# D-STEP: Dynamic Spatio-Temporal Pruning

Avraham Raviv
avraham.r@partner.samsung.com
Yonatan Dinai
yonatan.dina@samsung.com
Igor Drozdov
igor.drozdov@partner.samsung.com
Niv Zehngut
niv.z@samsung.com
Ishay Goldin
ishay.goldin@samsung.com

Samsung Israel R&D Center
Tel Aviv
Israel

**Abstract**

Video processing requires analysis of spatial features that are changing over time. By combining spatial and temporal modelling together, a neural network can gain a better understanding of the scene with no increase in computation. Spatio-temporal modeling can also be used to identify redundant and sparse information in both the spatial and the temporal domains. In this work we present *Dynamic Spatio-Temporal Pruning*, *D-STEP*, a new, simple, yet efficient method for learning the evolution of spatial mapping between frames. More specifically, we used a cascade of lightweight policy networks to dynamically filter out, per input, regions and channels that do not provide information while also sharing information across time. Guided by the policy networks, the model is able to focus on relevant data and filters, avoiding unnecessary computations. Extensive evaluations on Something-Something-V2, Jester and Mini-Kinetics action recognition datasets demonstrate that the proposed method shows a significantly improved accuracy-compute trade-off over the current state-of-the-art methods. We release our code and trained models at https://github.com/DynamicAR/DSTEP.

## 1 Introduction

The exponential growth in online video applications creates a huge demand for systems that can recognize and localize actions, events, objects, and interactions in video. Computers with sufficient computing power can handle those heavy tasks in real-time, but devices with limited resources, such as mobile phones, cameras and AR/VR glasses, require more efficient algorithms.

Recently there has been a lot of focus on building lightweight network architectures. MobileNet [11, 12, 23] and EfficientNet [25] are popular architectures proposed for reduced computations. Although they have achieved notable success, they are geared towards image-level tasks such as classification and object detection, and are suboptimal for video analysis.

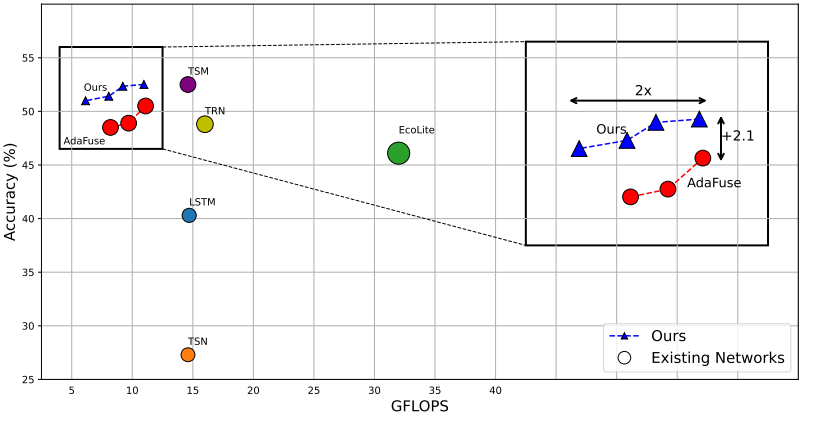


Figure 1: **Something-Something-V2 Validation Accuracy vs FLOPs.** Existing networks are marked with a circle, while the diameter of the circle is proportional to the model's size. Our model (blue triangles) provides comparable accuracy to AdaFuse while cutting the computation by half. With a similar number of FLOPs to AdaFuse, we achieve a state-of-the-art 52.7% top-1 accuracy on Something-Something-V2 with ResNet18 as a backbone.

Alternative networks dedicated to video have been proposed, such as MoViNets [14], but they lack a key component in videos, temporal aggregation of information from different timestamps.

An early approach for temporal aggregation was Temporal 3D ConvNets (also known as video convolutional networks) [3]. These networks seek to capture both short-term and long-term dynamics, but they require a lot of resources to train and evaluate. To address this issue, lightweight and memory-friendly architectures were proposed [4, 15], but their accuracy-computation trade-off was insufficient.

An alternative to changing the architecture is using the original architecture but pruning less significant channels [21]. In earlier works, pruning is static, i.e., weights are pruned offline during training and then the network remains sparse during inference. In later works, dynamic pruning is proposed [6], which preserves the entire structure of the network during training, but skips certain channels during inference. This methods added a simple policy network that identifies filters that contribute less to the final prediction.

Inspired by channel-wise dynamic pruning on images, AdaFuse [19] proposed an extension for video. A learned policy network dynamically combines information from current and previous frames' feature maps, specifying whether channels are to be computed, reused from past feature maps or skipped. This method aims to identify and avoid temporal redundancies in addition to utilizing temporal aggregation for improved accuracy.

In our work, we use policy networks not only to identify redundancy in the time domain but also sparsity in the spatial domain, ignoring unimportant regions. By using more information and a unique loss function, the network also allows better temporal aggregation. The policy network and its inputs have been redesigned with two major changes; the first is addressing both the temporal and the spatial domains to reduce computations. The second is to make decisions based on more valuable information.

To demonstrate the effectiveness of our approach, we run extensive experiments on common datasets, including Something-Something-V2, Kinetics and Jester. As shown in Figure 1, our method is able to reduce the number of FLOPs by 50% without any accuracy degradation compared to several leading algorithms. In addition, it is able to achieve the highest accuracy level by a wide margin without increasing the number of FLOPs. Our results demonstrate the effectiveness of our method focusing on relevant regions and filters and reusing past information. In summary, the main contributions of our work are as follows:

- A novel approach to dynamically prune tiles and filters, which identifies both spatial and channel sparsities as well as temporal redundancy, minimizing computations.
- The approach is generic for all spatio-temporal tasks, making it suitable for a wide range of video-specific architectures.
- A new state-of-the-art trade-off between accuracy and latency for action recognition tasks. Compared to the current leading adaptive methods, our method achieved +2% accuracy with similar FLOPs, and preserved accuracy while reducing half the FLOPs.

## 2 Related Work

Standard 2D CNNs were proven effective in solving a variety of video-related tasks such as action recognition. An early design [24] used a two-stream CNN to analyze both appearance (RGB) and motion (optical flow) inputs, but lacked the ability to infer complex temporal relationships. To alleviate it, 2D convolutions were extended to 3D convolutions [3, 27] operating on images and tensors concatenated from different timestamps. Following works used 3D convolutions more effectively [2] and demonstrated their advantages over 2D alternatives [8]. Recently, transformers [26, 35] have demonstrated impressive visual accuracy, though at a massive computational cost.

The high resource consumption of 3D convolutions and transformers has led to several studies offering alternatives for temporal modeling using 2D convolutions, and a number of enhancements and redesigned architectures have been proposed. Temporal Segment Networks (TSN) [30] derived averaged features from stride-sampled frames. Temporal Relation Network (TRN) [36] was designed to learn and reason about temporal dependencies between video frames at multiple time scales. RNN-based components such as LSTMs were integrated into video networks [28]. Temporal Shift Module (TSM) [17] performed a static 3D operation by using a preset fixed indentation for mixing channels from different timestamps.

Although these works have been successful, their efficiency is limited. In order to obtain a lightweight model, many methods focused on the architecture and the number of parameters. Pruning weights and channels [20], quantization [34], Neural Architecture Search [39], and Knowledge Distillation [10] are all successful model compression techniques. However, they are all static methods, i.e., the final model and the inference do not depend on the input. Dynamic methods, like ours, can be combined with or applied on top of the static methods. For images, Dynamic Channel Pruning [5] turns off some of the channels based on the input, while Dynamic Dual Gating [16] considers both channel pruning and spatial masking.

For Videos, FrameExit [7] propose a conditional early exiting framework, which automatically learns to process fewer frames for simpler videos and more frames for complex ones. AdaFocus [31, 32] maintains the main portion of the image while ignoring the rest, VA-RED$^2$ [22] suggests a redundancy reduction framework and performs pruning on the

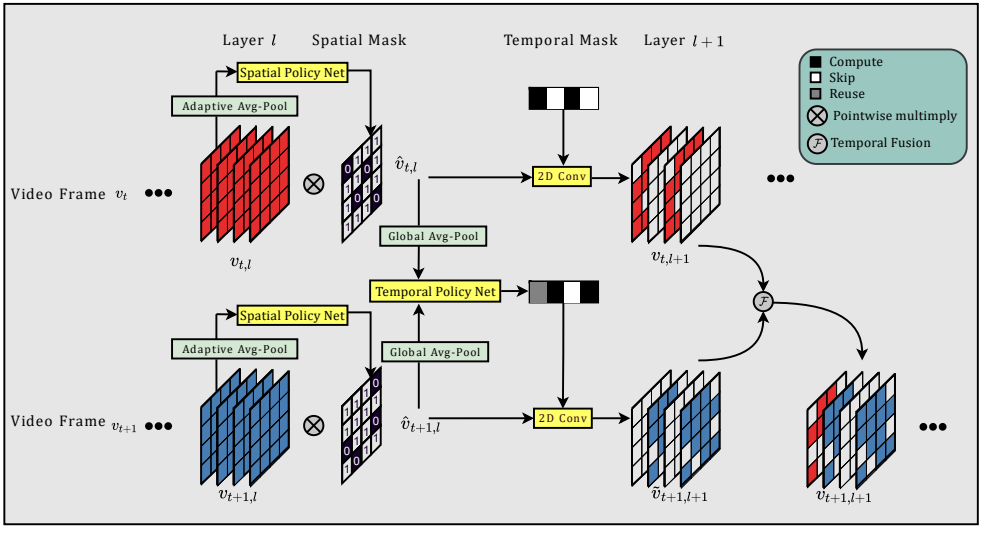


Figure 2: **An overview of the architecture**. We prune spatial and temporal redundancy and aggregate information during frames while paying attention to spatial maps $\mathcal{S}$. Each feature map at layer $l$ is first filtered through $\mathcal{S}$. Then, at time $t+1$, the 2D Conv layer processes *compute* channels (blue) in feature map $\mathcal{V}_{t+1,l}$, and fuses the *reuse* channels (red) from the history feature map $\mathcal{V}_{t,l+1}$. Best viewed in color.

temporal and channel dimensions, and AdaFuse [19] analyzes pairs of consecutive frames and adaptively identifies channels that can be skipped. Additionally, it suggests a fusion technique that can copy computed channels from the previous timestamp rather than recalculating unchanged information, hence saving computations. Our approach is based on the latter methods leading to a dynamic model whose inference depends on the input images and their evolution through time. Our model distinguishes itself from the recent by being the first to combine dynamic inference in both the spatial, temporal and channel domains. Furthermore, the same mechanism can be used for temporal aggregation as well, making the method accurate and efficient at the same time.

# 3 Method

Our method relies on simultaneously addressing spatial and channel sparsity, temporal redundancy and feature aggregation. We propose *Dynamic Spatio-TEmporal Pruning*, *D-STEP*, an efficient video processing method that dynamically identifies regions that contribute less to the final prediction and channels that can be reused from previous frames, saving unnecessary compute. The reuse of past channels also aids accuracy by serving as a feature aggregation mechanism.

## 3.1 Temporal and Spatial Dynamic Pruning

Our architecture employ two stages of adaptive policies – one regarding spatial aspects and another concerning temporal aspects – which we combine to form the core of the method.

As illustrated in Figure 2, per convolution layer or residual block, we have introduced two consecutive lightweight sub-networks: the first estimates a spatial mask and the second optimizes a temporal policy and identifies the most important filters. Spatial masks are applied at the spatial dimension, which means they look for unimportant tiles and avoid computing all corresponding filters, nullifying all channels. In contrast, and following [19], a temporal policy network predicts one of the following values for each channel – *compute, skip* and *reuse*. *Reusing* feature maps from previous timestamps achieves at once both a reduction in computations and temporal feature aggregation. Whenever the policy network dictates to *skip* a channel, the convolution filter is avoided reducing compute further. Notice that in this case a *skipped* channel can also be avoided when used as an input to the next layer reducing compute even more. To create the next layer's feature map, the network applies standard 2D convolutions on the rest of the channels, and the calculated features are merged with *reused* channels.

To formulate the whole layer process, consider a single 2D convolution. For layer $l$ in frame $t$, the output of a standard convolution is: $\mathcal{V}_{t,l+1} = \varphi(\mathcal{W} * \mathcal{V}_{t,l} + b)$, where $\mathcal{V}_{t,l} \in \mathbb{R}^{c_l \times h_l \times w_l}$ denotes the input feature map in layer $l$ at timestamp $t_l$ with $c_l$ channels and spatial dimension $h_l \times w_l$, and $\mathcal{V}_{t,l+1} \in \mathbb{R}^{c_{l+1} \times h_{l+1} \times w_{l+1}}$ is the output feature map, i.e. the feature map in layer $l+1$ at the same timestamp. $\mathcal{W} \in \mathbb{R}^{c_{l+1} \times k \times k \times c_l}$ denotes the convolution filters with kernel size $k \times k$, $b \in \mathbb{R}^{c_{l+1}}$ is the bias, and $\varphi$ is the activation function.

In between two successive layers, there are two stages. First, there is a spatial binary mask $\mathcal{S}$ that identifies spatial sparsity. $\mathcal{S}$ could be in the spatial dimensions of the outputs channel, i.e. $\mathbb{R}^{h_{l+1} \times w_{l+1}}$, meaning that the mask makes a decision per pixel. Alternatively, the input feature maps can be divided into $p \times p$ tiles in the spatial dimension, where the mask determines which tiles should be skipped. In this case, $\mathcal{S}$ is a matrix with $\lceil \frac{h_{l+1}}{p} \rceil \times \lceil \frac{w_{l+1}}{p} \rceil$ binary values. In order to evaluate the mask, we use *Average Pooling* operation to aggregate local information from input feature maps. After that, we use a standard $3 \times 3$ convolution to combine channels together and produce a 2D spatial attention map. During inference, we binarize the value by a simple round function. However, due to the non-differentiable nature of the binarization operation we rely on a modification of the *Gubmel-SoftMax* method [13], as used in [16], and described below.

In the second stage, a temporal channel-wise policy network is used, which yields a categorical vector $\mathcal{T} \in \{skip, reuse, compute\}^{c_{l+1}}$. in order to account for the masked spatial information within the temporal policy, we first mask out all channels using the spatial mask $\mathcal{S}$ (up-sampling the mask if the dimensions are inconsistent). Such masking allows the temporal policy network to attend only relevant information. The result of this multiplication is denoted by $\hat{\mathcal{V}}_{t,l}$. The temporal policy network processes two masked feature maps of two consecutive frames – $\hat{\mathcal{V}}_{t,l}$ and $\hat{\mathcal{V}}_{t+1,l}$. Based on their combined information, the adaptive policy determines whether each channel is uninformative (*skip*), needs to be copied from last frame (*reuse*), or needs to be calculated (*compute*). In particular, every channel is spatially reduced via *Global Average Pooling* operation, followed by a simple two-level MLP policy network that gets pairs of time-consecutive channel vectors and predicts whether to *skip*, *reuse*, or *compute* the channels. As for the channels that should be *computed* according to the policy network, for the informative tiles, the 2D convolution is computed, to get $\tilde{\mathcal{V}}_{t+1,l+1}$. Finally, we apply $\mathcal{T}$ to merge the reused data from the previous frame with the computed data from the current frame, i.e:

$$\mathcal{V}^{i,j,c}_{t+1,l+1} = \mathcal{S}^{i,j,c}_{t,l} \times [(\tilde{\mathcal{V}}^{i,j,c}_{t+1,l+1} | \mathcal{T}^{c}_{t,l} = compute) + (\mathcal{V}^{i,j,c}_{t,l+1} | \mathcal{T}^{c}_{t,l} = reuse)] \tag{1}$$

Notice the formula is written in pixel-wise notation for simplicity, while $\mathcal{S}$ is defined over spatial image patches, making the network HW acceleration efficient.

In our architecture we use a unique policy network for each ResNet block. While a shared network could have been used across several layers, this is not optimal as each block analyzes a different level of information. In addition, we only use a single policy network between the first and second convolution layers of a ResNet block. As mentioned in [1], this setting allows each ResNet block to use the full input and update the full output, and experimentally it performs better than applying multiple gates in a block.

The computation overhead of the policy networks is negligible. The total FLOPs of the temporal channel-wise policy networks is 0.14G for ResNet18 ($< 1\%$ of baseline) and 0.62G for ResNet50 (1.9%). The spatial policy networks use only a single $3 \times 3$ convolution to generate spatial attention maps from the pooled input features. The total FLOPs for these modules is 0.07M ($< 0.01\%$ of baseline) for ResNet18 and 0.045G (0.14%) for ResNet50.

## 3.2 Training Process

**Compute measurement**. Although *D-STEP* can be used to dynamically sparse any 2D convolution operation, regardless of the network architecture, as suggested in [19], we apply it here only to the first of the two convolutions composing each ResNet [9] residual block. The average network FLOPs count, $\mathcal{F}$, is the average over all frames of the sum of FLOPs in all dynamic residual blocks. Per block, let us denote $\mathcal{C}_{in}$ and $\mathcal{C}_{out}$ the block's number of input and output channels, $\mathcal{C}$ the number of channels between the two convolutions, and $k_0$, $k_1$ the kernels' sizes, then the amount of convolutional FLOPs per Residual block could be formulated as:

$$\mathcal{C}_{in}k_0^2 \sum_{c=1}^{\mathcal{C}} \sum_{i,j} \mathcal{S}_{i,j}(\mathcal{T}_t^c = compute || \mathcal{T}_{t+1}^c = reuse) + \mathcal{C}_{out}k_1^2 \sum_{c=1}^{\mathcal{C}} \sum_{i,j} \mathcal{S}_{i,j}(\mathcal{T}_t^c \neq skip) \quad (2)$$

**Loss function.** As we aim to both improve accuracy by aggregating spatio-temporal information and reduce computations by sparsing the features map, we use a loss function that combines these two objectives. The accuracy term, $\mathcal{L}_{task}$, is a *Cross-Entropy Loss* in our case as commonly used for classification, but can be adapted for other tasks.

To reduce computations we propose an effective and easy to control regularization term. Assume a static model requires $\mathcal{F}_S$ FLOPs, and we want to reduce it to a specific target of $\mathcal{F}_D$ FLOPs. Let $\mathcal{D} = \mathcal{F}_D/\mathcal{F}_S$, then using the loss term $\mathcal{L}_{spar} = (\mathcal{F}/\mathcal{F}_S - \mathcal{D})^2$ encourages the network to reduce computations but not below the target FLOPs count, $\mathcal{F}_D$. The even spread of our results over the FLOPs axis in Figure 1 demonstrates the ease of controlling the network compute budget. In other pruning methods, when the accuracy-compute trade-off is guided by weighting of the loss terms, it is much harder to guide the network towards a specific compute target.

As mentioned in earlier works [29], to avoid sub-optimal convergence of all the gates to 0s or 1s, the spatial sparsity has to be properly initialized. We adopt a recent approach [16], initially setting tight bounds on the sparsity level of the spatial mask, and relaxing those bounds during training using a dedicated loss term, $\mathcal{L}_{bound}$.

In contrast to other works [16, 19, 31], we warm-up the network for $\mathcal{K}$ epochs using only the accuracy loss $\mathcal{L}_{task}$ and bound loss $\mathcal{L}_{bound}$ and then activate the full combined loss to reduce computations, smoothly. Overall, the full training loss is formulated as:

$$\mathcal{L} = \mathcal{L}_{task} + \lambda(e) \times \mathcal{L}_{spar} + \gamma \times \mathcal{L}_{bound} \quad (3)$$

With $\lambda(e)$ set to zero for $\mathcal{K}$ epochs and gradually increased during training $\lambda(e) = \alpha \times e^{\beta \times e}$.

**Training non-differentiable operators.** The sparsity mechanism consists of two components, temporal and spatial. The temporal component should provide during inference a ternary map, $\mathcal{T} \in \{reuse, compute, skip\}^{C_{l+1}}$. During training the policy network propagates real numbers, and its output, $\hat{\mathcal{T}} \in \mathbb{R}^{3 \times C_{l+1}}$, calls for an *argmax* sampling. Since *argmax* is not differentiable, it cannot be optimized with backpropagation-based methods. In order to make this process differentiable, it is common to use score function estimators (e.g., REINFORCE [33]). Nevertheless, it does not fit our scenario due to the high dimension of the discrete variable, which causes the score function estimator to exhibit a high variance, leading to slow convergence [13] [18]. As an alternative followed [18], we use the *Gumbel SoftMax* reparameterization trick [13], which relaxes the discrete distribution into a continuous Gumbel distribution to approximate a differentiable version of the *argmax* operation. As a result, during inference and in the training forward pass we are able to sample the decision policy from a discrete distribution, while in the backward pass, we approximate the gradient of the discrete samples by computing it over the *Gumbel SoftMax* relaxation.

In the spatial case we would like to binarize the continuous policy network output, $\hat{\mathcal{S}} \in \mathbb{R}^{\lceil \frac{h_{l+1}}{p} \rceil \times \lceil \frac{w_{l+1}}{p} \rceil}$, into a spatial mask, $\mathcal{S} \in \{0,1\}^{\lceil \frac{h_{l+1}}{p} \rceil \times \lceil \frac{w_{l+1}}{p} \rceil}$. *Argmax* is not suitable here as it forces the output vector sum to 1, hence high values in one region force down values in other region, as opposed to our desire for independent decisions about each region's contribution to the final network outcome. Inspired by [16] we use a modified version of the *Gumbel SoftMax* which applies sigmoid over the relaxed continuous distribution rather than softmax, making a differentiable approximation for the binary map. As for the temporal case, we use the binary map during inference and forward pass and the differential approximation during backpropagation.

# 4 Results

In this section we demonstrate our method's performance. First, we demonstrate that our approach can significantly improve the accuracy vs. compute trade-off, surpassing other baselines by a wide margin on the Something-Something-V2 dataset. Then, across all datasets, our method consistently performs better than the comparable adaptive models. We further propose insights into the impact of each element in our architecture and show that combining spatial and temporal policies is highly effective.

We evaluate our method on common datasets including Something-Something-V2, Jester and Mini-Kinetics (a subset of the full Kinetics dataset). There are 174 human action labels in Something, 27 hand gesture classes in Jester, and 200 action classes in Mini-Kinetics. The datasets are already split into training and validation, containing 194k/25k, 119k/15k, and 121k/10k videos, respectively.

Following previous training procedures [17] allowed us to make a fair comparison: We uniformly sample $\mathcal{N}_f = 8$ frames from each video, while each frame has an input dimension of $224 \times 224$. Standard augmentations, such as random scaling and cropping, are used during training and center cropping is used during inference. We use ImageNet pretrained weights in all our networks, and set the hyperparameters as follows: 50 epochs with an initial learning rate of 0.001, decay x0.1 after 20 and 40 epochs, and batch size 16.

## 4.1 Comparisons with State-Of-The-Art Efficient Video Recognition Methods

First, we compare our method with several SOTA efficient video action recognition baselines: TSN [30], ECO [38], LSTM [28], TSM [17] and AdaFuse [19]. Our comparison was done on the Something-Something-V2 dataset, with two backbones: ResNet18 and ResNet50. As shown in Table 1, compared to static methods such as LSTM or TSM, our approach can significantly save computations while maintaining accuracy by using an adaptive policy. In addition, even compared with AdaFuse's dynamic model, our method gives a better compute vs. efficiency trade-off, largely due to the utilization of spatial sparsity.

Table 1: Action recognition results on Something-Something-V2 dataset. Compared to current baselines, our method provides several working points using different FLOPs targets ($\mathcal{D}_{high} = 0.8, \mathcal{D}_{medium} = 0.6, \mathcal{D}_{low} = 0.4$) with the highest accuracy using the same flops and $\times 2-3$ less computation without sacrificing accuracy.

| | ResNet18 | | | ResNet50 | | |
|---|---|---|---|---|---|---|
| **Model** | **#Params** | **FLOPs** | **Top 1** | **#Params** | **FLOPs** | **Top 1** |
| TSN | 11.2M | 14.6G | 27.3 | 24.3M | 33.2G | 27.8 |
| EcoLite | 47.5M | 32.0G | 46.1 | – | – | – |
| LSTM | 11.7M | 14.7G | 28.4 | – | – | – |
| TSM | 10.4M | 14.6G | 52.6 | 24.3M | 33.2G | **59.1** |
| AdaFuse + TSM | 15.6M | 11.0 | 50.9 | 37.8M | 22.4G | 57.3 |
| AdaFuse + TSN | 15.6M | 11.1 | 49.6 | 37.8M | 22.4G | 56.2 |
| D-STEP $\mathcal{D}_h$ | 15.6M | 12.2G | **52.7** | 37.8M | 21.4G | 57.4 |
| D-STEP $\mathcal{D}_m$ | 15.6M | 8.1G | 51.4 | 37.8M | 16.2G | 56.2 |
| D-STEP $\mathcal{D}_l$ | 15.6M | **6.14G** | 51.0 | 37.8M | **11.6G** | 53.7 |

Second, We compare our approach to leading adaptive inference methods, such as AR-Net, which adapts frame resolution, and AdaFuse, which applies a temporal and channel policy. Over a number of datasets, as shown in Table 2, we achieve a better accuracy-efficiency trade-off. In particular, on both Something-Something-V2 and Jester, we are able to reduce computations by $\sim 50\%$ with comparable or even higher accuracy with the ResNet18 based model. The fact that each dataset has unique characteristics illustrates that our method is robust and suited to a variety of video scenes. When evaluating on the Mini-Kinetics dataset, we faced the reproducibility issues discussed in [37]. For fair comparison, we report here the competitor results obtained using code provided by the authors on our video sequences. For consistency, and due to differences in the training batch size, results on Something-V2 and Jester datasets were also re-produced with the original AdaFuse code.

## 4.2 Ablation study on Dynamic Inference Methods

Table 3 provides deeper understanding of the contribution of different architectural components. The static model fails in terms accuracy due to lack of temporal aggregation. Static temporal shifts (TSM) are beneficial even on top of dynamic temporal shifts (*reuse*). Addressing spatial sparsity is very significant. It reduces computation by 16% while increasing accuracy by 0.8% or 1.4% when added to dynamic channel pruning and dynamic or static (TSM) temporal shifts, respectively.

Table 2: Comparison with adaptive video recognition models using ResNet18/ResNet50 backbones. With a significant reduction in FLOPs, our method significantly outperforms others by a substantial margin in accuracy.

| | Method | Params | Something-V2 | | Jester | | Mini-Kinetics | |
|---|---|---|---|---|---|---|---|---|
| | | | FLOPs | Top 1 | FLOPs | Top 1 | FLOPs | Top 1 |
| **ResNet18** | AdaFuse + TSN | 15.6M | 11.1G | 50.5 | 7.6G | 93.7 | 11.7G | 64.7 |
| | D-STEP $\mathcal{D}_h$ | 15.6M | 12.2G | **52.7** | 7.92G | **95.42** | 12.0G | **65.7** |
| | D-STEP $\mathcal{D}_m$ | 15.6M | 8.1G | 51.42 | 6.43G | 95.3 | 10.9G | 65.3 |
| | D-STEP $\mathcal{D}_l$ | 15.6M | **6.14G** | 50.99 | **3.59G** | 94.56 | **8.0G** | 64.1 |
| **ResNet50** | AR-Net | 63.0M | 41.4G | 18.9 | 21.2G | 87.8 | – | – |
| | AdaFuse + TSN | 37.8M | 18.1G | 56.8 | 16.1G | 94.7 | 23.8G | **68.3** |
| | D-STEP $\mathcal{D}_h$ | 37.8M | 21.4G | **57.4** | 16.1G | **95.7** | 21.7G | 67.3 |
| | D-STEP $\mathcal{D}_m$ | 37.8M | 16.2G | 56.2 | 11.84G | 95.54 | 19.7G | 67.1 |
| | D-STEP $\mathcal{D}_l$ | 37.8M | **11.6G** | 53.7 | **9.84G** | 95.16 | **12.4G** | 65.4 |

Table 3: Ablation study of dynamic inference components using ResNet18 on Something-V2 dataset. DCP - Dynamic Channel Pruning. TS - Temporal Shift. SS - Spatial Sparsity.

| Model | #Params | FLOPs | Top 1 | Top5 |
|---|---|---|---|---|
| Baseline – Static Model [TSN] | 11.2M | 14.6G | 27.3 | 38.0 |
| Static TS + DCP | 15.6M | 10.78G | 51.96 | 80.12 |
| Dynamic TS + DCP [AdaFuse] | 15.6M | 11.1G | 50.5 | 67.8 |
| Dynamic TS + DCP + SS | 15.6M | 9.32G | 51.29 | 79.15 |
| Static and Dynamic TS + DCP | 15.6M | 11.02G | 50.92 | 78.9 |
| Static and Dynamic TS + DCP + SS | 15.6M | **9.22G** | **52.34** | **80.48** |

In Figure 3 we show qualitative results of the efficiency achieved by the spatial policy network. A random video was sampled from the Something-V2 dataset and annotated using the computed spatial masks. In order to demonstrate the induced sparsity, we calculate for each frame the averaged *Gumbel Sigmoid* outputs across all network layers. As can be seen in the figure, the policy network focuses on the most indicative regions, in this case the hand connecting the USB cable to the computer.

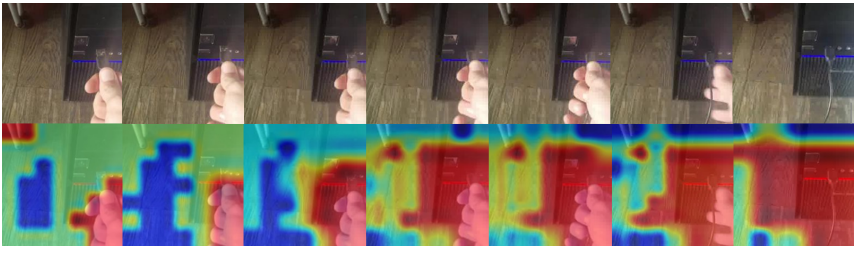

Figure 3: **Spatial gating visualization:** The top row shows a random clip from Something-V2, while the bottom row shows the averaged *Gumbel Sigmoid* outputs across all layers of the network. Regions marked in red correspond to areas with high computation while blue regions were mostly skipped.

### 4.3 Loss function

Figure 4 demonstrates the benefit of our sparsity loss term. In green, constant coefficient multiplied by the FLOPs count, as in [19]. In red, our $\mathcal{L}_{spar}$ multiplied by a constant coefficient, similar to [16]. In yellow, $\mathcal{L}_{spar}$ multiplied by $\lambda(e)$ increasing with epochs, and blue, with the addition of 10 warm-up epochs. $\mathcal{L}_{spar}$ is favorable to FLOPs count loss, variable sparsing coefficient improves further, and adding warm-up produces the highest accuracy with the lowest FLOPs by a wide margin (12%).

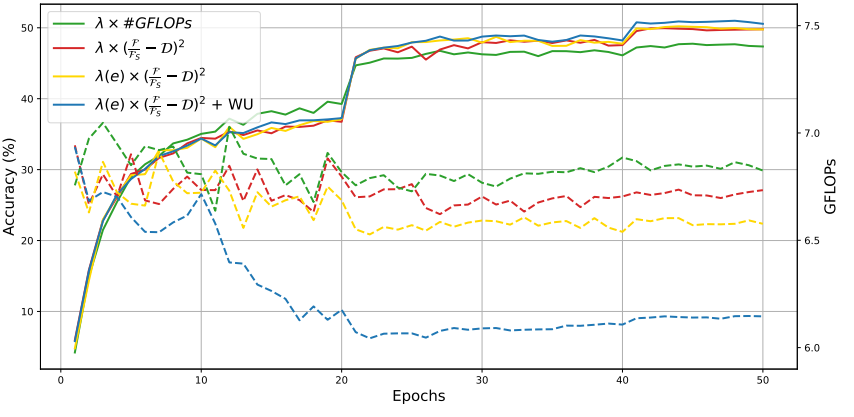


Figure 4: **Sparsity loss comparison.** A smoothly increasing coefficient with warm-up shows the best accuracy with a notable FLOPs reduction.

## 5 Conclusions

This work introduces a novel method for adaptive pruning that simultaneously addresses spatial sparsity, channel sparsity and temporal redundancy. In addition, the temporal component not only saves computation, but also allows effective feature aggregation. As a result of our suggested design, we were able to achieve the same accuracy as recent works with much fewer computations, and surpass them by a significant margin using the same number of FLOPs. Most video-specific architectures can incorporate our adaptive policies, and improve their results using a model that gains a better understanding of the spatio-temporal nature of video scenes.

## References

[1] Babak Ehteshami Bejnordi, Tijmen Blankevoort, and Max Welling. Batch-shaping for learning conditional channel gated networks. *arXiv preprint arXiv:1907.06627*, 2019.

[2] Joao Carreira and Andrew Zisserman. Quo vadis, action recognition? a new model and the kinetics dataset. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[3] A. Diba, M. Fayyaz, V. Sharma, A. H. Karami, M. Mahdi Arzani, R. Yousefzadeh, and L. Van Gool. Temporal 3D ConvNets: New Architecture and Transfer Learning for Video Classification. *ArXiv e-prints*, November 2017.

[4] Quanfu Fan, Chun-Fu (Ricarhd) Chen, Hilde Kuehne, Marco Pistoia, and David Cox. More Is Less: Learning Efficient Video Representations by Temporal Aggregation Modules. In *Advances in Neural Information Processing Systems 33*. 2019.

[5] Xitong Gao, Yiren Zhao, Łukasz Dudziak, Robert Mullins, and Cheng-zhong Xu. Dynamic channel pruning: Feature boosting and suppression. *arXiv preprint arXiv:1810.05331*, 2018.

[6] Xitong Gao, Yiren Zhao, Łukasz Dudziak, Robert Mullins, and Cheng zhong Xu. Dynamic channel pruning: Feature boosting and suppression. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=BJxh2j0qYm.

[7] Amir Ghodrati, Babak Ehteshami Bejnordi, and Amirhossein Habibian. Frameexit: Conditional early exiting for efficient video recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2021.

[8] 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] 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.

[10] Geoffrey Hinton, Oriol Vinyals, Jeff Dean, et al. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2(7), 2015.

[11] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, et al. Searching for mobilenetv3. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 1314–1324, 2019.

[12] 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.

[13] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*, 2016.

[14] Dan Kondratyuk, Liangzhe Yuan, Yandong Li, Li Zhang, Mingxing Tan, Matthew Brown, and Boqing Gong. Movinets: Mobile video networks for efficient video recognition. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 16020–16030, 2021.

[15] Okan Kopuklu, Neslihan Kose, Ahmet Gunduz, and Gerhard Rigoll. Resource efficient 3d convolutional neural networks. In *Proceedings of the IEEE/CVF International Conference on Computer Vision Workshops*, pages 0–0, 2019.

[16] Fanrong Li, Gang Li, Xiangyu He, and Jian Cheng. Dynamic dual gating neural networks. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 5330–5339, 2021.

[17] Ji Lin, Chuang Gan, and Song Han. Tsm: Temporal shift module for efficient video understanding. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 7083–7093, 2019.

[18] Yue Meng, Chung-Ching Lin, Rameswar Panda, Prasanna Sattigeri, Leonid Karlinsky, Aude Oliva, Kate Saenko, and Rogerio Feris. Ar-net: Adaptive frame resolution for efficient action recognition. In *European Conference on Computer Vision*, pages 86–104. Springer, 2020.

[19] Yue Meng, Rameswar Panda, Chung-Ching Lin, Prasanna Sattigeri, Leonid Karlinsky, Kate Saenko, Aude Oliva, and Rogerio Feris. Adafuse: Adaptive temporal fusion network for efficient action recognition. *arXiv preprint arXiv:2102.05775*, 2021.

[20] Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, and Jan Kautz. Pruning convolutional neural networks for resource efficient inference. *arXiv preprint arXiv:1611.06440*, 2016.

[21] Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, and Jan Kautz. Pruning convolutional neural networks for resource efficient inference. *arXiv preprint arXiv:1611.06440*, 2016.

[22] B. Pan, R. Panda, C. Fosco, C. Lin, A. Andonian, Y. Meng, K. Saenko, A. Oliva, and R. Feris. Va-red2: Video adaptive redundancy reduction. 2021.

[23] 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.

[24] Karen Simonyan and Andrew Zisserman. Two-stream convolutional networks for action recognition in videos. *Advances in neural information processing systems*, 27, 2014.

[25] Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International conference on machine learning*, pages 6105–6114. PMLR, 2019.

[26] Zhan Tong, Yibing Song, Jue Wang, and Limin Wang. Videomae: Masked autoencoders are data-efficient learners for self-supervised video pre-training. *arXiv preprint arXiv:2203.12602*, 2022.

[27] 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 (ICCV)*, December 2015.

[28] Amin Ullah, Jamil Ahmad, Khan Muhammad, Muhammad Sajjad, and Sung Wook Baik. Action recognition in video sequences using deep bi-directional lstm with cnn features. *IEEE access*, 6:1155–1166, 2017.

[29] Thomas Verelst and Tinne Tuytelaars. Dynamic convolutions: Exploiting spatial sparsity for faster inference. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2320–2329, 2020.

[30] 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.

[31] Yulin Wang, Zhaoxi Chen, Haojun Jiang, Shiji Song, Yizeng Han, and Gao Huang. Adaptive focus for efficient video recognition. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 16249–16258, 2021.

[32] Yulin Wang, Yang Yue, Yuanze Lin, Haojun Jiang, Zihang Lai, Victor Kulikov, Nikita Orlov, Humphrey Shi, and Gao Huang. Adafocus v2: End-to-end training of spatial dynamic networks for video recognition. *arXiv preprint arXiv:2112.14238*, 2021.

[33] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3):229–256, 1992.

[34] Jiaxiang Wu, Cong Leng, Yuhang Wang, Qinghao Hu, and Jian Cheng. Quantized convolutional neural networks for mobile devices. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4820–4828, 2016.

[35] Mingze Xu, Yuanjun Xiong, Hao Chen, Xinyu Li, Wei Xia, Zhuowen Tu, and Stefano Soatto. Long short-term transformer for online action detection. *Advances in Neural Information Processing Systems*, 34:1086–1099, 2021.

[36] Bolei Zhou, Alex Andonian, Aude Oliva, and Antonio Torralba. Temporal relational reasoning in videos. In *Proceedings of the European conference on computer vision (ECCV)*, pages 803–818, 2018.

[37] Yi Zhu, Xinyu Li, Chunhui Liu, Mohammadreza Zolfaghari, Yuanjun Xiong, Chongruo Wu, Zhi Zhang, Joseph Tighe, R Manmatha, and Mu Li. A comprehensive study of deep video action recognition. *arXiv preprint arXiv:2012.06567*, 2020.

[38] Mohammadreza Zolfaghari, Kamaljeet Singh, and Thomas Brox. Eco: Efficient convolutional network for online video understanding. In *Proceedings of the European conference on computer vision (ECCV)*, pages 695–712, 2018.

[39] Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*, 2016.