RubiksNet: Learnable 3D-Shift for Efficient Video Action Recognition

Linxi Fan[1]*, Shyamal Buch[1]*, Guanzhi Wang[1], Ryan Cao[1], Yuke Zhu[2,3], Juan Carlos Niebles[1], and Li Fei-Fei[1]

[1]Stanford Vision and Learning Lab [2]UT Austin [3]NVIDIA

Abstract. Video action recognition is a complex task dependent on modeling spatial and temporal context. Standard approaches rely on 2D or 3D convolutions to process such context, resulting in expensive operations with millions of parameters. Recent efficient architectures leverage a channel-wise shift-based primitive as a replacement for temporal convolutions, but remain bottlenecked by spatial convolution operations to maintain strong accuracy and a fixed-shift scheme. Naively extending such developments to a 3D setting is a difficult, intractable goal. To this end, we introduce RubiksNet, a new efficient architecture for video action recognition which is based on a proposed *learnable* 3D spatiotemporal shift operation instead. We analyze the suitability of our new primitive for video action recognition and explore several novel variations of our approach to enable stronger representational flexibility while maintaining an efficient design. We benchmark our approach on several standard video recognition datasets, and observe that our method achieves comparable or better accuracy than prior work on efficient video action recognition at a fraction of the performance cost, with 2.9-5.9x fewer parameters and 2.1-3.7x fewer FLOPs. We also perform a series of controlled ablation studies to verify our significant boost in the efficiency-accuracy tradeoff curve is rooted in the core contributions of our RubiksNet architecture.

Keywords: efficient action recognition, spatiotemporal, learnable shift, budget-constrained, video understanding

1 Introduction

Analyzing videos to recognize human actions is a critical task for general-purpose video understanding algorithms. However, action recognition can be computationally costly, requiring processing of several frames spatiotemporally to ascertain the correct action. As embodied applications of video recognition for autonomous agents and mobile devices continue to scale, the need for efficient recognition architectures with fewer parameters and compute operations remains ever-growing.

Prior efforts for video action recognition [1,23] rely on deep neural network architectures with expensive convolution operations across spatial and temporal

* equal contribution lead author; {jimfan,shyamal}@cs.stanford.edu

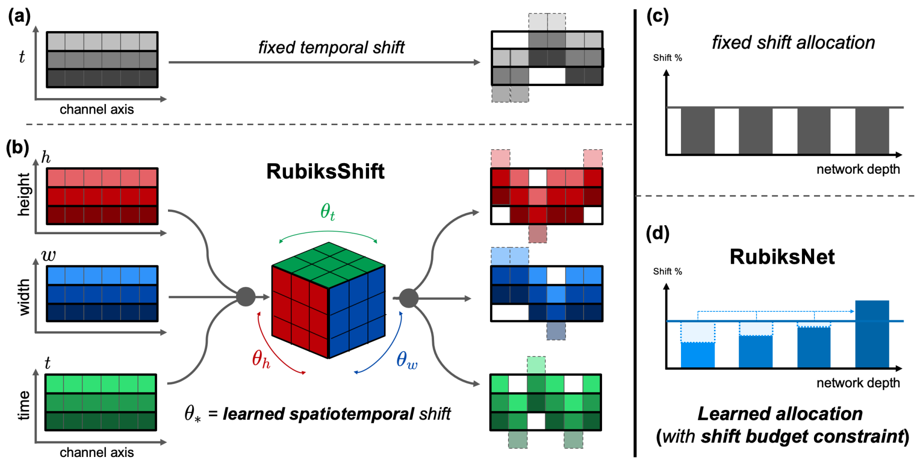


Fig. 1: *Top:* Prior work for efficient shift-based video recognition [16] has explored **(a)** fixed temporal shift operation on expensive 2D convolution features, with **(c)** a fixed shift allocation network design. *Bottom:* We introduce **RubiksNet**, a new architecture based on a **(b)** learnable 3D-shift layer (RubiksShift) that learns to perform spatial (h, w) and temporal (t) shift operations jointly end-to-end, while also **(d)** learning an effective layer-wise network allocation of a constrained shift budget. Our model significantly improves the accuracy-efficiency boundary.

context, which are often prohibitive for resource-constrained application domains. Recent work has proposed to improve the efficiency of the spatial and temporal operations separately [16,18,25,31]. However, they are still largely bounded by the efficiency of the base spatial aggregation method through spatial convolutions.

In images, spatial shift operations [11,29] have been proposed as a GPU-efficient alternative to traditional convolution operations, and architectures built using these operations have shown promise for efficient image recognition, though accuracy is limited. Recently, a temporal shift module (TSM) [16] based on hand-designed fixed temporal shift (Fig. 1, *top*) has been proposed to be used with existing 2D convolutional image recognition methods [26] for action recognition. However, the efficiency of their architecture family remains limited by the parameter and computation cost for spatial operations, as in other prior work.

A crucial observation is that much of the input spatiotemporal context contained in consecutive frames of a video sequence is often redundant to the action recognition task. Furthermore, the impact of modeling capacity on the action recognition task likely varies significantly at different depths in the architecture. In other words, action recognition in videos poses a unique opportunity for pushing the boundaries of the efficiency-accuracy tradeoff curve by considering efficient shift operations in both space and time dimensions with flexible allocations.

However, a **key limitation** towards a naive generalization from fixed temporal shift [16] to a spatiotemporal shift scheme is that the design space becomes intractable. In particular, we would need to exhaustively explore the number of channels that are shifted for each dimension, the magnitude of these shifts, and the underlying layer design combining each of these operations, among other design choices. Thus, there is a clear motivation for enabling the architecture to learn the shift operations during training itself. Recent work [11] has explored the possibility of learning shifts for 2D image processing. However, the challenge of generalizing this formulation to enable stable and effective spatiotemporal optimization of shift-based operations on high-dimensional video input has remained unexplored.

To this end, we propose **RubiksNet**: a new video action recognition architecture based on a novel **spatiotemporal** shift layer (*RubiksShift*) that **learns** to perform shift operations jointly on spatial and temporal context. We explore several variations of our design, and find that our architecture **learns effective allocations** of shift operations within a **constrained shift budget**. We benchmark our overall approach on several standard action recognition benchmarks, including large-scale temporal-focused datasets like Something-Something-v1 [6] and Something-Something-v2 [17], as well as others like UCF-101 [22] and HMDB [15]. We observe that our architecture can maintain competitive accuracy with the prior state-of-the-art on efficient shift-based action recognition [16] while simultaneously improving the efficiency and number of parameters by a large margin, up to 5.9x fewer parameters and 3.7x fewer FLOPs. Our controlled ablation analyses also demonstrate that these efficiency-accuracy gains come from the joint ability of our architecture to synergize spatial and temporal shift learning, as well as effective learned allocation of shift across the network.

2 Related Work

Action Recognition. The action recognition task in videos focuses on the classification of activities in video clips amongst a set of action classes. The initial set of deep network-based approaches processed frames individually using 2D convolutional neural networks [13,21,26]. For example, Temporal Segment Network (TSN) extracts features from sampled frames before averaging them for the final prediction [26]. While such methods are relatively efficient and parallelizable, they also do not model the temporal dynamics well. As such, the dominant paradigm in video action recognition is centered around the usage of spatial and temporal convolutions over the 3D space [1,5,12,23]. There are also other action recognition works that exploit temporal information [32]. While these deep networks are more accurate, the increase in computational cost has proven to be substantial and prohibitive for efficiency-conscious applications.

Recent progress in action recognition has largely focused on two directions: (1) improving efficiency by considering a mixture of 3D convolutions and separate spatial + temporal convolution operations [18,25,31,36], and (2) incorporating longer term temporal reasoning [27,34,36] to further improve the accuracy of the methods. We differentiate this work from prior efforts along both axes. Along the first, we note that the above methods make progress towards bringing cubic scaling of 3D convolutions towards the quadratic scaling of 2D convolutions, but the

spatial kernel remains an inherent strong bound on performance. Our approach offers an alternative that is much more efficient and has much fewer parameters by eliminating the need for 2D or 3D convolutions in the architecture. Along the second, our method introduces a learnable spatiotemporal shift operation that efficiently increases the effective receptive field capacity of the network, and allows it to flexibly allocate its capacity across the network, in contrast with prior efforts that leveraged fixed shift operations throughout the network [16].
Efficient Neural Networks for Images and Video. Convolutions have been the main computational primitive in deep neural network approaches for computer vision tasks [3,8,9,29,33]. Recently, the shift operation has been proposed as a more hardware efficient alternative to spatial convolution for image classification [11,29,35]. While shift has been examined as a replacement for 1D temporal convolutions recently [16], the possibility of a learnable 3D shift operation has remained unexplored due to the challenges afforded by joint learning of the shift primitive across spatial and temporal context, and the traditional advantage spatial convolutions hold in action recognition architectures in terms of accuracy. We aim to propose a technique that need not depend on any spatial convolution operations during inference – only shift and pointwise convolution.

Additionally, our work remains complementary to literature on neural network compression [30], so while our proposed method saves substantially on model size and computation cost, it is possible that application of such techniques can lead to further gains. We also highlight work that ShuffleNet [33], MobileNet [9,20], and SqueezeNet [10] for efficient 2D image classification. Tran et al. [24] factorizes 3D group convolutions while preserving channel interactions. The aim of our work is to develop video models that can similarly be applied in embodied vision applications, where efficiency is essential.

3 Technical Approach

In this section, we describe our proposed method for efficient action recognition, where the task is to take an input video clip and classify which of a set of human action categories is present. We first describe our proposed RubiksShift layers for replacing spatiotemporal convolutions with learnable spatiotemporal shifts. Then, we detail how we compose these operations into the RubiksNet architecture.

3.1 RubiksShift: Learnable 3D Shift

Spatiotemporal Convolution. In a traditional convolutional network with 3D spatiotemporal convolutions, the input to each layer of the network is a 4D tensor $F \in \mathbb{R}^{C\times T\times H\times W}$, C is the number of input channels, T the temporal length, and H, W are the height and width respectively. The 3D spatiotemporal convolution operation [23] is then defined as:

$$O_{c',t,h,w} = \sum_{c,i,j,k} K_{c',c,k,i,j} F_{c,t+\hat{k},h+\hat{i},w+\hat{j}} \tag{1}$$

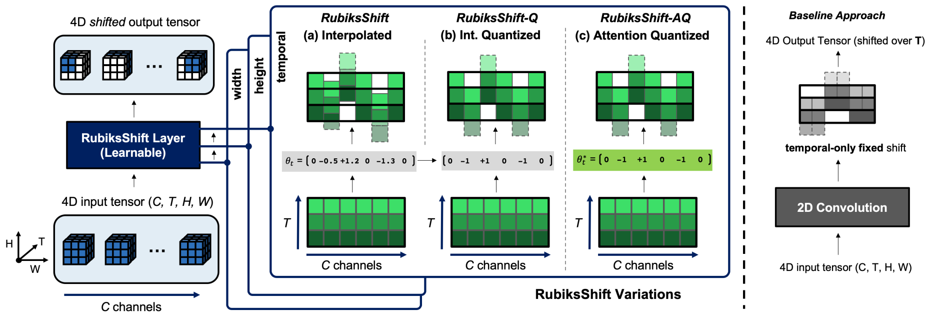

4D *shifted* output tensor
RubiksShift Layer (Learnable)
4D input tensor (*C*, *T*, *H*, *W*)
C channels
width
height
temporal
RubiksShift
(a) Interpolated
RubiksShift-Q
(b) Int. Quantized
RubiksShift-AQ
(c) Attention Quantized
$\theta_t = [0\ -0.5\ +1.2\ 0\ -1.3\ 0]$
$[0\ -1\ +1\ 0\ -1\ 0]$
$\theta_t^* = [0\ -1\ +1\ 0\ -1\ 0]$
T
C channels
RubiksShift Variations
Baseline Approach
4D Output Tensor (shifted over **T**)
temporal-only fixed shift
2D Convolution
4D input tensor (C, T, H, W)

Fig. 2: Our proposed RubiksShift layer (Section 3.1) is the primary driver of our overall RubiksNet architecture (Section 3.2). RubiksShift aims to perform a spatiotemporal shift operation on input 4D tensor along each of the input channels, for spatial and temporal axes. **(a)** Our primary RubiksShift layer is based on a continuous interpolated shift primitive, which enables a true gradient with low additional overhead, and **(b)** our RubiksShift-Q alternative is a quantized variant. **(c)** Finally, our RubiksShift-AQ variant enables learning integer shift using a temporal attention layer (also see Figure 3).

where $O \in \mathcal{R}^{C' \times T \times H \times W}$ is the output tensor, $K \in \mathbb{R}^{C' \times C \times T_K \times H_K \times W_K}$ is the 3D convolution kernel, i, j, k index along the temporal, height, and width dimensions of the kernel, and c, c' index along the channel dimensions. The indices $\hat{i}, \hat{j}, \hat{k}$ are the re-centered spatial and temporal indices, with $\hat{k} = k - \lfloor T_K/2 \rfloor, \hat{i} = i - \lfloor H_K/2 \rfloor, \hat{j} = j - \lfloor W_K/2 \rfloor$. Assuming $H = W$ for simplicity, the total number of parameters in this operation is thus $C \times C' \times T_K \times {H_K}^2$ and the computational cost is $C \times C' \times (T \times H^2) \times (T_K \times {H_K}^2)$. Indeed, this operation scales quadratically with the spatial input and cubically when considering temporal dimensions as well, both in parameters and number of operations.

Fixed Spatiotemporal Shift. In this context, we propose a spatiotemporal shift operation as an alternative to traditional 3D convolutions. Since the shift primitive proposed in prior work can be considered an efficient special case of depthwise-separable convolutions with fewer memory access calls [3,29,35], we can formalize the spatiotemporal shift operation as follows:

$$O'_{c,t,h,w} = \sum_{i,j,k} S_{c,k,i,j} F_{c,t+\hat{k},h+\hat{i},w+\hat{j}} \tag{2}$$

where $S \in \{0,1\}^{C \times T_K \times H_K \times W_K}$, such that $S_{c,k,i,j} = 1$ if and only if $(i,j,k) = (i_c, j_c, k_c)$, where i_c, j_c, k_c are the spatiotemporal shifts associated with each channel index c. Intuitively, if S is the identity tensor, no shift occurs since every element maps back to itself. We note that in practice, this operation can be efficiently implemented by indexing into the appropriate memory address, meaning that the shift operation itself in this form requires no floating point operations (FLOPs). We then apply a pointwise convolution to the output of Eq. 2 to integrate the information across channels to obtain our final output:

$$O_{c',t,h,w} = \sum_{c} P_{c',c} O'_{c,t,h,w} \tag{3}$$

where P is the pointwise convolution kernel with only $C \times C'$ parameters. Assuming $H = W$, the computational cost is $C \times C' \times (T \times H^2)$, much lower than the spatiotemporal convolution. Notably, when deploying such an architecture, we can fuse the shift and pointwise convolution operations into a single efficient kernel call, meaning the final parameters and operation complexity from the spatiotemporal shift operation itself is subsumed entirely.

Learnable Spatiotemporal Shift. Finally, a key design aspect of our proposed spatiotemporal shift operation which differentiates itself from prior work in temporal modeling [16] is the ability of our model to *learn* the temporal shift primitive. In particular, our method learns to shift over the joint spatiotemporal context jointly, which affords the network the ability to efficiently consider a significant span of spatiotemporal context with fewer overall parameters.

Following prior work in spatial shift [11], we consider a continuous form of the traditionally discrete shift operation, allowing us to optimize the shift parameters directly by backpropagation. We define the 3D shift learnable parameters as:

$$\theta = \{(\gamma_c, \alpha_c, \beta_c) \mid c \in C\} \tag{4}$$

where $\gamma_c, \alpha_c, \beta_c$ are the temporal, vertical, and horizontal shift *parameters* for each channel c. With this, we consider an alternative formulation of Equation 2:

$$O'_{c,t,h,w} = \sum_{i,j,k} S^{\theta}_{c,k,i,j} F_{c,t+\hat{k},h+\hat{i},w+\hat{j}} \tag{5}$$

$$S^{\theta}_{c,k,i,j} = \prod_{(z,g)\in\{(\gamma_c,k),(\alpha_c,i),(\beta_c,j)\}} \begin{cases} \Delta z & \text{if } g = \lceil z \rceil, \\ 1 - \Delta z & \text{if } g = \lfloor z \rfloor, \\ 0 & \text{otherwise} \end{cases} \tag{6}$$

with $\Delta z = z - \lfloor z \rfloor$ and g is the corresponding index to the z parameter dimension (e.g., γ_c corresponds to the time index k). Here, each sparse entry $S^{\theta}_{c,k,i,j}$ is a coefficient representing the *product* of interpolated shift contributions from all 3 dimensions. Intuitively, we are constructing a shift operation over an implicit continuous trilinearly-interpolated input activation tensor, where the interpolation is evaluated *sparsely* around the local neighborhood (2^3-point cube) around each shift operation at the location of each shift parameter. We note that this equation is once again a formalism; in practice, the operation can still be written efficiently on the GPU with minimal additional overhead with respect to discrete shift. We provide additional technical discussion in the supplement.

Taken together, we term this combined learnable spatiotemporal shift operation *RubiksShift* as shown in Figure 1. We observe that it enables our overall architecture to learn a joint spatiotemporal shift kernel that aggregates discriminative features over the full activation in an efficient manner.

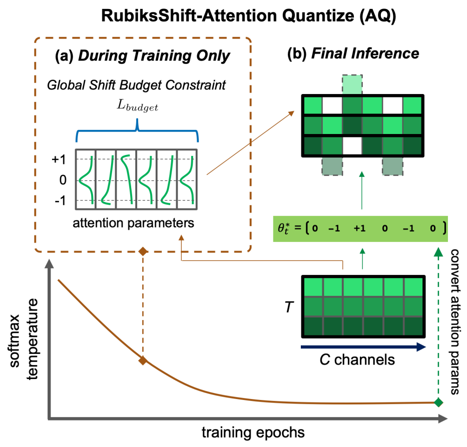


Fig. 3: **(a)** During training, our RubiksShift-AQ variant parameterizes shift with an attention distribution corresponding to integer shift values. We can then train quantized temporal shift with a true gradient (under a budget constraint), in constrast with interpolated quantized methods [2]. Attention softmax temperature is annealed during training. **(b)** After training, the resulting one-hot attention parameters are converted to final integer shift values for efficient inference at test time.

Interpolated Quantized shift. While the interpolated formulation above remains efficient, we can push efficiency a little higher by considering a quantized version of the above spatiotemporal shift scheme. Our second RubiksShift variant is a *naive* spatiotemporal extension based on an interpolated shift quantization mechanism on images [2]. Briefly, during training, the model maintains a copy of the interpolated shift parameters as floats. In the forward pass, all shifts are rounded to the nearest integer. In the backward pass, gradient is computed with respect to the continuous shifts and these parameters are updated by regular gradient descent. Our ablative analysis shows that while this technique does work, the lack of a *true* gradient ends up hindering performance by a large margin.
Attention Quantized Shift. To address the shortcomings of the interpolated quantized variant, we propose RubiksShift-AQ (Fig. 3) as an alternative way to quantize temporal shifts with *exact* gradient. Related to concurrent attention pruning work [7] for 2D object recognition, we formulate *temporal attention shift* as an operator for video action recognition that parameterizes shift with an attention distribution corresponding to integer shift values. The attention weights are then annealed to one-hot, integer-valued shifts over the course of the training process. In this manner, the model is able to flexibly and stably learn a high-capacity spatiotemporal representation suitable for video action recognition, but at final inference time is as efficient as quantized shift. Given an attention weight tensor $\mathbf{W_{attn}}$, we compute:

$$\mathbf{W_{attnshift}} = softmax\left(\tau_a \frac{\mathbf{W_{attn}}}{std(\mathbf{W_{attn}})}\right) \tag{7}$$

where τ_a denotes the temperature of the softmax. After the softmax operation, every value in $\mathbf{W}$ is normalized between 0 and 1, which represents the attention weight. At high τ_a, the values in $\mathbf{W}$ are close to uniform, while at extremely low τ_a, $\mathbf{W}$ approaches a binary tensor with only one non-zero entry along each channel slice. During training, we anneal τ_a exponentially to a low value, typically 10^{-3}, at which stage we take the hard max operation to convert attention shift $\mathbf{W_{attnshift}}$ to the equivalent discrete integer shift operation S for efficient inference.

Shift Operations Budget Constraint. Prior work [16,11] has noted that while shift operations save parameters and FLOPs, they can still incur latency cost through memory management. We incorporate this important aspect into our RubikShift design. A key feature of our proposed temporal attention shift is that we can apply a novel flexible constraint that corresponds to an overall "global shift budget", which penalizes the model for aggressively shifting. Importantly, the constraint is flexible in that it only penalizes a global metric for the shift – the specific allocation across layers at various depths in the network can then be learned by the network. Specifically, our budget constraint loss takes the form:

$$L_{budget} = \left\| \frac{1}{N_L} \sum_{l=1}^{N_L} \left(\frac{1}{C'} \sum_{c,*} \mathbf{W}^{(l)}_{\mathbf{nonzero}} \right) - B \right\| \tag{8}$$

where N_L is the number of layers, B is the shift budget between 0 and 1, and $\mathbf{W}^{(l)}_{\mathbf{nonzero}}$ denotes the attention weights in $\mathbf{W_{attnshift}}$ at layer l corresponding to the non-zero shift positions. We find that our proposed RubiksShift-AQ under budget constraint enables RubiksNet to discover interesting, non-uniform allocations of the shift budget across the network layers while preserving the global fraction of shift operations. To our knowledge, this is the first such method with learnable discrete shift under a shift resource constraint. Further, we find such learned allocation is critical to our overall design in careful ablation analysis.

3.2 RubiksNet: Model Architecture Design

Our overall architecture mirrors the residual block design in the ResNet architecture [8]. Each RubiksShift layer includes shift operations as described in Sec 3.1 and pointwise Conv1×1 operations (Eq. 3) to facilitate information exchange across channels. We place RubiksShift layers at the "residual shift" position [16], which fuses temporal information inside a residual branch. While we choose ResNet-like structure as our backbone design, the RubiksShift operator is flexible to be plugged into any architecture to replace the heavy convolutional layers.

Stable Shift Training. To improve the stability of spatiotemporal shift training, we normalize the shift gradients to use only the direction of change [11], rather than the magnitude. In practice, equal normalization over all 3 axes is sub-optimal, because the spatial dimensions of video inputs (e.g., 224 × 224) are substantially larger than the temporal dimension (e.g., 8 frames). We propose *scaled* gradient normalization, which scales the gradient vector for the spatial shifts $\Delta\theta_h$, $\Delta\theta_w$ and temporal shift $\Delta\theta_t$ onto an ellipsoid rather than a unit sphere:

$$\overline{\Delta\theta_h} = \frac{\Delta\theta_h}{Z}; \quad \overline{\Delta\theta_w} = \frac{\Delta\theta_w}{Z}; \quad \overline{\Delta\theta_t} = \frac{\lambda\Delta\theta_t}{Z}, \tag{9}$$

where λ is the temporal gradient scaling factor and Z is the normalization factor:

$$Z = \sqrt{||\Delta\theta_h||^2 + ||\Delta\theta_w||^2 + \lambda||\Delta\theta_t||^2}. \tag{10}$$

Table 1: Benchmark results on the Something-Something-v2 dataset [17]. RubiksNet offers strong performance across a range of base architectures as compared with TSM [16] architecture family. 2-clip accuracy metric per [16]; FLOPs reported for 1 clip, center crop for all architectures. (-) indicates value not reported. (#x) denotes efficiency savings factor relative to analogous size TSM model.

Method	Size	Input	1-Clip Val		2-Clip Val		#Param.	FLOPs/Video
			Top-1	Top-5	Top-1	Top-5		
TSN [26]	Large	8	30.0	60.5	30.4	61.0	24.3M	33G
TRN [36]	Large	8	48.8	77.6	-	-	18.3M	16G
bLVNet-TAM [4]	Large	8×2	59.1	86	-	-	25M	23.8G
TSM [16]	Large	8	58.8	85.6	61.3	87.3	24.3M	33G
	Medium	8	56.9	84.0	59.3	85.9	21.4M	29.4G
	Small	8	49.3	77.6	51.3	79.5	11.3M	14.6G
RubiksNet (Ours)	Large	8	59.0	85.2	61.7	87.3	8.5M (2.9x)	15.8G (2.1x)
	Medium	8	58.3	85.0	60.8	86.9	6.2M (3.5x)	11.2G (2.6x)
	Small	8	57.5	84.3	59.8	86.2	3.6M (3.1x)	6.8G (2.1x)
	Tiny	8	54.6	82.0	56.7	84.1	1.9M (5.9x)	3.9G (3.7x)

Architecture Size Versions. We design several variants of RubiksNet models with different sizes to accommodate different computational budgets, analogous to prior work on shift for image classification [11]. Please refer to our supplementary material for full architecture breakdown tables. In our experiments (e.g., Table 1), we consider different size *classes* of our architecture, RubiksNet-Large, Medium, Small which all have the same channel width but different layer depths. These size classes are chosen to correspond with TSM [16] operating on standard ResNet-50, ResNet-34, and ResNet-18 backbones respectively. Our RubiksNet-Tiny model has the same depth as RubiksNet-Small, but a thinner width. We leverage this spectrum of models to generate our Pareto curves in Sec 4.

4 Experiments and Analysis

In this section, we describe the experimental details (Sec. 4.1) and results of our method. In Sec 4.2, we detail our comparisons and analysis against the prior art methods on several standard benchmarks, and we show our architecture significantly pushes the state-of-the-art on the accuracy-efficiency frontier across large and smaller scale benchmarks. Finally, in Sec. 4.3, we conduct a series of controlled ablation studies and analysis to verify that the core scientific aspects of our architecture are responsible for this significant gain.

4.1 Experimental Setup

Overview. We leverage the Something-Something-v1 [6], Something-Something-v2 [17], UCF-101 [22], and HMDB [15] datasets to benchmark our approach. As

Table 2: Benchmark results on the Something-Something-v1 dataset [6]. Results are reported as 1-clip accuracy; FLOPs are reported for 1 clip, center crop for all architectures. (-) indicates value not reported.

Method	Input	Val Top-1	Val Top-5	#Param.	FLOPs/Video
I3D [1]	64	45.8	76.5	12.7M	111G
NL I3D + GCN [28]	32+32	46.1	76.8	303M	62.2G
S3D [31]	64	47.3	78.1	8.8M	66G
bLVNet-TAM [4]	8×2	46.4	76.6	25M	23.8G
TSN [26]	8	19.5	-	10.7M	16G
TRN [36]	8	34.4	-	18.3M	16G
ECO [37]	8	39.6	-	47.5M	32G
TSM [16]	8	45.6	74.2	24.3M	33G
RubiksNet (Ours)	8	46.4	74.5	8.5M	15.8G

a general rule, we follow training and evaluation protocols established in recent work [4,16] for fair comparison, including input and metrics. We implement our RubiksNet architecture and training pipeline in PyTorch, and write the RubiksShift operators in CUDA and Pytorch C++ for efficiency.[*]

Spatial Shift Pretraining. Per prior works [4,16], we pretrain the spatial portion of our RubiksNet models on ImageNet-1k [19] to reach comparable accuracy with spatial-shift image classification literature [11,29]. Analogous to inflated convolution kernels [1], we initialize the spatial components of the 3D RubiksShift layers with the learned 2D shift patterns before benchmark training.

Something-Something-V2 and -V1. Something-Something-(v2,v1) are both large-scale datasets; SS-v2 in particular has 220k video clips and 174 action classes. The action labels, such as "pushing something from left to right" and "putting something next to something", cannot be predicted by looking at only a single frame. This challenging aspect separates this benchmark from similar large-scale benchmarks like Kinetics [14], as also noted in [16,4]. Our spatial-pretrained RubiksNet is jointly trained end-to-end on the full benchmark, with the gradient normalization described in Eq. 9 for stability. For temporal attention shift, we initialize all the attention weights by sampling from $Uniform[1, 1.05]$, so that the initial attention distribution is roughly uniform over all possible shift locations. The softmax temperature is exponentially annealed from $T = 2.0$ to 10^{-3} over 40 epochs, before conversion to discrete integer shifts for evaluation.

UCF-101 and HMDB-51. These standard benchmarks have 101 and 51 action classes, respectively, and are smaller scale than the SS-v1/2 datasets. We follow the standard practice from prior work [16,26] and pretrain our model on Kinetics [1] before fine-tuning on each benchmark. For fine-tuning, we follow the same general learning schedule protocol as Something-Something-v2, normalizing gradients again per Equation 9 and following a similar attention shift annealing schedule.

[*]See `rubiksnet.stanford.edu` project page for supplementary material.

Table 3: Quantitative results on UCF-101 [22] and HMDB-51 [15]. 2-clip metric, 1-clip FLOPs per [16]. Pareto curve results in Fig. 4 and supplement.

Method	Size	UCF-101		HMDB-51		#Param.	FLOPs
		Val Top-1	Val Top-5	Val Top-1	Val Top-5		
TSN [26]	Large	91.7	99.2	64.7	89.9	23.7M	33G
TSM [16]	Large	95.2	99.5	73.5	94.3	23.7M	33G
RubiksNet	Large	95.5	99.6	74.6	94.4	8.5M	15.8G

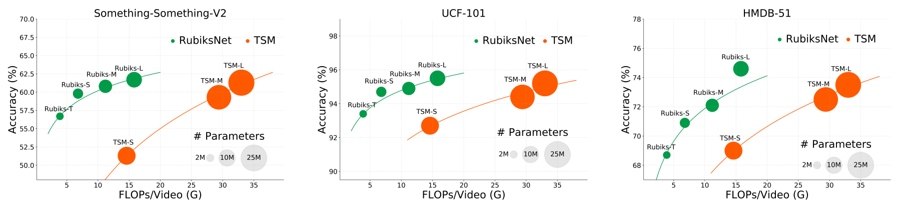


Fig. 4: We report the Pareto curves for our method compared with prior work [16], with size of the circle corresponding to the number of model parameters, as per Tables 1-3. Our RubiksNet architecture family consistently offers better performance-efficiency tradeoff across datasets. (Additional vis. in supplement)

4.2 Benchmark Comparisons and Analysis

Baselines. Our key point of comparison is the recent state-of-the-art shift-based action recognition architecture TSM [16] from ICCV 2019. In contrast with our technique, TSM operates with a hand-designed, fixed shift approach on the time dimension only, with heuristics found by extensive architecture tuning. TSM also has a fixed allocation scheme across its network. In our benchmark comparisons, we also include comparisons against much heavier but well-known architectures, like I3D, S3D, and others [1,28,31,4] for reference. Other networks, like TSN [26] and ECO [37] are also included as comparison points.

Evaluation. We follow the evaluation convention in prior work [16,27,28] and report results from two evaluation protocols. For “1-Clip Val” (Table 1), we sample only a single clip per video and the center 224×224 crop for evaluation. For “2-Clip Val”, we sample 2 clips per video and take 3 equally spaced 224×224 crops from the full resolution image scaled to 256 pixels on the shorter side. 2-Clip evaluation yields higher accuracy, but requires more computation than 1-Clip evaluation. We employ the same protocol for all methods in all the tables.

Quantitative Analysis. In Tables 1-3, we demonstrate that our proposed architectures consistently achieve competitive or better accuracies than their baseline counterparts at a range of model capacities, while achieving significantly higher efficiency in both parameter counts and FLOPs. Additionally, we provide a detailed efficiency analysis breakdown in our supplement.

Table 4: Ablation analysis: effect of learnable spatial and temporal shifts (Sec. 4.3). RubiksNet's ability to learn spatial and temporal shift jointly significantly improves over fixed, heuristic methods over spatial [29] and time [16] dimensions.

Spatial Shift Type	Temporal Shift Type	Val Top-1 (Large)	Val Top-1 (Small)
Learned	**Learned**	**71.4**	**69.5**
Learned	Fixed [16]	69.5	67.8
Fixed [29]	Learned	70.0	67.5
Fixed [29]	Fixed [16]	68.3	65.7

Table 5: Ablation analysis: impact of RubiksShift design (Sec. 4.3). Our attention-quantized variant is able to learn discrete shift operations with comparable performance to full interpolated, while observing shift/latency budget constraints.

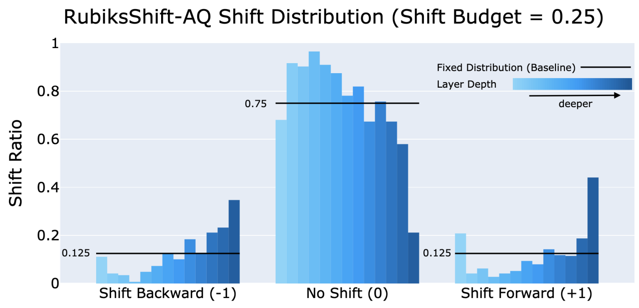
RubiksShift Type	Val Top-1	Exact Gradient	Integer Shift	Budget
Interpolated (RS)	61.7	✓		
Interpolated Quantized (RS-Q)	58.2		✓	
Attention Quantized (RS-AQ)	61.6	✓	✓	✓ (0.125)

We also benchmark several sizes of our model to draw a Pareto curve of performance and efficiency. In Figure 4, we visualize Pareto curves for our approach against TSM on multiple benchmarks. We observe a consistent trend that our method significantly improves the accuracy-efficiency tradeoff for efficient action recognition architectures. On Something-Something-v2 datasets (Table 1), our most efficient model, RubiksNet-Tiny, outperforms TSM-Small by 5.3 absolute percentage points, while reducing parameters by 5.9x and FLOPs by 3.7x. This indicates that RubiksNet performs especially well in the low-resource regime when compared against prior work. Towards the other extreme, our highest-end model (RubiksNet-Large) consumes 24.1% *fewer* parameters and comparable FLOPs to the *lowest-end* baseline model (TSM-Small), while exceeding the latter's top 1 accuracy by more than 10 absolute percentage points.

Qualitative Analysis. In Figure 6, we visualize spatiotemporal shift operations across different layers of a trained RubiksNet architecture, showing how the model efficiently incorporates video context to provide an action recognition prediction by increasing its receptive field and shift operations deeper in the network. We include additional video visualizations and analysis in the supplement.

4.3 Ablations and Analysis

Finally, we provide controlled ablations and analysis over our core RubiksNet design principles. In particular, we verify that *jointly learning* our 3D-shift operations is key to our accuracy-efficiency improvements by synergizing both spatial and temporal shift learning. Futher, we verify our RubiksShift-AQ variant provides strong performance under a strict global shift (latency) budget.



Shift-T Budget (Latency) (Ratio of Non-Zero Shift)		
Layers	Rubiks-AQ	Fixed
all	**0.25**	**0.25**
1-4	0.23	0.25
5-12	0.15	0.25
13-48	0.23	0.25
49-51	0.74	0.25

Fig. 5: We visualize our learned shifts distribution using our proposed attention quantized shift RubiksNet. The bottom labels are the different shifts $(-1, 0, +1)$ for a kernel size 3 RubiksShift-AQ temporal kernel, and the y-axis shows the proportion of channels with that temporal shift operation. Each colored bar represents a different layer, and we increase the depth of the network moving left to right. We observe that RubiksNet is able to learn to save its shift operations budget from early layers (few nonzero shift operations) to increase temporal modeling ability at deeper ones. Table 4 shows how this learned allocation consistently improves over heuristic techniques like TSM [16] that have a fixed shift budget allocation regardless of depth (shown by black horizontal bars above).

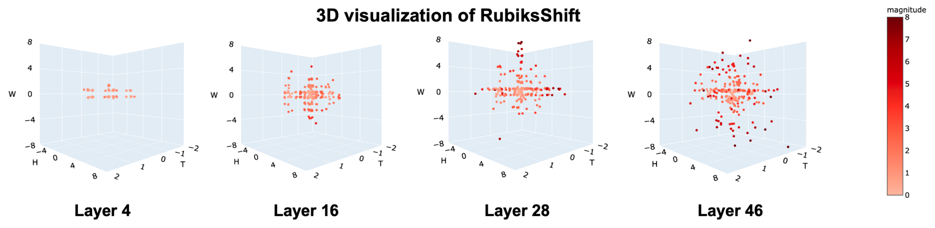


Fig. 6: We visualize the overall learned interpolated shift distribution across spatial (H, W) and temporal (T) dimensions at different layers in the RubiksNet architecture. RubiksNet is conservative with shift operations in early layers, while increasing the 3D receptive field in deeper layers for better spatiotemporal modeling. Please refer to supplement for additional video visualizations.

Ablation: Learned vs. Fixed. Our first ablation experiment provides a controlled analysis on the effect of learning the spatial and temporal shift aspects respectively. We describe our results in Table 4. We report top-1 accuracy results on the HMDB dataset for both Large and Small model size class. Critically, the architecture in all cases remains *constant* within a given model class size so there is no confounder due to different backbones. The only change is whether an aspect is "learned" or "fixed". In the fixed cases, we initialize the spatial and temporal shift parameters based on the heuristic initialization provided by ShiftNet [29] and TSM [16], respectively. Spatial and temporal learned cases are based on the RubiksNet (RubiksShift-AQ) method. For learned temporal, we set our budget constraint to 0.25 to exactly match that of the TSM [16] fixed setting for fair comparison and to ensure the same number of shift operations are performed. We find that our RubiksNet approach for learning spatial and temporal dimensions jointly consistently outperforms the ablations.

Ablation: RubiksShift-AQ. Our second ablation verifies the efficacy of the proposed RubiksShift layers and its variations. We report our results in Table 5. Here, we highlight that our RubiksShift-AQ is able to achieve comparable accuracy with a budget constraint of only 0.125 shift ratio, in comparison to the full RubikShift variant. In contrast with the naive RubiksShift-Q variant, RubiksShift-AQ enables discrete shift learned with true gradient and substantially outperforms. We observe that given a shift budget constraint, our attention shift mechanism is able to learn nontrivial temporal patterns (Figure 5) without hand-engineered prior knowledge. The network chooses to allocate more non-zero shifts for deeper layers, likely because heavier information exchange in the more abstract feature space is beneficial to the network's temporal modeling capability. Such findings are in alignment with traditional hand-designed "top-heavy" spatiotemporal convolutional networks [31]. Importantly, RubiksNet's learned temporal shift pattern can be thought of as an allocation of the limited "temporal modeling power budget". Prior works like [31] enumerate many configurations of temporal modeling allocation (i.e. permutations of Conv2D and Conv3D layers), and test them individually to find the best candidate. In contrast, our proposed method discovers a good temporal allocation pattern from random temporal initialization.

5 Conclusion

We introduced RubiksNet, a new efficient architecture for video action recognition. We examined the potential for a model based on our proposed 3D-spatiotemporal RubiksShift operations, and explored several novel variations of our design that enable stable joint training with flexible shift budget allocation. We benchmarked our method on several standard action recognition benchmarks, and find that RubiksNet can match or exceed the accuracies given by the previous state-of-the-art shift-based action recognition architecture at a fraction of the parameter and FLOP cost. Through careful and controlled ablations, we verified these gains are rooted in our core architecture contributions, from the joint learning of the spatial and temporal shifts to RubiksNet's ability to learn a flexible allocation of shift budget to maximize accuracy at minimal shift cost.

Acknowledgements. L. Fan and S. Buch are supported by SGF and NDSEG fellowships, respectively. This research was supported in part by grants from Toyota Research Institute (TRI), ONR grant N00014-19-1-2477, and ONR MURI grant N00014-16-1-2127. Some computational support for experiments was provided by Google Cloud and NVIDIA. This article reflects the authors' opinions and conclusions, and not any other entity. We thank Ji Lin, Song Han, De-An Huang, Danfei Xu, the general Stanford Vision Lab (SVL) community, and our anonymous reviewers for helpful comments and discussion.

References

1. Carreira, J., Zisserman, A.: Quo vadis, action recognition? a new model and the kinetics dataset. In: proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 6299–6308 (2017) 1, 3, 10, 11
2. Chen, W., Xie, D., Zhang, Y., Pu, S.: All you need is a few shifts: Designing efficient convolutional neural networks for image classification. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 7241–7250 (2019) 7
3. Chollet, F.: Xception: Deep learning with depthwise separable convolutions. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 1251–1258 (2017) 4, 5
4. Fan, Q., Chen, C.F.R., Kuehne, H., Pistoia, M., Cox, D.: More is less: Learning efficient video representations by big-little network and depthwise temporal aggregation. In: Advances in Neural Information Processing Systems. pp. 2261–2270 (2019) 9, 10, 11
5. Feichtenhofer, C., Pinz, A., Wildes, R.: Spatiotemporal residual networks for video action recognition. In: Advances in neural information processing systems. pp. 3468–3476 (2016) 3
6. Goyal, R., Ebrahimi Kahou, S., Michalski, V., Materzynska, J., Westphal, S., Kim, H., Haenel, V., Fruend, I., Yianilos, P., Mueller-Freitag, M., Hoppe, F., Thurau, C., Bax, I., Memisevic, R.: The "something something" video database for learning and evaluating visual common sense. In: The IEEE International Conference on Computer Vision (ICCV) (Oct 2017) 3, 9, 10
7. Hacene, G.B., Lassance, C., Gripon, V., Courbariaux, M., Bengio, Y.: Attention Based Pruning for Shift Networks. arXiv:1905.12300 [cs] (May 2019) 7
8. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 770–778 (2016) 4, 8
9. Howard, A.G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., Adam, H.: Mobilenets: Efficient convolutional neural networks for mobile vision applications. arXiv preprint arXiv:1704.04861 (2017) 4
10. Iandola, F.N., Han, S., Moskewicz, M.W., Ashraf, K., Dally, W.J., Keutzer, K.: Squeezenet: Alexnet-level accuracy with 50x fewer parameters and¡ 0.5 mb model size. arXiv preprint arXiv:1602.07360 (2016) 4
11. Jeon, Y., Kim, J.: Constructing fast network through deconstruction of convolution. In: Advances in Neural Information Processing Systems. pp. 5951–5961 (2018) 2, 3, 4, 6, 8, 9, 10
12. Ji, S., Xu, W., Yang, M., Yu, K.: 3d convolutional neural networks for human action recognition. IEEE transactions on pattern analysis and machine intelligence **35**(1), 221–231 (2012) 3
13. Karpathy, A., Toderici, G., Shetty, S., Leung, T., Sukthankar, R., Fei-Fei, L.: Large-scale video classification with convolutional neural networks. In: Proceedings of the IEEE conference on Computer Vision and Pattern Recognition. pp. 1725–1732 (2014) 3
14. Kay, W., Carreira, J., Simonyan, K., Zhang, B., Hillier, C., Vijayanarasimhan, S., Viola, F., Green, T., Back, T., Natsev, P., et al.: The kinetics human action video dataset. arXiv preprint arXiv:1705.06950 (2017) 10
15. Kuehne, H., Jhuang, H., Garrote, E., Poggio, T., Serre, T.: Hmdb: a large video database for human motion recognition. In: Computer Vision (ICCV), 2011 IEEE International Conference on. pp. 2556–2563. IEEE (2011) 3, 9, 11

16. Lin, J., Gan, C., Han, S.: Tsm: Temporal shift module for efficient video understanding (2018) 2, 3, 4, 6, 8, 9, 10, 11, 12, 13, 14
17. Mahdisoltani, F., Berger, G., Gharbieh, W., Fleet, D., Memisevic, R.: On the effectiveness of task granularity for transfer learning. arXiv preprint arXiv:1804.09235 (2018) 3, 9
18. Qiu, Z., Yao, T., Mei, T.: Learning spatio-temporal representation with pseudo-3d residual networks. In: proceedings of the IEEE International Conference on Computer Vision. pp. 5533–5541 (2017) 2, 3
19. Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., et al.: Imagenet large scale visual recognition challenge. International Journal of Computer Vision **115**(3), 211–252 (2015) 10
20. Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., Chen, L.C.: Mobilenetv2: Inverted residuals and linear bottlenecks. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 4510–4520 (2018) 4
21. Simonyan, K., Zisserman, A.: Two-stream convolutional networks for action recognition in videos. In: Advances in neural information processing systems. pp. 568–576 (2014) 3
22. Soomro, K., Zamir, A.R., Shah, M.: Ucf101: A dataset of 101 human actions classes from videos in the wild. arXiv preprint arXiv:1212.0402 (2012) 3, 9, 11
23. Tran, D., Bourdev, L., Fergus, R., Torresani, L., Paluri, M.: Learning spatiotemporal features with 3d convolutional networks. In: Proceedings of the IEEE international conference on computer vision. pp. 4489–4497 (2015) 1, 3, 4
24. Tran, D., Wang, H., Torresani, L., Feiszli, M.: Video classification with channel-separated convolutional networks. In: Proceedings of the IEEE International Conference on Computer Vision. pp. 5552–5561 (2019) 4
25. Tran, D., Wang, H., Torresani, L., Ray, J., LeCun, Y., Paluri, M.: A closer look at spatiotemporal convolutions for action recognition. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 6450–6459 (2018) 2, 3
26. Wang, L., Xiong, Y., Wang, Z., Qiao, Y., Lin, D., Tang, X., Van Gool, L.: Temporal segment networks: Towards good practices for deep action recognition. In: European Conference on Computer Vision. pp. 20–36. Springer (2016) 2, 3, 9, 10, 11
27. Wang, X., Girshick, R., Gupta, A., He, K.: Non-local neural networks. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 7794–7803 (2018) 3, 11
28. Wang, X., Gupta, A.: Videos as space-time region graphs. In: Proceedings of the European Conference on Computer Vision (ECCV). pp. 399–417 (2018) 10, 11
29. Wu, B., Wan, A., Yue, X., Jin, P., Zhao, S., Golmant, N., Gholaminejad, A., Gonzalez, J., Keutzer, K.: Shift: A zero flop, zero parameter alternative to spatial convolutions. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 9127–9135 (2018) 2, 4, 5, 10, 12, 13
30. Wu, C.Y., Zaheer, M., Hu, H., Manmatha, R., Smola, A.J., Krähenbühl, P.: Compressed video action recognition. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 6026–6035 (2018) 4
31. Xie, S., Sun, C., Huang, J., Tu, Z., Murphy, K.: Rethinking spatiotemporal feature learning: Speed-accuracy trade-offs in video classification. In: Proceedings of the European Conference on Computer Vision (ECCV). pp. 305–321 (2018) 2, 3, 10, 11, 14
32. Yao, G., Lei, T., Zhong, J.: A review of convolutional-neural-network-based action recognition. Pattern Recognition Letters **118**, 14–22 (2019) 3

33. Zhang, X., Zhou, X., Lin, M., Sun, J.: Shufflenet: An extremely efficient convolutional neural network for mobile devices. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 6848–6856 (2018) 4
34. Zhao, Y., Xiong, Y., Lin, D.: Trajectory convolution for action recognition. In: Advances in neural information processing systems. pp. 2204–2215 (2018) 3
35. Zhong, H., Liu, X., He, Y., Ma, Y., Kitani, K.: Shift-based primitives for efficient convolutional neural networks. arXiv preprint arXiv:1809.08458 (2018) 4, 5
36. Zhou, B., Andonian, A., Oliva, A., Torralba, A.: Temporal relational reasoning in videos. In: Proceedings of the European Conference on Computer Vision (ECCV). pp. 803–818 (2018) 3, 9, 10
37. Zolfaghari, M., Singh, K., Brox, T.: Eco: Efficient convolutional network for online video understanding. In: Proceedings of the European Conference on Computer Vision (ECCV). pp. 695–712 (2018) 10, 11

RubiksNet: Learnable 3D-Shift for Efficient Video Action Recognition
(Supplementary Material)

In this supplementary material, we include:

1. **Additional Visualizations** of our model in Section A1. We include video visualizations in our overview video (see `rubiksnet.stanford.edu`).
2. **Additional Architecture Details** in Section A2 which provides additional details (e.g. size classes, layout) at the network-level for our architecture.
3. **Interpolated Shift Equation Details** in Section A3 provides expanded technical discussion of the 3D shift equations in the main paper.
4. **Efficiency Analysis Details** in Section A4 provides additional details and breakdowns at the layer- and operation-level for our architecture to highlight how our efficiency gains reported in the main paper are rooted in our main proposed learnable 3D RubiksShift operations.
5. **Additional Results** in Section A5, including additional results for the main benchmarks reported in the paper as well as a comparison (verifying consistent improvement in efficiency-accuracy) on the **Kinetics** dataset against our main shift-based action recognition baseline (TSM) from ICCV19.
6. **Additional Training Details** in Section A6 which provides additional details (e.g. hyperparameters, learning rate schedule) for training.
7. **Code release** is available on the project website `rubiksnet.stanford.edu`.

A1 Additional Visualizations

We include additional visualizations of the learned 3D-shift weights in our **supplementary video**. In Figure A1, we show an expansion of the layer views from Figure 6 in the main paper, adding more views of the 3D filters from different angles.

A2 Architecture Details

We include additional architecture details in Table A1. This table captures the architecture layout details for RubiksNet-Large, Medium, Small, Tiny, respectively. Across all RubiksNet architectures, we follow an overall ResNet block design style as per prior work [16]. Our spatial shift is designed to be compatible with transfer from 2D spatial shift pretraining [11,29]. We plan to open source our implementation, including our PyTorch, PyTorch C++, and CUDA code. Our RubiksNet architecture primarily relies on the shift operation and (pointwise) convolution operation for its spatiotemporal modeling. In Sec. A4, we show our efficiency gains are rooted in our new (generic) RubiksShift Block, consisting of 3D

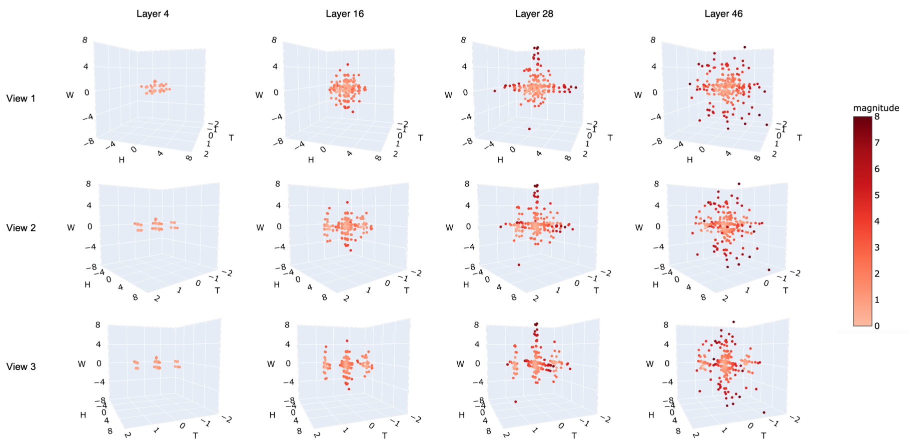


Fig. A1: Expanded visualization of Figure 6 in the main paper, showing different 3D viewpoints of the visualized spatiotemporal RubiksShift. We include further visualizations in our supplementary video.

learnable shift layers (between the Pointwise Conv/ReLU/BN operations). Our novel RubiksShift operation requires no spatial or spatiotemporal convolutions and can jointly learn spatiotemporal 3D shift.

A3 Interpolated Shift Equation Details

In this section, we provide additional discussion for our technical discussion for interpolated 3D shift. From the main paper, Equation 5 and 6 can be expanded as follows:

$$
\begin{aligned}
O'_{c,t,h,w} &= \tilde{F}_{c,t+\gamma_c,h+\alpha_c,w+\beta_c} \\
&= Z_c^1 \cdot (1-\Delta\gamma_c)\cdot(1-\Delta\alpha_c)\cdot(1-\Delta\beta_c) \\
&+ Z_c^2 \cdot \Delta\gamma_c \cdot (1-\Delta\alpha_c)\cdot(1-\Delta\beta_c) \\
&+ Z_c^3 \cdot (1-\Delta\gamma_c)\cdot \Delta\alpha_c \cdot (1-\Delta\beta_c) \\
&+ Z_c^4 \cdot \Delta\gamma_c \cdot \Delta\alpha_c \cdot (1-\Delta\beta_c) \\
&+ Z_c^5 \cdot (1-\Delta\gamma_c)\cdot(1-\Delta\alpha_c)\cdot \Delta\beta_c \\
&+ Z_c^6 \cdot \Delta\gamma_c \cdot (1-\Delta\alpha_c)\cdot \Delta\beta_c \\
&+ Z_c^7 \cdot (1-\Delta\gamma_c)\cdot \Delta\alpha_c \cdot \Delta\beta_c \\
&+ Z_c^8 \cdot \Delta\gamma_c \cdot \Delta\alpha_c \cdot \Delta\beta_c
\end{aligned}
$$

Table A1: RubiksNet Architecture Table, for different size classes (Large, Medium, Small, Tiny). As described in the main paper, these size classes are grouped to correspond with TSM. N_c refers to the number of output classes for dataset. Please refer to Sec. 4 for efficiency analysis on all the size classes and Sec. A4 for lower-level efficiency analysis.

Group	Type	Out Channels		Repeat	Stride
		L/M/S	T	L/M/S/T	
-	Input Block	72	54	1	2
1	RubiksShift Block	72	54	1	1
	RubiksShift Block			1	2
	RubiksShift Block			2	1
2	RubiksShift Block	144	108	1	2
	RubiksShift Block			7/3/3/3	1
3	RubiksShift Block	288	216	1	2
	RubiksShift Block			35/22/5/5	1
4	RubiksShift Block	576	432	1	2
	RubiksShift Block			2	1
-	Avg Pool	-	-	1	-
-	FC	-	N_c	1	-

where $\tilde{F}_{c,t+\gamma_c,h+\alpha_c,w+\beta_c}$ is the corresponding interpolated value at position (t,h,w) of the feature map at channel c after shift and

$$\begin{aligned}
Z_c^1 &= F_{c,t+\lfloor\gamma_c\rfloor,h+\lfloor\alpha_c\rfloor,w+\lfloor\beta_c\rfloor}, Z_c^2 = F_{c,t+\lceil\gamma_c\rceil,h+\lfloor\alpha_c\rfloor,w+\lfloor\beta_c\rfloor},\\
Z_c^3 &= F_{c,t+\lfloor\gamma_c\rfloor,h+\lceil\alpha_c\rceil,w+\lfloor\beta_c\rfloor}, Z_c^4 = F_{c,t+\lceil\gamma_c\rceil,h+\lceil\alpha_c\rceil,w+\lfloor\beta_c\rfloor},\\
Z_c^5 &= F_{c,t+\lfloor\gamma_c\rfloor,h+\lfloor\alpha_c\rfloor,w+\lceil\beta_c\rceil}, Z_c^6 = F_{c,t+\lceil\gamma_c\rceil,h+\lfloor\alpha_c\rfloor,w+\lceil\beta_c\rceil},\\
Z_c^7 &= F_{c,t+\lfloor\gamma_c\rfloor,h+\lceil\alpha_c\rceil,w+\lceil\beta_c\rceil}, Z_c^8 = F_{c,t+\lceil\gamma_c\rceil,h+\lceil\alpha_c\rceil,w+\lceil\beta_c\rceil}.
\end{aligned}$$

with $\lfloor\cdot\rfloor$, $\lceil\cdot\rceil$ as denoting floor and ceiling functions, respectively. Z_c^* correspond to the eight nearest integer points around the local neighbourhood at the location of each shift parameter. These eight points consist of a 2^3 cube and are used for trilinear interpolation (evaluated locally in an efficient manner).

We can also show why Equation 6 holds by showing how each dimension contributes to the final coefficients from a bottom-up approach (composing partial terms along the way). To begin, we can look at the temporal dimension first. By the definition of linear interpolation, the interpolated point which lies between Z_c^1 and Z_c^2 is

$$T_c^1 = Z_c^1 \cdot (1 - \Delta\gamma_c) + Z_c^2 \cdot \Delta\gamma_c.$$

Similarly, the interpolated values between $\{Z_c^3, Z_c^4\}$, $\{Z_c^5, Z_c^6\}$ and $\{Z_c^7, Z_c^8\}$ are:

$$\begin{aligned}
T_c^2 &= Z_c^3 \cdot (1 - \Delta\gamma_c) + Z_c^4 \cdot \Delta\gamma_c,\\
T_c^3 &= Z_c^5 \cdot (1 - \Delta\gamma_c) + Z_c^6 \cdot \Delta\gamma_c,\\
T_c^4 &= Z_c^7 \cdot (1 - \Delta\gamma_c) + Z_c^8 \cdot \Delta\gamma_c \text{ , respectively.}
\end{aligned}$$

Now that we have accounted for the temporal dimension contributions, we can account for the contributions from the vertical dimension to our intermediate values T_c^*. We have the interpolated value in between T_c^1 and T_c^2 as

$$H_c^1 = T_c^1 \cdot (1 - \Delta\alpha_c) + T_c^2 \cdot \Delta\alpha_c,$$

and the interpolated point in between T_c^3 and T_c^4 as

$$H_c^2 = T_c^3 \cdot (1 - \Delta\alpha_c) + T_c^4 \cdot \Delta\alpha_c.$$

Finally, we account for the linear interpolation for the horizontal dimension, and get the interpolated contribution between H_c^1 and H_c^2 as

$$W_c^1 = H_c^1 \cdot (1 - \Delta\beta_c) + H_c^2 \cdot \Delta\beta_c.$$

where W_c^1 is the final trilinearly interpolated value. If we expand and write it using Z_c^*, it has the same form as $O'_{c,t,h,w}$, recovering the product coefficients described Equation 6.

Finally, we reiterate the point from the main paper that this equation is a formalism; in practice we are able to implement the whole operation in CUDA/C++ efficiently with minimal FLOPs overhead (see efficiency analysis in Sec. A4). We also emphasize that with the budget-constrained attention shift (RubiksShift-AQ), we can replace some or all of these dimensions (e.g. temporal) with a discrete integer shift (described in Sec. 4 in the main paper), in which case any remaining dimensions are interpolated with the lower-dimensional versions of Equation 6.

A4 Efficiency Analysis Details

In this section, we provide additional details and analysis of efficiency of our models, breaking down the contribution of our 3D RubikShift block from an operations perspective with respect to traditional 3D Convolution as well as the recent 2D Convolution + Shift (TSM) block from ICCV 2019. We also provide **runtime analysis** of our method.

FLOPs and Parameters Protocol. Our FLOP and parameter computation procedure aligns with prior work [16] for consistency. In a RubiksShift layer, the main contributor to the FLOP count is the 1x1 pointwise convolution layers, which are dramatically less expensive than traditional 2D or 3D conv. The traditional shift operation itself is considered zero-FLOP, since it can be fused into the pointwise convolution as one GPU kernel call [29]. Learnable shift incurs small FLOP/param cost, but otherwise similarly efficient when properly implemented.

Additional Efficiency Analysis. We visualize a full efficiency analysis breakdown of our RubiksShift layer in Figure A2, A3, and A4. For our analysis here, we **control** the **same input** ($(T, H, W) = (8, 112, 112)$) and **input/output channels** (input and output is fixed to 72 channels for all blocks, so that channel count does not affect the analysis) for all calculations. Further, all blocks here are standard blocks with consistent channels throughout the block. Figure A2 shows the total cost comparison; we calculate that a RubiksShift layer has ∼25x fewer FLOPs/params in contrast with traditional 3D, and ∼8x fewer than TSM (shift + 2D conv). Figure A3 shows the breakdown by percentage of FLOPs, and Figure A4 shows the breakdown by percentage of parameters; each plot is shown **(a)** normalized to the 3D conv block (with a "savings" section indicating the saved relative compute) and **(b)** to itself. Similarly, these gains translate to our overall **RubiksNet** architecture, our gains are chiefly due to the replacement of *all* spatial and spatiotemporal convolutions with a learnable shift-based operation. We note that the full *RubiksNet* numbers described in the main paper (which are relatively lower, but show significant improvement) also account for all the extra layers in the full architecture (e.g. the fully-connected layers, which are not replaced by RubiksShift blocks).

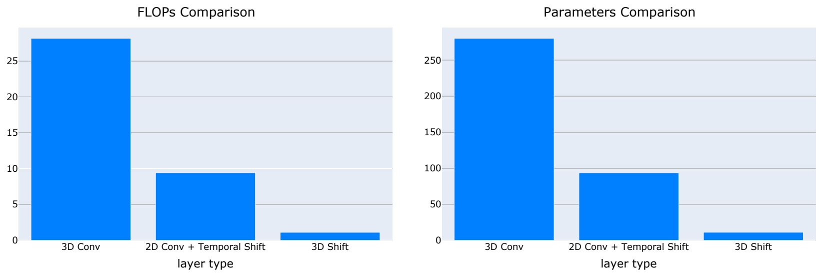


Fig. A2: Efficiency comparison at layer level. Our RubiksShift (3D learnable shift) layer shows a large efficiency gain over analogous 3D convolution and Shift+2D convolution [16] prior work. See Sec. A4.

Table A2: Runtime Latency Comparison; Block types correspond with Figure A2. See Section A4 for details.

Block Type	Runtime Latency
3D Conv	7.98ms ± 0.75ms
2D Conv + Shift (TSM) [16]	3.59ms ± 0.13ms
3D Shift (Ours)	**0.90ms ± 0.12ms**

Runtime/Latency. We also report latency analysis in the Table A2. We benchmark each layer/block type for runtime on the same GPU and hardware set-up (single GPU, Titan Xp) and averaged over 100 trials. Our input tensor in all cases is $(N, T, C, H, W) = (8, 8, 72, 56, 56)$ (batch size N is 8), and architecture blocks are similarly controlled for same input/output channels as in our other efficiency analysis breakdown. We observe that our 3D RubiksShift method has consistently better runtime than prior work. Sec. 4 in our main paper contains shift analysis at the architecture level, showing higher accuracy than prior fixed-shift [16] with consistent global shift budget.

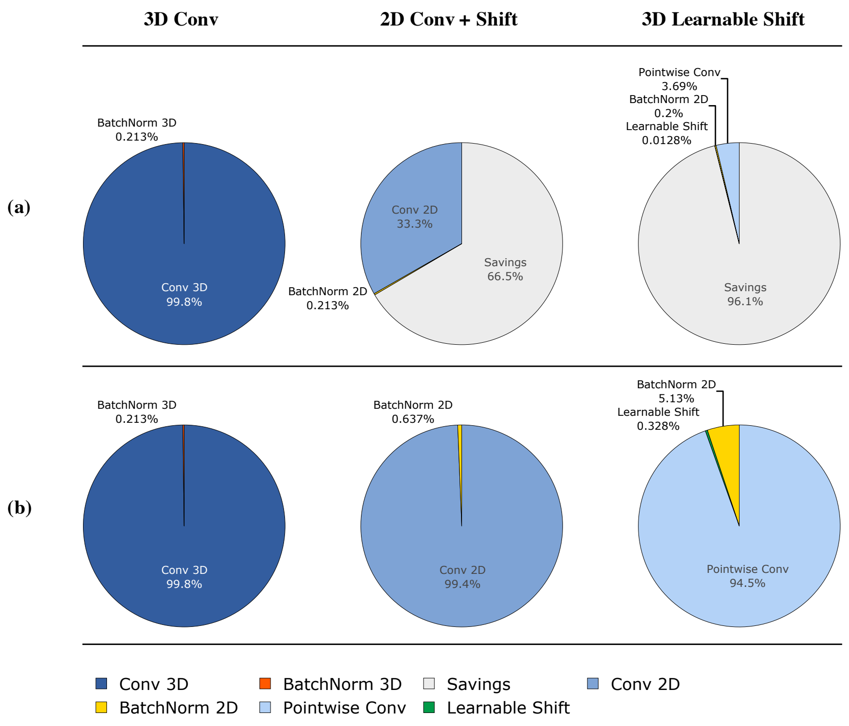


Fig. A3: Breakdown of FLOPs by percentage for RubiksShift (Learnable 3D Shift) against 3D Conv and 2D Conv+Shift [16] analogous blocks, controlling for channel/input. **(a)** shows breakdown of FLOPs for all three blocks normalized to the 3D conv block. The "savings" section indicates the saved relative compute. **(b)** is normalized to itself. See Sec. A4.

A5 Additional Results

In this section, we report additional results that we were not able to include in the main paper due to space. In particular, we provide additional results for our four main datasets in the main paper, including our larger scale (Something-Something-v2, Something-Something-v1) and smaller scale (UCF-101, HMDB-51) datasets. We also report an additional comparison against our main baseline (TSM[16]) on a fifth benchmark – the **Kinetics** dataset [14]. Note that we chose to prioritize our analysis and model training in the main paper on the two large-scale Something-Something benchmarks since both contain action classes which require more complex temporal understanding.

Additional Results. We visualize additional results for the benchmarks in the main paper in Figure A5, A6, A7. We observe that our RubiksShift model family consistently improves over the TSM [16] model family from ICCV 2019 on the efficiency-accuracy tradeoff by a significant margin across datasets.

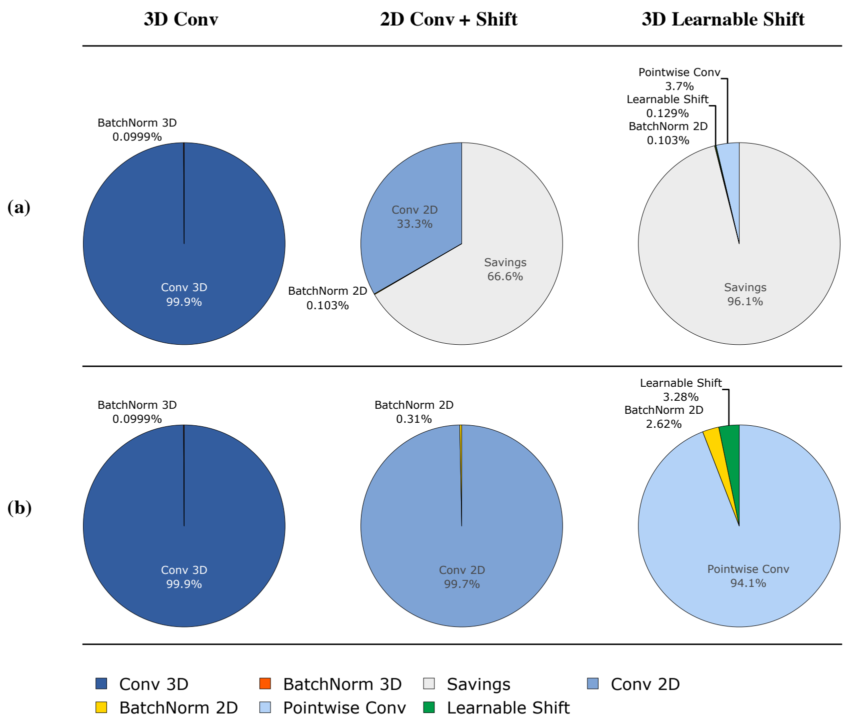


Fig. A4: Breakdown of parameters by percentage for RubiksShift (Learnable 3D Shift) against 3D Conv and 2D Conv+Shift [16] analogous blocks, controlling for channel/input. The "savings" section indicates the saved relative parameters. **(a)** shows breakdown of parameters for all three blocks normalized to the 3D conv block. **(b)** is normalized to itself. See Sec. A4.

Kinetics Comparison. We also provide additional comparison against the TSM model [16] on Kinetics [14], as shown in Figure A7. We observe that we maintain the consistent trend on this benchmark as well. We also highlight that in the high efficiency regime (e.g. TSM-Small vs. RubiksNet-Small), we substantially increase accuracy *and* efficiency by a wide margin (increase accuracy by 3.9 absolute percentage point while reducing parameters by 3.1x and FLOPs by 2.2x).

A6 Additional Training Details

Reproducibility/Code Release. Please refer to our project website for our low-level CUDA/C++ kernels as well as our higher-level PyTorch and PyTorch C++ layer and architecture code. We've also included representative pre-trained models with inference code pipeline and corresponding log files.

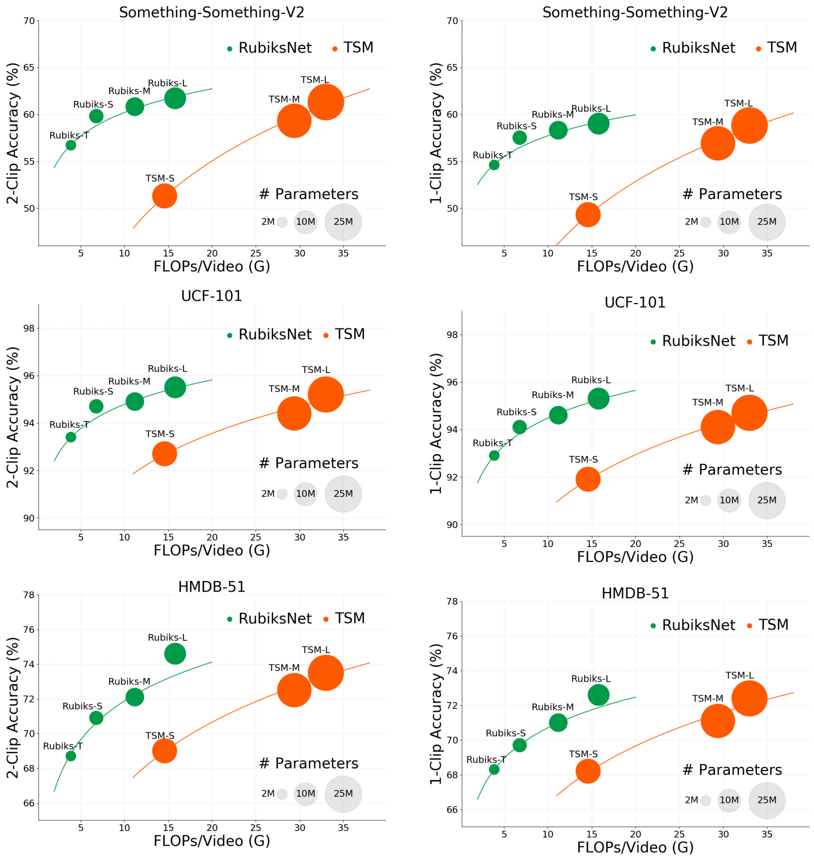


Fig. A5: We report the Pareto curves for our method compared with prior work [16], with size of the circle corresponding to the number of model parameters. Results are reported as both 2-clip accuracy (left) and 1-clip accuracy (right).

Something-Something-(V2,V1). Representative hyperparameters for our RubiksNet experiments on the Something-Something-V2 dataset in Figure 4: initial learning rate 0.025 for batch size 64, distributed across 8 GPUs. We follow the standard step-annealing scheme in the ResNet literature [6] and divide learning rate by 10 at epoch 26 and 36. We apply dropout 0.3 only to the fully connected layer. Weight decay is set to 10^{-4}. After a warm-up period, the learning rate for RubiksShift layers is set to a 1:100 ratio of the global learning rate for stability. Gradient scaling factor Z in Eq. 9 is 0.1.

UCF101 and HMDB51. The set of tuneable hyperparameters on both of these datasets is the same as with Something-Something, with similar values to the representative set above; we follow pre-training protocol from prior work [16] on Kinetics before fine-tuning on both of these datasets.

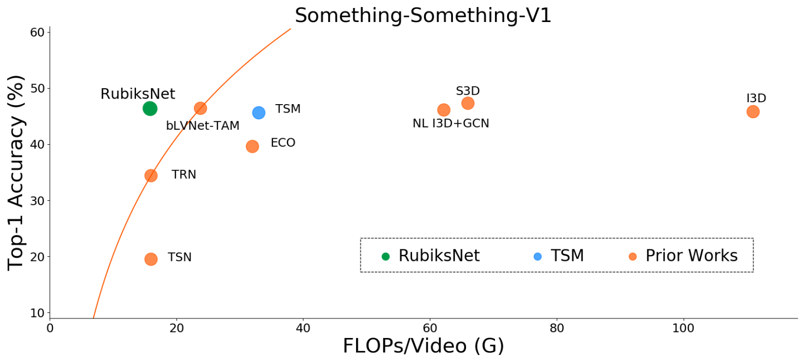


Fig. A6: We also show efficiency-accuracy comparison for Something-Something-V1 (1-clip, top-1 accuracy as per prior work); for visual clarity among multiple prior works we show analogous prior work models with a single point. TSM [16] is our main prior work comparison from ICCV 2019, and is shown in blue.

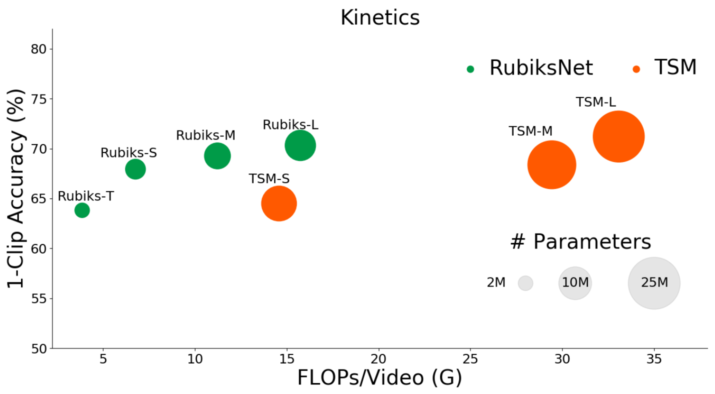


Fig. A7: We also report the Pareto curves for our method compared with prior work [16], with size of the circle corresponding to the number of model parameters. Results are reported as 1-clip, top-1 accuracy. We highlight that in the high-efficiency regime, Rubiks-Small is able to show large efficiency *and* accuracy gains over its counterpart TSM-Small.

Kinetics. The Kinetics dataset contains 306k video clips and 400 action classes. Please see Additional Results in our supplement (Section A5) for the results. For training, we adopt similar protocols to our other two large-scale datasets (Something V1 and V2) above (e.g. learning rate schedule, regularization, and RubiksShift training) and provide consistent comparison with respect to the TSM baseline [16].