connection, residual shift still achieves better performance than the



(a) In-place TSM. **(b)** Residual TSM.

Fig. 3. Residual shift is better than in-place shift. In-place shift happens before a convolution layer (or a residual block). Residual shift fuses temporal information inside a residual branch.

2D baseline. Another finding is that the performance is related to the proportion of shifted channels: if the proportion is too small, the ability of temporal reasoning may not be enough to handle complicated temporal relationships; if too large, the spatial feature learning ability may be hurt. For residual shift, we found that the performance reaches the peak when $1/4$ ($1/8$ for each direction) of the channels are shifted. Therefore, we use this setting for the rest of the paper.

4 TSM Video Network

4.1 Offline Models with Bi-directional TSM

We insert bi-directional TSM to build offline video recognition models. Given a video V, we first sample T frames $F_i, F_1, ..., F_T$ from the video. After frame sampling, 2D CNN baselines process each of the frames individually, and the output logits are averaged to give the final prediction. Our proposed TSM model has exactly the same parameters and computation cost as 2D model. During the inference of convolution layers, the frames are still running independently just like the 2D CNNs. The difference is that TSM is inserted for each residual block, which enables temporal information fusion at no computation. For each inserted temporal shift module, the temporal receptive field will be enlarged by 2, as if running a convolution with the kernel size of 3 along the temporal dimension. Therefore, our TSM model has a very large temporal receptive field to conduct highly complicated temporal modeling. In this paper, we used ResNet-50 [22] as the backbone unless otherwise specified.

A unique advantage of TSM is that it can easily convert any off-the-shelf 2D CNN model into a pseudo-3D model that can handle both spatial and temporal information, without adding additional computation. Thus the deployment of our framework is hardware friendly: we only need to support the operations in 2D CNNs, which are already well-optimized at both framework level (CuDNN [6], MKL-DNN, TVM [5]) and hardware level (CPU/GPU/TPU/FPGA).

4.2 Online Models with Uni-directional TSM

Video understanding from online video streams is important in real-life scenarios. Many real-time applications requires online video recognition with low latency, such as AR/VR and self-driving. In this section, we show that we can adapt TSM to achieve online video recognition while with multi-level temporal fusion.

As shown in Figure 1, offline TSM shifts part of the channels bi-directionally, which requires features from future frames to replace the features in the current frame. If we only shift the feature from previous frames to current frames, we can achieve online recognition with uni-directional TSM.

The inference graph of uni-directional TSM for online video recognition is shown in Figure 4. During inference, for each frame, we save the first 1/8 feature maps of each residual block and cache it in the memory. For the next frame, we replace the first 1/8 of the current feature maps with the cached feature maps. We use the

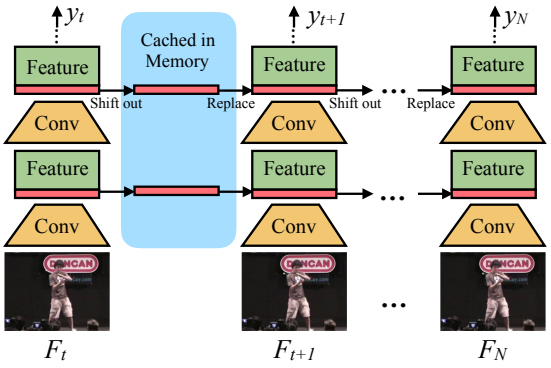


Fig. 4. Uni-directional TSM for online video recognition.

combination of 7/8 current feature maps and 1/8 old feature maps to generate the next layer, and repeat. Using the uni-directional TSM for online video recognition shares several unique advantages:

1. Low latency inference. For each frame, we only need to replace and cache 1/8 of the features, without incurring any extra computations. Therefore, the latency of giving per-frame prediction is almost the same as the 2D CNN baseline. Existing methods like [72] use multiple frames to give one prediction, which may leads to large latency.

2. Low memory consumption. Since we only cache a small portion of the features in the memory, the memory consumption is low. For ResNet-50, we only need 0.9MB memory cache to store the intermediate feature.

3. Multi-level temporal fusion. Most of the online method only enables late temporal fusion after feature extraction like [68] or mid level temporal fusion [72], while our TSM enables all levels of temporal fusion. Through experiments (Table 2) we find that multi-level temporal fusion is very important for complex temporal modeling.

5 Experiments

We first show that TSM can significantly improve the performance of 2D CNN on video recognition while being computationally free and hardware efficient. It further demonstrated state-of-the-art performance on temporal-related datasets, arriving at a much better accuracy-computation pareto curve. TSM models achieve an order of magnitude speed up in measured GPU throughput compared to conventional I3D model from [60]. Finally, we leverage uni-directional TSM to conduct low-latency and real-time online prediction on both video recognition and object detection.

5.1 Setups

5.1.1 *Training & Testing.*

We conducted experiments on video action recognition tasks. The training parameters for the Kinetics dataset are: 100 training epochs, initial learning rate 0.01 (decays by 0.1 at epoch 40&80), weight decay 1e-4, batch size 64, and dropout 0.5. For other datasets, we scale the training epochs by half. For most of the datasets, the model is fine-tuned from ImageNet pre-trained weights; while HMDB-51 [30] and UCF-101 [47] are too small and prone to over-fitting [58], we followed the common practice [58], [59] to fine-tune from Kinetics [29] pre-trained weights and freeze the Batch Normalization [27] layers. For testing, when pursue high accuracy, we followed the common setting in [59], [60] to sample multiple clips per video (10 for Kinetics, 2 for others) and use the

	Dataset	Model	Acc1	Acc5	△ Acc1
Less Temporal	Kinetics	TSN	70.6	89.2	+3.5
		Ours	**74.1**	**91.2**	
	UCF101	TSN	91.7	99.2	+4.2
		Ours	**95.9**	**99.7**	
	HMDB51	TSN	64.7	89.9	+8.8
		Ours	**73.5**	**94.3**	
More Temporal	Something V1	TSN	20.5	47.5	+28.0
		Ours	**47.3**	**76.2**	
	Something V2	TSN	30.4	61.0	+31.3
		Ours	**61.7**	**87.4**	
	Jester	TSN	83.9	99.6	+11.7
		Ours	**97.0**	**99.9**	

TABLE 1. Our method consistently outperforms 2D counterparts on multiple datasets at zero extra computation (protocol: ResNet-50 8f input, 10 clips for Kinetics, 2 for others, full-resolution).

full resolution image with shorter side 256 for evaluation, so that we can give a direct comparison; when we consider the efficiency (*e.g.*, as in Table 2), we used just 1 clip per video and the center 224×224 crop for evaluation. We keep the same protocol for the methods compared in the same table.

5.1.2 Model.

To have an apple-to-apple comparison with the state-of-the-art method [60], we used the same backbone (ResNet-50) on the dataset (Something-Something-V1 [18]).This dataset focuses on temporal modeling. The difference is that [60] used 3D ResNet-50, while we used 2D ResNet-50 as the backbone to demonstrate efficiency.

5.1.3 Datasets.

Kinetics dataset [29] is a large-scale action recognition dataset with 400 classes. As pointed in [63], [68], datasets like Something-Something (V1&V2) [18], Charades [45], and Jester [38] are more focused on modeling the temporal relationships , while UCF101 [47], HMDB51 [30], and Kinetics [29] are less sensitive to temporal relationships. Since TSM focuses on temporal modeling, we mainly focus on datasets with stronger temporal relationships like Something-Something. Nevertheless, we also observed strong results on the other datasets and reported it.

5.2 Improving 2D CNN Baselines

We can seamlessly inject TSM into a normal 2D CNN and improve its performance on video recognition. In this section, we demonstrate a 2D CNN baseline can significantly benefit from TSM with double-digits accuracy improvement. We chose TSN [58] as the 2D CNN baseline. We used the same training and testing protocol for TSN and our TSM. The only difference is with or without TSM.

5.2.1 Comparing Different Datasets.

We compare the results on several action recognition datasets in Table 1. The chart is split into two parts. The upper part contains datasets Kinetics [29], UCF101 [47], HMDB51 [30], where temporal relationships are less important, while our TSM still consistently outperforms the 2D TSN baseline at no extra computation. For the lower part, we present the results on Something-Something V1 and V2 [18] and Jester [38], which depend heavily on temporal relationships. 2D CNN baseline cannot achieve a good accuracy, but once equipped with TSM, the performance improved by double digits.

5.2.2 Scaling over Backbones.

TSM scales well to backbones of different sizes. We show the Kinetics top-1 accuracy with MobileNet-V2 [42], ResNet-50 [22], ResNext-101 [62] and ResNet-50 + Non-local module [59] backbones in Table 3. TSM consistently improves the accuracy over different backbones, even for NL R-50, which already has temporal modeling ability.

5.3 Comparison with State-of-the-Arts

TSM not only significantly improves the 2D baseline but also outperforms state-of-the-art methods, which heavily rely on 3D convolutions. We compared the performance of our offline (bi-directional) TSM model with state-of-the-art methods on both Something-Something V1&V2 because these two datasets focus on temporal modeling.

5.3.1 Something-Something-V1.

Something-Something-V1 is a challenging dataset, as activity cannot be inferred merely from individual frames (*e.g.*, pushing something from *right to left*). We compared TSM with current state-of-the-art methods in Table 2. We only applied center crop during testing to ensure the efficiency unless otherwise specified. TSM achieves *the first place* on the leaderboard upon publication.

We first show the results of the 2D based methods TSN [58] and TRN [68]. TSN with different backbones fails to achieve decent performance (<20% Top-1) due to the lack of temporal modeling. For TRN, although **late temporal fusion** is added after feature extraction, the performance is still significantly lower than state-of-the-art methods, showing the importance of temporal fusion across all levels.

The second section shows the state-of-the-art efficient video understanding framework ECO [72]. ECO uses an early 2D + late 3D architecture which enables **medium-level temporal fusion**. Compared to ECO, our method achieves better performance at a smaller FLOPs. For example, when using 8 frames as input, our TSM achieves 45.6% top-1 accuracy with 33G FLOPs, which is 4.2% higher accuracy than ECO with 1.9× less computation. The ensemble versions of ECO ($\text{ECO}_{En}Lite$ and $\text{ECO}_{En}Lite_{\text{RGB+Flow}}$, using an ensemble of $\{16, 20, 24, 32\}$ frames as input) did achieve competitive results, but the computation and parameters are too large for deployment. While our model is much more efficient: we only used $\{8, 16\}$ frames model for ensemble (TSM_{En}), and the model achieves better performance using 2.7× less computation and 3.1× fewer parameters.

The third section contains previous state-of-the-art methods: Non-local I3D + GCN [60], that enables **all-level temporal fusion**. The GCN needs a Region Proposal Network [40] trained on MSCOCO object detection dataset [35] to generate the bounding boxes, which is unfair to compare since external data (MSCOCO) and extra training cost is introduced. Thus we compared TSM to its CNN part: Non-local I3D. Our TSM (8f) achieves 1.2% better accuracy with 10× fewer FLOPs on the validation set compared to the Non-local I3D network. Note that techniques like Non-local module [59] are orthogonal to our work, which could also be added to our framework to boost the performance further.

1. We reported the performance of NL I3D described in [60], which is a variant of the original NL I3D [59]. It uses fewer temporal dimension pooling to achieve good performance, but also incur larger computation.

2. Includes parameters and FLOPs of the Region Proposal Network.

Model	Backbone	#Frame	FLOPs/Video	#Param.	Val Top-1	Val Top-5	Test Top-1
TSN [68]	BNInception	8	16G	10.7M	19.5	-	-
TSN (our impl.)	ResNet-50	8	33G	24.3M	19.7	46.6	-
TRN-Multiscale [68]	BNInception	8	16G	18.3M	34.4	-	33.6
TRN-Multiscale (our impl.)	ResNet-50	8	33G	31.8M	38.9	68.1	-
Two-stream $TRN_{RGB+Flow}$ [68]	BNInception	8+8	-	36.6M	42.0	-	40.7
ECO [72]	BNIncep+3D Res18	8	32G	47.5M	39.6	-	-
ECO [72]	BNIncep+3D Res18	16	64G	47.5M	41.4	-	-
ECO_{EnLite} [72]	BNIncep+3D Res18	92	267G	150M	46.4	-	42.3
$ECO_{EnLite_{RGB+Flow}}$ [72]	BNIncep+3D Res18	92+92	-	300M	49.5	-	43.9
I3D from [60]	3D ResNet-50	32×2clip	$153G^{1}\times2$	28.0M	41.6	72.2	-
Non-local I3D from [60]	3D ResNet-50	32×2clip	$168G^{1}\times2$	35.3M	44.4	76.0	-
Non-local I3D + GCN [60]	3D ResNet-50+GCN	32×2clip	$303G^{2}\times2$	$62.2M^{2}$	46.1	76.8	45.0
CorrNet-50 [56]	R(2+1)D-50	32×10clip	115G×10	-	49.3	-	-
ip-CSN-152 [53]	3D ResNet-152	32×10clip	83.3G×10	33.0M	53.3	-	-
TSM	ResNet-50	8	33G	24.3M	45.6	74.2	-
TSM	ResNet-50	16	65G	24.3M	47.2	77.1	46.0
TSM_{En}	ResNet-50	24	98G	48.6M	49.7	78.5	-
$TSM_{RGB+Flow}$	ResNet-50	16+16	-	48.6M	**52.6**	**81.9**	**50.7**

TABLE 2. Comparing TSM against other methods on Something-Something dataset (center crop, 1 clip/video unless otherwise specified).

	Mb-V2	R-50	RX-101	NL R-50
TSN	66.5	70.7	72.4	74.6
TSM	69.5	74.1	76.3	75.7
ΔAcc.	+3.0	+3.4	+3.9	+1.1

TABLE 3. TSM can consistently improve the performance over different backbones on Kinetics dataset.

Method	Val		Test	
	Top-1	Top-5	Top-1	Top-5
TSN (our impl.)	30.0	60.5	-	-
MultiScale TRN [68]	48.8	77.6	50.9	79.3
2-Stream TRN [68]	55.5	83.1	56.2	83.2
TSM_{8F}	59.1	85.6	-	-
TSM_{16F}	63.4	88.5	64.3	89.6
$TSM_{RGB+Flow}$	**66.0**	**90.5**	**66.6**	**91.3**

TABLE 4. Results on Something-Something-V2. Our TSM achieves state-of-the-art performance.

We further include two recent state-of-the-art methods that achieve state-of-the-art performance: CorrNet [56] and CSN [53]. For CorrNet, we compare to CorrNet-50 which has a similar backbone shape; For CSN, we compare to ip-CSN-152, which is the largest model and achieves the highest accuracy. Both methods achieve high accuracy on Something-Something dataset. However, they still require sampling 10 clips to get the average prediction. The total computation is larger than 800G FLOPs, which is not practical for edge deployment.

5.3.2 *Generalize to Other Modalities.*

We also show that our proposed method can generalize to other modalities like optical flow. To extract the optical flow information between frames, we followed [58] to use the TVL1 optical flow algorithm [65] implemented in OpenCV with CUDA. We conducted two-stream experiments on both Something-Something V1 and V2 datasets, and it consistently improves over the RGB performance: introducing optical flow branch brings 5.4% and 2.6% top-1 improvement on V1 and V2.

5.3.3 *Something-Something-V2.*

We also show the result on Something-Something-V2 dataset, which is a newer release to its previous version. The results compared to other state-of-the-art methods are shown in Table 4. On Something-Something-V2 dataset, we achieved state-of-the-art performance while only using RGB input.

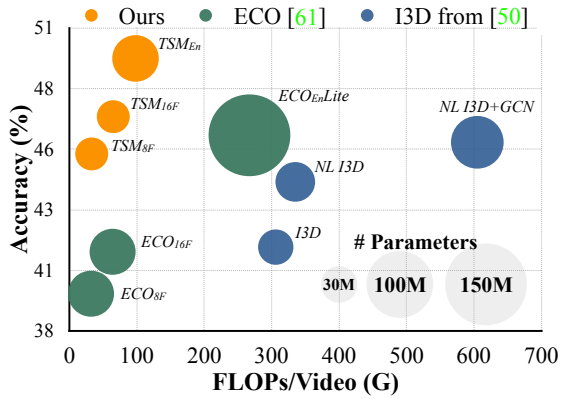


Fig. 5. TSM enjoys better accuracy-cost trade-off than I3D family and ECO family on Something-Something-V1 [18] dataset. (GCN includes the cost of ResNet-50 RPN to generate region proposals.)

5.3.4 *Cost vs. Accuracy.*

Our TSM model achieves very competitive performance while enjoying high efficiency and low computation cost for fast inference. We show the FLOPs for each model in Table 2. Although GCN itself is light, the method used a ResNet-50 based Region Proposal Network [40] to extract bounding boxes, whose cost is also

Model	Latency/Clip	Top-1
I3D NL R-50 [59]	30.7ms	74.9%
SlowFast R-50 [10]	33.3ms	75.6%
TSM NL R-50	**25.6ms**	**75.7%**
X3D-M [9]	73.5ms	76.0%
ir-CSN-152 [53]	138.6ms	**76.8%**
TSM RX-101	**40.7ms**	76.3%

TABLE 5. Compare to state-of-the-art methods on Kinetics. TSM can achieve higher or comparable performance at a lower inference latency.

considered in the chart. Note that the computation cost of optical flow extraction is usually larger than the video recognition model itself. Therefore, we do not report the FLOPs of two-stream based methods.

We show the accuracy, FLOPs, and number of parameters trade-off in Figure 5. The accuracy is tested on the validation set of Something-Something-V1 dataset, and the number of parameters is indicated by the area of the circles. We can see that our TSM based methods have a better Pareto curve than both previous state-of-the-art efficient models (ECO based models) and high-performance models (non-local I3D based models). TSM models are both efficient and accurate. It can achieve state-of-the-art accuracy at high efficiency: it achieves better performance while consuming $3\times$ less computation than the ECO family . Considering that ECO is already an efficiency-oriented design, our method enjoys highly competitive hardware efficiency.

5.3.5 *Kinetics.*

Although Kinetics does not focus on temporal modeling (Table 1), we compare TSM with state-of-the-art methods on Kinetics to give a comprehensive comparison. The results are show in Table 5.

We compare to several state-of-the-art methods: Non-local networks [59] (I3D NL R-50), SlowFast [10] (SlowFast R-50), X3D [9] (X3D-M), and CSN [53] (ir-CSN-152). We report the latency and accuracy trade-off of different methods. The latency is measured on NVIDIA RTX 2080 Ti GPU using batch size 1. We first warm-up the inference for 100 iterations and measure the average latency of the next 200 iterations. TSM can achieve higher or comparable performance at a lower inference latency. TSM (TSM NL R-50) achieves same-level of accuracy compared to SlowFast network (SlowFast R-50) at $1.3\times$ lower latency. TSM (TSM RX-101) also outperforms X3D (X3D-M) at $1.8\times$ lower latency. Notice that though X3D has a small computation FLOPs, its inferior hardware efficiency leads to the slow inference speed. ir-CSN-152 achieves slightly higher accuracy than TSM, but TSM runs $3.4\times$ faster. TSM is very competitive for accuracy-speed trade-off.

5.4 Latency and Throughput Speedup

The measured inference latency and throughput are important for the large-scale video understanding. TSM has low latency and high throughput. We performed measurement on a single NVIDIA Tesla P100 GPU. We used batch size of 1 for latency measurement; batch size of16 for throughput measurement. We made two comparisons:

(1) Compared with the I3D model from [60], our method is faster by an order of magnitude at 1.8% higher accuracy (Table 6). We also compared our method to the state-of-the-art efficient model ECO [72]: Our TSM model has $1.75\times$ lower latency (17.4ms *vs.* 30.6ms), $1.7\times$ higher throughput, and achieves 2% better accuracy.

Model	Efficiency Statistics				Accuracy	
	FLOPs	Param.	Latency	Thrput.	Sth.	Kinetics
I3D from [60]	306G	35.3M	165.3ms	6.1V/s	41.6%	-
ECO_{16F} [72]	64G	47.5M	30.6ms	45.6V/s	41.4%	-
I3D from [59]	**33G**	29.3M	25.8ms	42.4V/s	-	73.3%
$I3D_{replace}$	48G	33.0M	28.0ms	37.9V/s	44.9%	-
TSM_{8F}	**33G**	**24.3M**	**17.4ms**	**77.4V/s**	45.6%	74.1%
TSM_{16F}	65G	24.3M	29.0ms	39.5V/s	**47.2%**	**74.7%**

TABLE 6. TSM enjoys low GPU inference latency and high throughput. V/s means videos per second, higher the better (Measured on NVIDIA Tesla P100 GPU).

Model	Latency	Kinetics	UCF101	HMDB51	Something
TSN	4.7ms	70.6%	91.7%	64.7%	20.5%
+Offline	-	74.1%	95.9%	73.5%	47.3%
+Online	4.8ms	74.3%	95.5%	73.6%	46.3%

TABLE 7. Comparing the accuracy of offline TSM and online TSM on different datasets. Online TSM brings negligible latency overhead.

ECO has a two-branch (2D+3D) architecture, while TSM only needs the in-expensive 2D backbone.

(2) We then compared TSM to efficient 3D model designs. One way is to only inflate the first 1×1 convolution in each of the block as in [59], denoted as "I3D from [59]" in the table. Although the FLOPs are similiar due to pooling, it suffers from $1.5\times$ higher latency and only 55% the throughput compared with TSM, with worse accuracy. We speculate the reason is that TSM model only uses 2D convolution which is highly optimized for hardware. To exclude the factors of backbone design, we replace every TSM primitive with $3 \times 1 \times 1$ convolution and denote this model as $I3D_{replace}$. It is still much slower than TSM and performs worse.

5.5 Online Recognition with TSM

5.5.1 *Online* vs. *Offline*

Online TSM models shift the feature maps uni-directionally so that it can give predictions in real time. We compare the performance of offline and online TSM models to show that online TSM can still achieve comparable performance. Follow [72], we use the prediction averaged from all the frames to compare with offline models, *i.e.*, we compare the performance after observing the whole videos. The performance is provided in Table 7. We can see that for less temporal related datasets like Kinetics, UCF101 and HMDB51, the online models achieve comparable and sometimes even better performance compared to the offline models. While for more temporal related datasets Something-Something, online model performs worse than offline model by 1.0%. Nevertheless, the performance of online model is still significantly better than the 2D baseline.

We also compare the per-frame prediction latency of pure 2D backbone (TSN) and our online TSM model. We compile both models with TVM [5] on GPU. Our online TSM model only adds to less than 0.1ms latency overhead per frame while bringing up to 25% accuracy improvement. It demonstrates online TSM is hardware-efficient for latency-critical real-time applications.

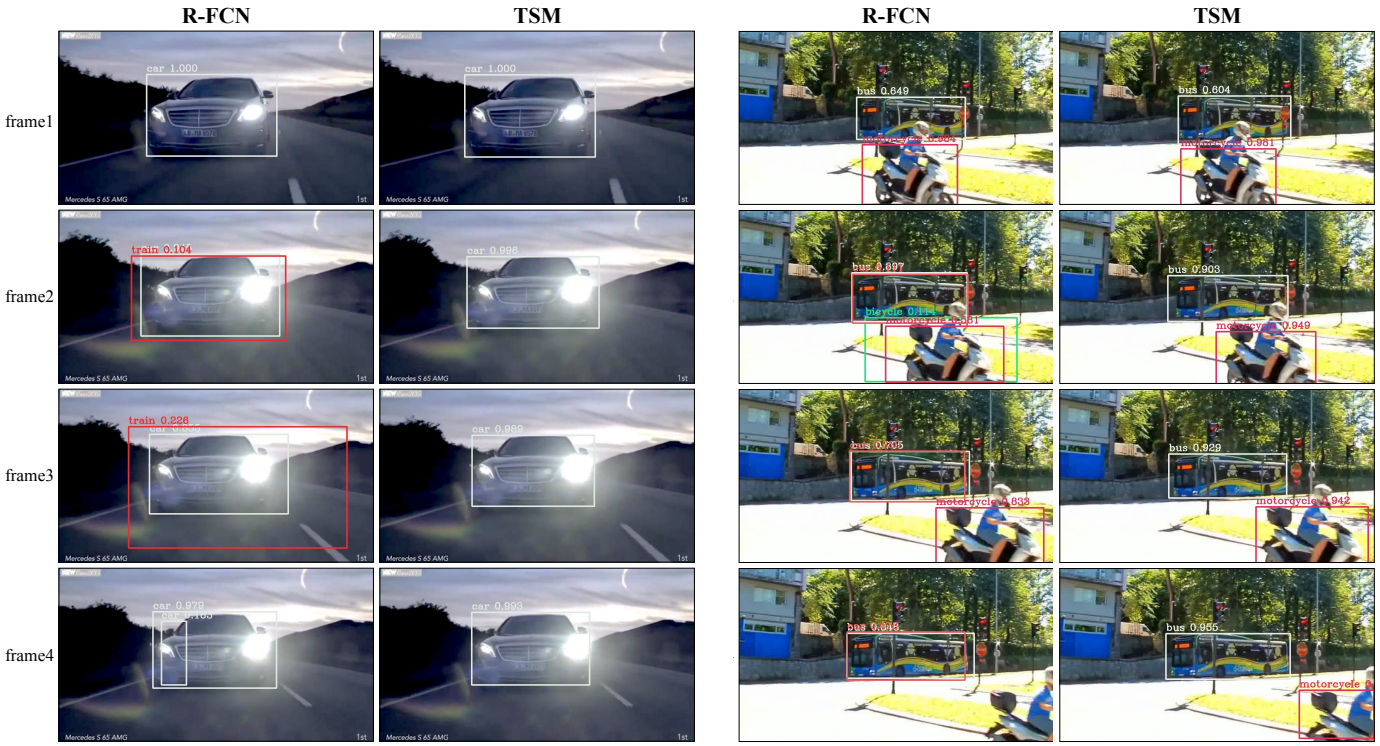


Fig. 6. TSM improves detection results with the help of temporal cues. For the left video, 2D baseline R-FCN generates false positive due to the glare of car headlight on frame 2/3/4, while TSM does not have such issue by considering the temporal information. For the right video, R-FCN generates false positive surrounding the bus due to occlusion by the traffic sign on frame 2/3/4. Also, it fails to detect motorcycle on frame 4 due to occlusion. TSM model addresses such issues with the help of temporal information.

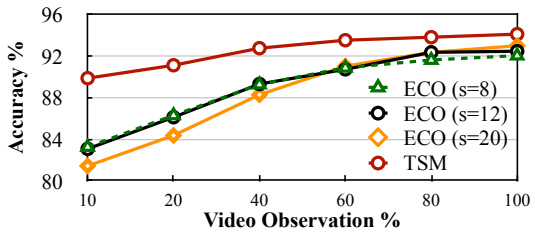
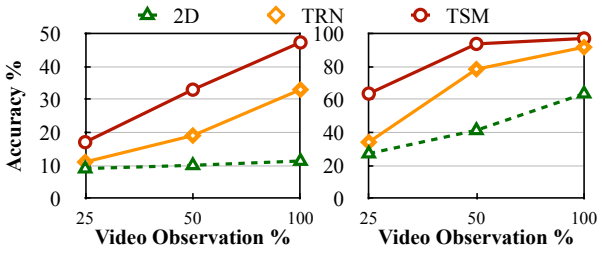


Fig. 7. Early recognition on UCF101. TSM gives high prediction accuracy after only observing a small portion of the video.



(a) Something-Something (b) Jester

Fig. 8. Early recognition on Something-Something and Jester datasets. TSM consistently outperforms TRN [68] at various portions by a large margin.

5.5.2 Early Recognition

Early recognition aims to classify the video while only observing a small portion of the frames. It gives fast response to the input video stream. Here we compare the early video recognition performance with ECO [72] on UCF101 (Figure 7) and TRN [68] on Something-Something and Jester (Figure 8). Compared to ECO, TSM gives much higher accuracy, especially when only observing a small portion of the frames. For example, when only observing the first 10% of video frames, TSM model can achieve 90% accuracy, which is 6.6% higher than the best ECO model. TSM also consistently outperforms TRN at various observation percentages by a large margin.

Devices	Jetson Nano		Jetson TX2		Rasp.	Note8	Pixel1
	CPU	GPU	CPU	GPU			
FPS	20.9	74.6	27.5	117.6	14.4	29.0	21.1
Power (watt)	4.8	4.5	5.6	5.8	3.8	-	-

TABLE 8. TSM efficiently runs on edge devices with low latency.

5.5.3 Edge Deployment

TSM is mobile device friendly. We build an online TSM model with MobileNet-V2 backbone, which achieves 69.5% accuracy on Kinetics. The latency and energy on NVIDIA Jetson Nano & TX2, Raspberry Pi 4B, Samsung Galaxy Note8, Google Pixel-1 is shown in Table 8. The models are compiled using TVM [5]. Power is measured with a power meter, subtracting the static power. TSM achieves low latency and low power on edge devices.

6 Online Object Detection

Real-time online video object detection is an important application in self-driving vehicles, robotics, *etc*. Most exiting methods treat video detection as image detection per frame, which is not robust since temporal information is not considered. Other methods on video object detection [71] fuses information along temporal

Model	Online	Need Flow	Latency	mAP			
				Overall	Slow	Medium	Fast
R-FCN [7]	✓		1×	74.7	83.6	72.5	51.4
FGFA [71]		✓	2.5×	75.9	**84.0**	74.4	55.6
Online TSM	✓		1×	**76.3**	83.4	**74.8**	**56.0**

TABLE 9. Video detection results on ImageNet-VID.

dimension after the 2D feature is extracted by the backbone, which results in high latency, and also loses low-level temporal cues.

Here we show that we can enable temporal fusion in online video object detection by injecting our uni-directional TSM into the backbone. We show that we can significantly improve the performance of video detection by simply modifying the backbone with online TSM, without changing the detection module design or using optical flow features.

We conducted experiments with R-FCN [7] detector on ImageNet-VID [41] dataset. Following the setting in [71], we used ResNet-101 [22] as the backbone for R-FCN detector. For TSM experiments, we inserted uni-directional TSM to the backbone, while keeping other settings the same. We used the official training code of [71] to conduct the experiments, and the results are shown in Table 9. Compared to 2D baseline R-FCN [7], our online TSM model significantly improves the performance, especially on the fast moving objects, where TSM increases mAP by 4.6%. FGFA [71] is a strong baseline that uses optical flow to aggregate the temporal information from 21 frames (past 10 frames and future 10 frames) for offline video detection. Compared to FGFA, TSM can achieve similar or higher performance while enabling online recognition (using information from only past frames) at much smaller latency per frame. The latency overhead of TSM module itself is less than 1ms per frame, making it a practical tool for real deployment.

We visualize the detection results of two video clips in Figure 6. In the left video clip, 2D baseline R-FCN generates false positive due to the glare of car headlight on frame 2/3/4, while TSM suppresses false positive. In the right video clip, R-FCN generates false positive surrounding the bus due to occlusion by the traffic sign on frame 2/3/4. Also, it fails to detect motorcycle on frame 4 due to occlusion. TSM model addresses such issues with the help of temporal information.

7 Scalability in Distributed Training

In this section, we study how the design of TSM helps to improve the scalability in distributed training of video models.

7.1 Factor of Video Network Design

To study the distributed training scalability, we first discuss the factors that might affect the scalability of video network training [33].

7.1.1 Temporal modeling unit.

3D convolution is the most widely used operator for spatial-temporal modeling. However, it suffers from two problems: (1) large computation and large parameter size, which slows down training and communication; (2) low hardware efficiency compared to 2D convolution. Give the same amount of FLOPs, 3D kernels run 1.2 to 3 times slower than 2D on cuDNN [6]. On the other hand, our TSM module is a highly efficient alternative.

7.1.2 Backbone topology.

Existing video networks usually sample **many** frames as input (32 frames [59] or 64 frames [4]), and perform temporal pooling later to progressively *reduce* the temporal resolution (Figure 10a). Another way is to sample **fewer** frames (e.g. 8 frames [58]) as input while keeping the *same* temporal resolution to keep the information (Figure 10b). Although the overall computation of the two designs are similar, the former significantly increases the data loading traffic, making the system I/O heavy, which could be quite challenging in a distributed system considering the limited disk bandwidth.

7.2 Design Guidelines to Video Model Architecture

To tackle the challenge in a distributed training systems, we propose three video model design guidelines: (1) To increase the *computation efficiency*, use operators with lower FLOPs and higher hardware efficiency; (2) To reduce *data loading traffic*, use a network topology with higher FLOPs/data ratio; (3) To reduce the *networking traffic*, use operators with fewer parameters.

We show the advantage of the above three design guidelines by experimenting on three models in Table 10. All the models use the ResNet-50 backbone to exclude the influence of spatial modeling. The model architectures are introduced as follows.

(1) The first model is an I3D model from [21]. The model takes 16 frames as input and inflate all the 3×3 convolutions to $3 \times 3 \times 3$. It performs temporal dimension pooling by four times to reduce the temporal resolution. We denote the model as $\text{I3D}_{3\times3\times3}$.

(2) The second model is an I3D model from [59], taking 32 frames as input and inflating the first 1×1 convolution in every other ResBlock. It applies temporal dimension pooling by three times. We denote this more computation and parameter efficient design as $\text{I3D}_{3\times1\times1}$.

(3) The third model is built with TSM. The TSM operator is inserted into every ResBlock. The model takes 8 frames as input and performs no temporal pooling. We denote this model as TSM.

7.2.1 Computation Efficiency.

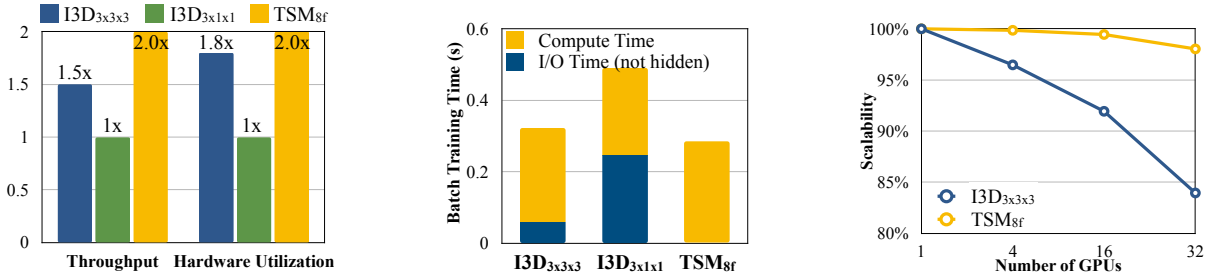
Computation efficiency is the most direct factor that influence the training time. As shown in Table 10, $\text{TSM}_{8\text{f}}$ has $1.2\times$ fewer FLOPs compared to $\text{I3D}_{3\times3\times3}$ and roughly the same FLOPs compared to $\text{I3D}_{3\times1\times1}$. However, the actual inference throughput also depends on the hardware utilization. We measure the inference throughput (defined as videos per second) of the three models on a single NVIDIA Tesla P100 GPU using batch size 16. We also measured the hardware utilization, defined as achieved FLOPs/second over peak FLOPs/second. The inference throughput and the hardware efficiency comparison is shown in Figure 9a. We can find that the *model is more hardware efficient if it has more 2D convolutions than 3D*: TSM is a fully 2D CNN, therefore it has the best hardware utilization ($2.0\times$); while the last several stage of $\text{I3D}_{3\times3\times3}$ (res_4, res_5) have few temporal resolution (as shown in Table 11), it is more similar to 2D convolution and thus is $1.8\times$ more hardware efficient than $\text{I3D}_{3\times1\times1}$ ($1.0\times$).

7.2.2 Data Loading Efficiency.

Video datasets are usually very large. For a distributed system like the Summit supercomputer, the data is usually stored in High Performance Storage System (HPSS) shared across all the worker nodes. Such file systems usually have great sequential I/O performance but inferior random access performance. Therefore, large data traffic could easily become the system bottleneck.

	Acc.↑	FLOPs↓	#Param.↓	Input size↓	Throughput↑	Compute/IO↑
$I3D_{3\times3\times3}$ [21]	68.0%	40G	47.0M	$16\times3\times224^2$	63.1V/s (1.5×)	16.6k (2.4×)
$I3D_{3\times1\times1}$ [59]	73.3%	33G	29.3M	$32\times3\times224^2$	41.9V/s (1.0×)	6.85k (1×)
TSM	**74.1%**	33G	24.3M	$8\times3\times224^2$	84.8V/s (**2.0**×)	27.4k (**4**×)

TABLE 10. Efficiency statistics of different models. Arrows show the better direction.



(a) TSM has fewer FLOPs, better throughput and utilization.

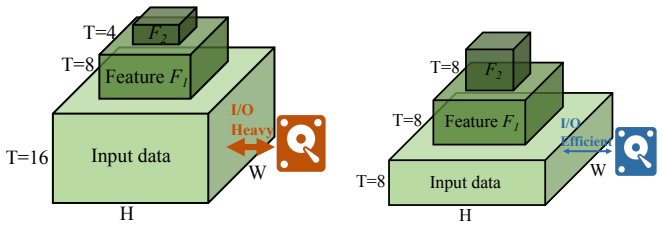
(b) TSM is I/O light, decreasing the total batch time.

(c) TSM has better scalability due to smaller model size.

Fig. 9. Analyzing how different design aspects influence the distributed training scalability of video recognition models: **(a) computation efficiency; (b) data loading efficiency; (c) networking efficiency**.

Block	data	$conv_1$	$pool_1$	res_2	res_3	res_3	res_4	res_5	global pool
$I3D_{3\times3\times3}$	16	16	8	8	-	4	2	1	1
$I3D_{3\times1\times1}$	32	16	8	8	4	4	4	4	1
TSM_{8f}	8	8	8	8	-	8	8	8	1

TABLE 11. The temporal resolution of output feature map for each block. TSM is a fully 2D structure, enjoying the best hardware efficiency. The last several stages of $I3D_{3\times3\times3}$ have fewer temporal resolution, making it more similar to 2D CNN, thus enjoying better hardware efficiency compared to $I3D_{3\times1\times1}$.



(a) Pooled-up

(b) Straight-up (ours)

Fig. 10. Two kinds of video backbone design. Straight-up backbone does not perform temporal pooling and is more data efficient. Pooled-up version requires many input frames and drains I/O.

Previous popular I3D models [4], [21] takes many frames per video (16 or 32) as input and perform down-sample over temporal dimension. We argue that such design is a waste of disk bandwidth: a TSM_{8f} only takes 8 frames as input while achieving better accuracy. The intuition is that nearby frames are similar; loading too many similar frames is redundant. We empirically test the data loading bottleneck on Summit. To exclude the communication cost from the experiments, we perform timing on single-node training. We measure the total time of one-batch training and the time for data loading (that is not hidden by the computation). As shown in Figure 9b, for $I3D_{3\times1\times1}$, it takes 32 frames as input. The data loading time cannot be hidden by the computation, therefore data I/O becomes the bottleneck. $I3D_{3\times3\times3}$ that takes 16 frame as input has less problem on data loading, while TSM_{8f} can fully hide the data loading time with computation. We also compute the model FLOPs divided by the input data size as a measurement of data efficiency. The value is denoted as "Compute/IO" as in Table 10. For scalable video recognition models, we want a model with larger Compute/IO ratio.

7.2.3 *Networking Efficiency.*

In distributed training system, the communication time can be modelled as:

$$\text{communication time} = \text{latency} + \frac{\text{model size}}{\text{bandwidth}} \tag{3}$$

The latency and bandwidth is determined by the network condition, which cannot be optimized through model design. However, we can reduce the model size to reduce the communication cost. Both I3D models inflate some of the 2D convolution kernels to 3D, which will increase the number of parameters by k_T. While TSM module does not introduce extra parameters. Therefore, it has the same model size as the 2D counterpart. For example, $I3D_{3\times3\times3}$ has 1.9× larger model size than TSM_{8f}, which introduces almost two times of network communication during distributed training. To test the influence of model size on scalability, we measure the scalability on a 8 node cluster. Each computer has 4 NVIDIA TESLA P100 GPUs. We define the scalability as the actual training speed divided by the ideal training speed (single machine training speed * number of nodes). The results are shown in Figure 9c. Even with the high-speed connection, the scalability of $I3D_{3\times3\times3}$ quickly drops as the number of training nodes increase: the scalability is smaller than 85% when applied to 8 nodes. While TSM_{8f} model still has over 98% of scalability thanks to the smaller model size thus smaller networking traffic.

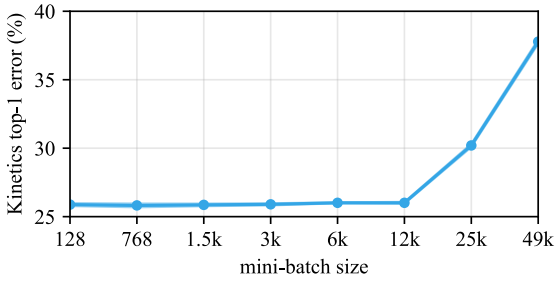


Fig. 11. Kinetics top-1 validation accuracy *vs*. mini-batch size. The performance of the model does not degrade when we scale up the mini-batch size to $12k$. The mean and standard deviation (the scale of the STD is hardly visible) are shown in the figure.

#Node	#GPU	Batch	#Frames	Accuracy	Time	*Note*
1	6	96	768	74.12±0.11	49h 55m	*Baseline*
8	48	384	3,072	74.12±0.08	7h 7m	*same level of accuracy*
16	96	768	6,144	74.18±0.14	3h 38m	
32	192	1,536	12,288	74.14±0.10	1h 50m	
64	384	3,072	24,576	74.10±0.08	55m 56s	
128	768	6,144	49,152	73.99±0.04	28m 14s	
256	1536	12,288	98,304	73.99±0.07	14m 13s	
384	2304	18,432	147,456	72.52±0.07	10m 9s	*lose accuracy*
512*	3072	24,576	196,608	69.80±0.13	-	
1024*	6144	49,152	393,216	62.22±0.17	-	

TABLE 12. Detailed statistics of different mini-batch size (* indicates simulated performance).

7.3 Large-scale Distributed Training on Summit

We scale up the training of video recognition model on Summit supercomputer. With the help of above hardware-aware model design techniques, we can scale up the training to 1536 GPUs, finishing the training of Kinetics in 15 minutes.

7.3.1 Setups

Summit [55] or OLCF-4 is a supercomputer at Oak Ridge National Laboratory, which as of September 2019 is the fastest supercomputer in the world. It consists of approximately 4,600 compute nodes, each with two IBM POWER9 processors and six NVIDIA Volta V100 accelerators. The POWER9 processor is connected via dual NVLINK bricks, each capable of a 25GB/s transfer rate in each direction. Nodes contain 512 GB of DDR4 memory for use by the POWER9 processors and 96 GB of High Bandwidth Memory (HBM2) for use by the accelerators [3].

We used PyTorch and Horovod [43] for distributed training. The framework uses ring-allreduce algorithm to perform synchronized SGD. The training is accelerated by CUDA and cuDNN. We used NVIDIA Collective Communication Library (NCCL) [4] for most of the communication.

For training on Kinetics, we used the same hyper-parameter at the same batch size, applying a linear scaling rule [17].

7.3.2 Experiments

Baseline. For the baseline, we trained a ResNet-50 TSM_{8f} model on a single Summit node with 6 GPUs, each GPU contains 16 video clips, resulting in a total batch size of $kn = 96$. We evaluate the performance of last 5 checkpoints, it achieves a top-1 accuracy of $74.12 \pm 0.11\%$.

3. https://www.olcf.ornl.gov/for-users/system-user-guides/summit/summit-user-guide

4. https://developer.nvidia.com/nccl

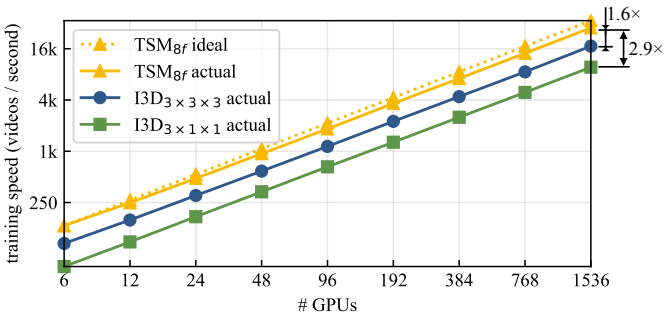


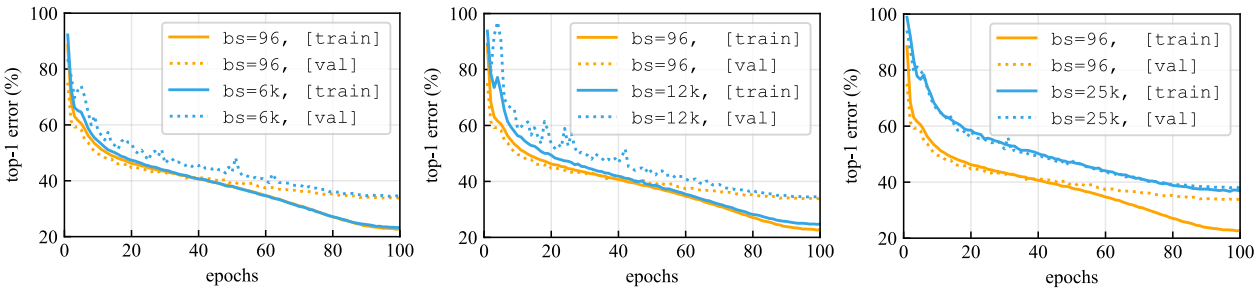
Fig. 12. The training speed and scalability of distributed synchronous SGD training. TSM_{8f} achieves a good scalability (>80%) even when using 1536 GPUs. TSM_{8f} can achieve 1.6× higher training speed compared to $\text{I3D}_{3\times3\times3}$ and 2.9× compared to $\text{I3D}_{3\times1\times1}$, showing the effectiveness of the proposed design guidelines.

Performance *vs*. Batch Size. We first compare the training error *vs*. the batch size. As shown in [17], the accuracy will not degrade when the batch size is relatively small. Therefore, our experiments start from 8 computing nodes (48 GPUs, 384 video clips, 3072 frames) to 1024 computing nodes (6144 GPUs, 49152 video clips, 393216 frames) . Note that each sample in a video recognition model is a video clip consisting of several frames/images (in our case, 8). Therefore, the actual number of images used in one batch could be much larger than ImageNet training (*e.g.*, $98k$ *vs*. $8k$ [17]).

We first plot the error *vs*. batch size trade-off in Figure 11. The error does not increase when we scale the number of computing nodes up to 256 (1536 GPU), where the batch size is 12288, the total frame number is 98304. The detailed statistics are shown in Table 12. The scalability of TSM model is very close to the ideal case. Note that due to quota limitation, the largest physical nodes we used is 384 with 2304 GPUs. For 512 and 1024 nodes, we used gradient accumulation to simulate the training process (denoted by *).

We also provide the training and testing convergence curves using 768, 1536 and 3072 GPUs in Figure 13. For 768 GPUs and 1536 GPUs, although the convergence of large-batch distributed training is slower than single-machine training baseline, the final converged accuracy is similar, so that the model does not lose accuracy. For 3072 GPUs, the accuracy degrades for both training and testing.

Scalabilty. We test the scalability of distributed training on Summit. According to the results from last section, we can keep the accuracy all the way to 256 computing nodes. Therefore, we sweep the number of computing nodes from 1 to 256 to measure the scalability. We keep a batch size of 8 for each GPU and each node has 6 GPUs. So the batch sizes change from 48 to 18,432. Each video clips contains 8 frames in our model, resulting a total number of frames from 384 to 147,456. We measure the training speed (videos/second) to get the actual speed-up. We calculate the ideal training speed using the single node training speed multiplied by number of nodes. The comparison of different models is provided in Figure 12. The actual training speed is just marginally below the ideal scaling, achieving $> 80\%$ scaling efficiency. We also provide the detailed overall training time in Table 12. With 1536 GPUs, we can finish the Kinetics training with TSM within 14 minutes and 13 seconds, achieving a top-1 accuracy of 74.0%. The overall training speed of TSM_{8f} is 1.6× larger than $\text{I3D}_{3\times3\times3}$ and 2.9×



(a) Mini-batch size $6k$. **(b)** Mini-batch size $12k$. **(c)** Mini-batch size $25k$ (degrade).

Fig. 13. The learning curve for baseline training and large-batch distributed training (batch size 6144, 12228, 24576). The performance does not degrade for batch size 6144 and 12228, while degrades for a batch size of 24576.

Fig. 14. Spatial-temporal action detector emerges in TSM video classification models, while single-frame baseline (TSN) cannot localize the action. *Italic* title indicates the action category. In the first example, our TSM model precisely localize the "hitting" action, while TSN can only highlight the object. In the second example, TSM localizes the "plugging" action but not the other hand motion. Finally, TSM accurately locates the temporal region where the "removing" action happens.

larger than $\text{I3D}_{3\times1\times1}$, showing the advantage of hardware-aware model design.

8 Video Network Dissection

In this section, we dissect the trained TSM model to understand how it learns temporal information compared to 2D networks.

To investigate what the action recognition network is learning, we adopt a similar method as in [69] to get the Class Activation Mapping (CAM), which shows the salience of each class over the input image. Take ResNet backbone as an example, the original output for the video network is:

$$\text{logit} = \text{fc}(\text{pool}(x_{conv})) \tag{4}$$

where x_{conv} is the output activation of the last convolutional layer (*e.g.*, has shape $1 \times 2048 \times 8 \times 7 \times 7$), pool is the global average pooling over both spatial and temporal dimension (reduce the shape to 1×2048), and fc is the last fully connected layer for classification. To get CAM, we remove the global average pooling layer, and change the fc layer to a $1 \times 1 \times 1$ convolution using the same weights, which results in a output tensor of shape $1 \times \#\text{class} \times 8 \times 7 \times 7$. We use the CAM map of the predicted

category (highest probability) as the attention of the network.

For visualization, we used a similar method as in [1]. Specifically, we first resize the spatial resolution of CAM feature map to the size of the input video clip ($1 \times \#\text{class} \times 8 \times 224 \times 224$) with bilinear interpolation and use a threshold to divide the attention foreground and background. We set the threshold to preserve 20% of the pixels over the validation set.

We perform experiments on Something-Something V2 [18] dataset. And some results are shown in Figure 14. We compare the attention distribution between our TSM model and 2D baseline TSN. The background of the category-aware attention map is darkened. We find that spatial-temporal action detector emerges in TSM video network, even though we only provide classification label during the training. TSM models can accurately localize the "action", instead of the "object". For example, in the first video clip labeled as "Hitting something with something", TSM model only highlights the region where a pen is hitting the card box, *i.e.*, when and where the action is happening. However, for the 2D baseline, since it does not have the temporal information, it only highlights the object box. The same situation happens for the following two video clips. Note that in the third clip ("Removing something, revealing something behind"), the 5-th frame and the 6-th frame look exactly the same, while with the help of the temporal modeling, TSM model can tell that the 5-th frame is part of the action while the 6-th frame not.

9 Conclusion

We propose Temporal Shift Module for hardware-efficient video recognition. It can be inserted into 2D CNN backbone to enable joint spatial-temporal modeling at no additional cost. The module shifts part of the channels along temporal dimension to exchange information with neighboring frames. Our framework is both efficient and accurate, enabling low-latency video recognition on edge devices. It has better scalability than 3D networks, enabling large-scale training on video recognition. We also show that spatial-temporal action detector emerges in TSM network.

References

[1] David Bau, Bolei Zhou, Aditya Khosla, Aude Oliva, and Antonio Torralba. Network dissection: Quantifying interpretability of deep visual representations. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6541–6549, 2017. 13

[2] Hakan Bilen, Basura Fernando, Efstratios Gavves, Andrea Vedaldi, and Stephen Gould. Dynamic image networks for action recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3034–3042, 2016. 2

[3] Han Cai, Ligeng Zhu, and Song Han. ProxylessNAS: Direct neural architecture search on target task and hardware. In *International Conference on Learning Representations*, 2019. 2

[4] Joao Carreira and Andrew Zisserman. Quo vadis, action recognition? a new model and the kinetics dataset. In *Computer Vision and Pattern Recognition (CVPR), 2017 IEEE Conference on*, pages 4724–4733. IEEE, 2017. 1, 2, 9, 10

[5] Tianqi Chen, Thierry Moreau, Ziheng Jiang, Lianmin Zheng, Eddie Yan, Haichen Shen, Meghan Cowan, Leyuan Wang, Yuwei Hu, Luis Ceze, et al. {TVM}: An automated end-to-end optimizing compiler for deep learning. In *13th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 18)*, pages 578–594, 2018. 4, 7, 8

[6] Sharan Chetlur, Cliff Woolley, Philippe Vandermersch, Jonathan Cohen, John Tran, Bryan Catanzaro, and Evan Shelhamer. cudnn: Efficient primitives for deep learning. *arXiv preprint arXiv:1410.0759*, 2014. 4, 9

[7] Jifeng Dai, Yi Li, Kaiming He, and Jian Sun. R-FCN: Object detection via region-based fully convolutional networks. 2016. 9

[8] Jeffrey Donahue, Lisa Anne Hendricks, Sergio Guadarrama, Marcus Rohrbach, Subhashini Venugopalan, Kate Saenko, and Trevor Darrell. Long-term recurrent convolutional networks for visual recognition and description. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2625–2634, 2015. 1, 2

[9] Christoph Feichtenhofer. X3d: Expanding architectures for efficient video recognition. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 203–213, 2020. 7

[10] Christoph Feichtenhofer, Haoqi Fan, Jitendra Malik, and Kaiming He. Slowfast networks for video recognition. In *Proceedings of the IEEE international conference on computer vision*, pages 6202–6211, 2019. 7

[11] Christoph Feichtenhofer, Axel Pinz, and Richard Wildes. Spatiotemporal residual networks for video action recognition. In *Advances in neural information processing systems*, pages 3468–3476, 2016. 2

[12] Christoph Feichtenhofer, Axel Pinz, and Andrew Zisserman. Convolutional two-stream network fusion for video action recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1933–1941, 2016. 1, 2

[13] Chuang Gan, Chen Sun, Lixin Duan, and Boqing Gong. Webly-supervised video recognition by mutually voting for relevant web images and web video frames. In *European Conference on Computer Vision*, pages 849–866. Springer, 2016. 2

[14] Chuang Gan, Naiyan Wang, Yi Yang, Dit-Yan Yeung, and Alex G Hauptmann. Devnet: A deep event network for multimedia event detection and evidence recounting. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2568–2577, 2015. 2

[15] Chuang Gan, Ting Yao, Kuiyuan Yang, Yi Yang, and Tao Mei. You lead, we exceed: Labor-free video concept learning by jointly exploiting web videos and images. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 923–932, 2016. 2

[16] Rohit Girdhar, Deva Ramanan, Abhinav Gupta, Josef Sivic, and Bryan Russell. Actionvlad: Learning spatio-temporal aggregation for action classification. In *CVPR*, volume 2, page 3, 2017. 1, 2

[17] Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, large minibatch sgd: Training imagenet in 1 hour. *arXiv preprint arXiv:1706.02677*, 2017. 11

[18] Raghav Goyal, Samira Ebrahimi Kahou, Vincent Michalski, Joanna Materzynska, Susanne Westphal, Heuna Kim, Valentin Haenel, Ingo Fruend, Peter Yianilos, Moritz Mueller-Freitag, et al. The" something something" video database for learning and evaluating visual common sense. In *The IEEE International Conference on Computer Vision (ICCV)*, volume 1, page 3, 2017. 5, 6, 13

[19] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *International Conference on Learning Representations*, 2016. 2

[20] Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. In *Advances in neural information processing systems*, pages 1135–1143, 2015. 2

[21] Kensho Hara, Hirokatsu Kataoka, and Yutaka Satoh. Can spatiotemporal 3d cnns retrace the history of 2d cnns and imagenet? In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 6546–6555, 2018. 9, 10

[22] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016. 4, 5, 9

[23] Yihui He, Ji Lin, Zhijian Liu, Hanrui Wang, Li-Jia Li, and Song Han. Amc: Automl for model compression and acceleration on mobile devices. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 784–800, 2018. 2

[24] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997. 2

[25] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017. 2

[26] Forrest N Iandola, Song Han, Matthew W Moskewicz, Khalid Ashraf, William J Dally, and Kurt Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and¡ 0.5 mb model size. *arXiv preprint arXiv:1602.07360*, 2016. 2

[27] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015. 4

[28] Andrej Karpathy, George Toderici, Sanketh Shetty, Thomas Leung, Rahul Sukthankar, and Li Fei-Fei. Large-scale video classification with convolutional neural networks. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 1725–1732, 2014. 1, 2

[29] 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. 3, 4, 5

[30] Hildegard Kuehne, Hueihan Jhuang, Estíbaliz Garrote, Tomaso Poggio, and Thomas Serre. Hmdb: a large video database for human motion recognition. In *Computer Vision (ICCV), 2011 IEEE International Conference on*, pages 2556–2563. IEEE, 2011. 4, 5

[31] Myunggi Lee, Seungeui Lee, Sungjoon Son, Gyutae Park, and Nojun Kwak. Motion feature network: Fixed motion filter for action recognition. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 387–403, 2018. 2

[32] Zhenyang Li, Kirill Gavrilyuk, Efstratios Gavves, Mihir Jain, and Cees GM Snoek. Videolstm convolves, attends and flows for action recognition. *Computer Vision and Image Understanding*, 166:41–50, 2018. 2

[33] Ji Lin, Chuang Gan, and Song Han. Training kinetics in 15 minutes: Large-scale distributed training on videos. *arXiv preprint arXiv:1910.00932*, 2019. 9

[34] Ji Lin, Yongming Rao, Jiwen Lu, and Jie Zhou. Runtime neural pruning. In *Advances in Neural Information Processing Systems*, pages 2181–2191, 2017. 2

[35] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014. 5

[36] Chenxi Liu, Barret Zoph, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan Yuille, Jonathan Huang, and Kevin Murphy. Progressive neural architecture search. *arXiv preprint arXiv:1712.00559*, 2017. 2

[37] Xiang Long, Chuang Gan, Gerard de Melo, Jiajun Wu, Xiao Liu, and Shilei Wen. Attention clusters: Purely attention based local feature integration for video classification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7834–7843, 2018. 2

[38] Joanna Materzynska, Guillaume Berger, Ingo Bax, and Roland Memisevic. The jester dataset: A large-scale video dataset of human gestures. In *The IEEE International Conference on Computer Vision (ICCV) Workshops*, Oct 2019. 5

[39] Zhaofan Qiu, Ting Yao, and Tao Mei. Learning spatio-temporal representation with pseudo-3d residual networks. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 5534–5542. IEEE, 2017. 2

[40] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015. 5, 6

[41] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015. 9

[42] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4510–4520, 2018. 2, 5

[43] Alexander Sergeev and Mike Del Balso. Horovod: fast and easy distributed deep learning in tensorflow. *arXiv preprint arXiv:1802.05799*, 2018. 11

[44] Shikhar Sharma, Ryan Kiros, and Ruslan Salakhutdinov. Action recognition using visual attention. *arXiv preprint arXiv:1511.04119*, 2015. 2

[45] Gunnar A Sigurdsson, Gül Varol, Xiaolong Wang, Ali Farhadi, Ivan Laptev, and Abhinav Gupta. Hollywood in homes: Crowdsourcing data collection for activity understanding. In *European Conference on Computer Vision*, pages 510–526. Springer, 2016. 5

[46] Karen Simonyan and Andrew Zisserman. Two-stream convolutional networks for action recognition in videos. In *Advances in neural information processing systems*, pages 568–576, 2014. 1, 2

[47] Khurram Soomro, Amir Roshan Zamir, and Mubarak Shah. Ucf101: A dataset of 101 human actions classes from videos in the wild. *arXiv preprint arXiv:1212.0402*, 2012. 4, 5

[48] Nitish Srivastava, Elman Mansimov, and Ruslan Salakhudinov. Unsupervised learning of video representations using lstms. In *International conference on machine learning*, pages 843–852, 2015. 2

[49] Lin Sun, Kui Jia, Dit-Yan Yeung, and Bertram E Shi. Human action recognition using factorized spatio-temporal convolutional networks. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 4597–4605, 2015. 2

[50] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015. 2

[51] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, and Quoc V Le. Mnasnet: Platform-aware neural architecture search for mobile. *arXiv preprint arXiv:1807.11626*, 2018. 2

[52] Du Tran, Lubomir Bourdev, Rob Fergus, Lorenzo Torresani, and Manohar Paluri. Learning spatiotemporal features with 3d convolutional networks. In *Proceedings of the IEEE international conference on computer vision*, pages 4489–4497, 2015. 1, 2

[53] Du Tran, Heng Wang, Lorenzo Torresani, and Matt Feiszli. Video classification with channel-separated convolutional networks. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 5552–5561, 2019. 6, 7

[54] Du Tran, Heng Wang, Lorenzo Torresani, Jamie Ray, Yann LeCun, and Manohar Paluri. A closer look at spatiotemporal convolutions for action recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6450–6459, 2018. 1, 2

[55] Sudharshan S Vazhkudai, Bronis R de Supinski, Arthur S Bland, Al Geist, James Sexton, Jim Kahle, Christopher J Zimmer, Scott Atchley, Sarp Oral, Don E Maxwell, et al. The design, deployment, and evaluation of the coral pre-exascale systems. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis*, page 52. IEEE Press, 2018. 11

[56] Heng Wang, Du Tran, Lorenzo Torresani, and Matt Feiszli. Video modeling with correlation networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 352–361, 2020. 6

[57] Kuan Wang, Zhijian Liu, Yujun Lin, Ji Lin, and Song Han. Haq: Hardware-aware automated quantization. *arXiv preprint arXiv:1811.08886*, 2018. 2

[58] 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 *European Conference on Computer Vision*, pages 20–36. Springer, 2016. 1, 2, 4, 5, 6, 9

[59] Xiaolong Wang, Ross Girshick, Abhinav Gupta, and Kaiming He. Non-local neural networks. *arXiv preprint arXiv:1711.07971*, 10, 2017. 1, 2, 4, 5, 7, 9, 10

[60] Xiaolong Wang and Abhinav Gupta. Videos as space-time region graphs. *arXiv preprint arXiv:1806.01810*, 2018. 2, 4, 5, 6, 7

[61] Bichen Wu, Alvin Wan, Xiangyu Yue, Peter Jin, Sicheng Zhao, Noah Golmant, Amir Gholaminejad, Joseph Gonzalez, and Kurt Keutzer. Shift: A zero flop, zero parameter alternative to spatial convolutions. *arXiv preprint arXiv:1711.08141*, 2017. 1, 2, 3

[62] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1492–1500, 2017. 5

[63] Saining Xie, Chen Sun, Jonathan Huang, Zhuowen Tu, and Kevin Murphy. Rethinking spatiotemporal feature learning: Speed-accuracy trade-offs in video classification. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 305–321, 2018. 1, 2, 5

[64] Joe Yue-Hei Ng, Matthew Hausknecht, Sudheendra Vijayanarasimhan, Oriol Vinyals, Rajat Monga, and George Toderici. Beyond short snippets: Deep networks for video classification. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4694–4702, 2015. 2

[65] Christopher Zach, Thomas Pock, and Horst Bischof. A duality based approach for realtime tv-l 1 optical flow. In *Joint Pattern Recognition Symposium*, pages 214–223. Springer, 2007. 2, 6

[66] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. *CoRR*, abs/1707.01083, 2017. 2

[67] Huasong Zhong, Xianggen Liu, Yihui He, Yuchun Ma, and Kris Kitani. Shift-based primitives for efficient convolutional neural networks. *arXiv preprint arXiv:1809.08458*, 2018. 2

[68] Bolei Zhou, Alex Andonian, and Antonio Torralba. Temporal relational reasoning in videos. *arXiv preprint arXiv:1711.08496*, 2017. 1, 2, 4, 5, 6, 8

[69] Bolei Zhou, Aditya Khosla, Agata Lapedriza, Aude Oliva, and Antonio Torralba. Learning deep features for discriminative localization. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2921–2929, 2016. 12

[70] Chenzhuo Zhu, Song Han, Huizi Mao, and William J Dally. Trained ternary quantization. *International Conference on Learning Representations*, 2016. 2

[71] Xizhou Zhu, Yujie Wang, Jifeng Dai, Lu Yuan, and Yichen Wei. Flow-guided feature aggregation for video object detection. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 408–417, 2017. 8, 9

[72] Mohammadreza Zolfaghari, Kamaljeet Singh, and Thomas Brox. Eco: Efficient convolutional network for online video understanding. *arXiv preprint arXiv:1804.09066*, 2018. 1, 2, 4, 5, 6, 7, 8
[73] Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*, 2016. 2
[74] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. Learning transferable architectures for scalable image recognition. *arXiv preprint arXiv:1707.07012*, 2(6), 2017. 2

ACKNOWLEDGMENTS

We thank MIT Quest for Intelligence, MIT-IBM Watson AI Lab, MIT-SenseTime Alliance, Samsung, SONY, AWS, Google for supporting this research. We thank Oak Ridge National Lab for Summit supercomputer. We thank John Cohn for the support for our work.

Ji Lin is currently a second-year Ph.D. student at MIT EECS. Previously, he graduated from Department of Electronic Engineering, Tsinghua University. His research interests lie in efficient and hardware-friendly machine learning and its applications.

Chuang Gan is a research staff member at MIT-IBM Watson AI Lab. He is also an affiliated researcher at MIT EECS. His research interests focus on computer vision and machine learning.

Kuan Wang is a fourth-year undergraduate degree at Tsinghua University, and a visiting student at MIT, advised by Dr. Song Han. His current research interests lie on the intersection of computer vision, deep learning and efficient hardware architecture. He is a student member of the IEEE.

Song Han is an assistant professor at MIT EECS Department. Dr. Han received the Ph.D. degree in Electrical Engineering from Stanford University and B.S. degree in Electrical Engineering from Tsinghua University. Dr. Han's research focuses on efficient deep learning computing at the intersection between machine learning and computer architecture. He proposed "Deep Compression" and the "Efficient Inference Engine" that impacted the industry. He is a recipient of NSF CAREER Award, MIT Technology Review Innovators Under 35, best paper award at the ICLR'16 and FPGA'17, Facebook Faculty Award, SONY Faculty Award, AWS Machine Learning Research Award. Contact: songhan@mit.edu.