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

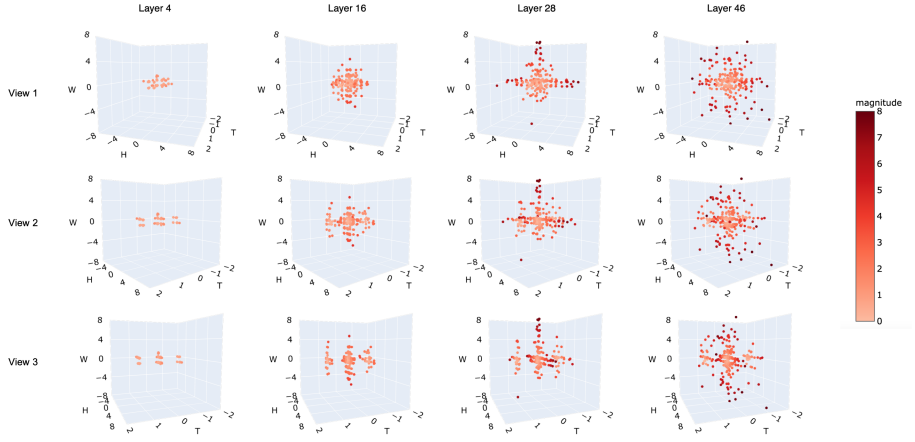

Fig. A1: Expanded visualization of Figure 6 in the main paper, showing different 3D viewpoints of the visualized spatiotemporal RubiksShift. We include further visualizations in our supplementary video.

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

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

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:

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

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).

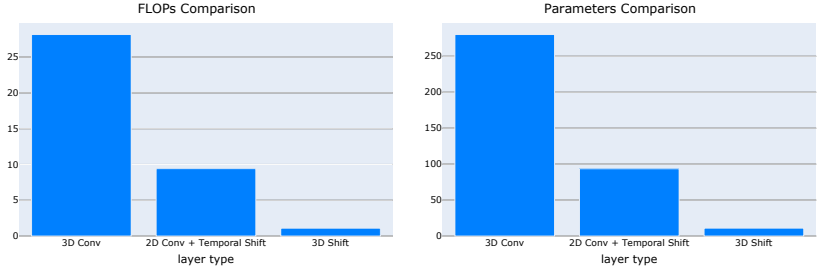

Fig. A2: Efficiency comparison at layer level. Our RubiksShift (3D learnable shift) layer shows a large efficiency gain over analogous 3D convolution and Shift+2D convolution [16] prior work. See Sec. A4.

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.

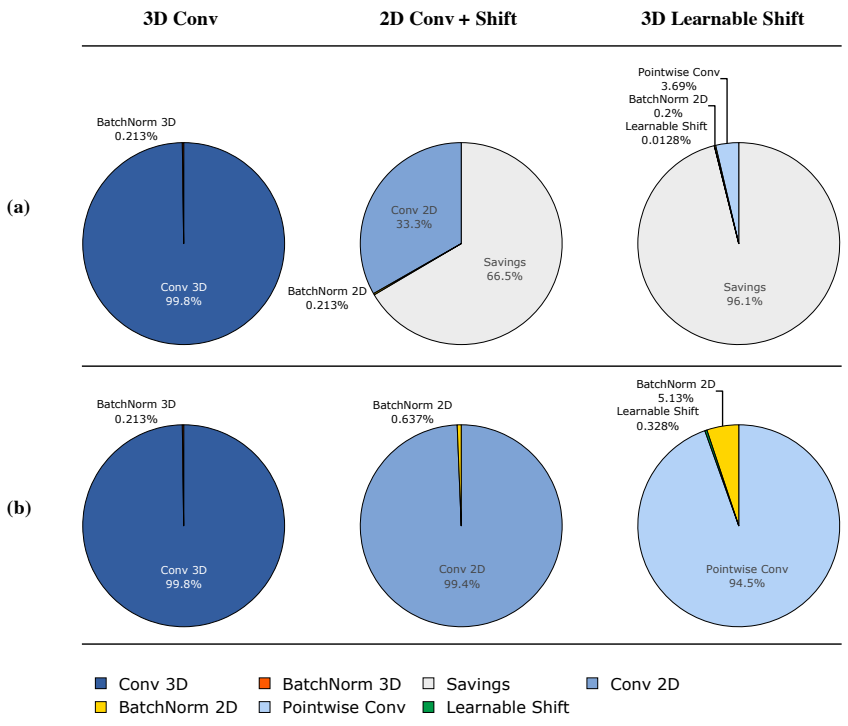

Fig. A3: Breakdown of FLOPs by percentage for RubiksShift (Learnable 3D Shift) against 3D Conv and 2D Conv+Shift [16] analogous blocks, controlling for channel/input. **(a)** shows breakdown of FLOPs for all three blocks normalized to the 3D conv block. The "savings" section indicates the saved relative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

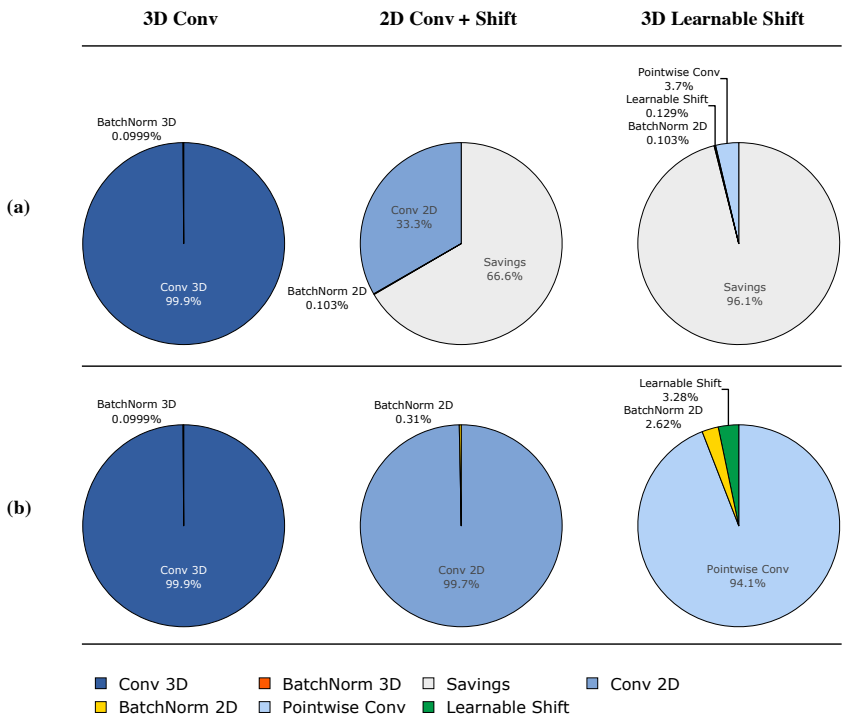

Fig. A4: Breakdown of parameters by percentage for RubiksShift (Learnable 3D Shift) against 3D Conv and 2D Conv+Shift [16] analogous blocks, controlling for channel/input. The "savings" section indicates the saved relative parameters. **(a)** shows breakdown of parameters for all three blocks normalized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

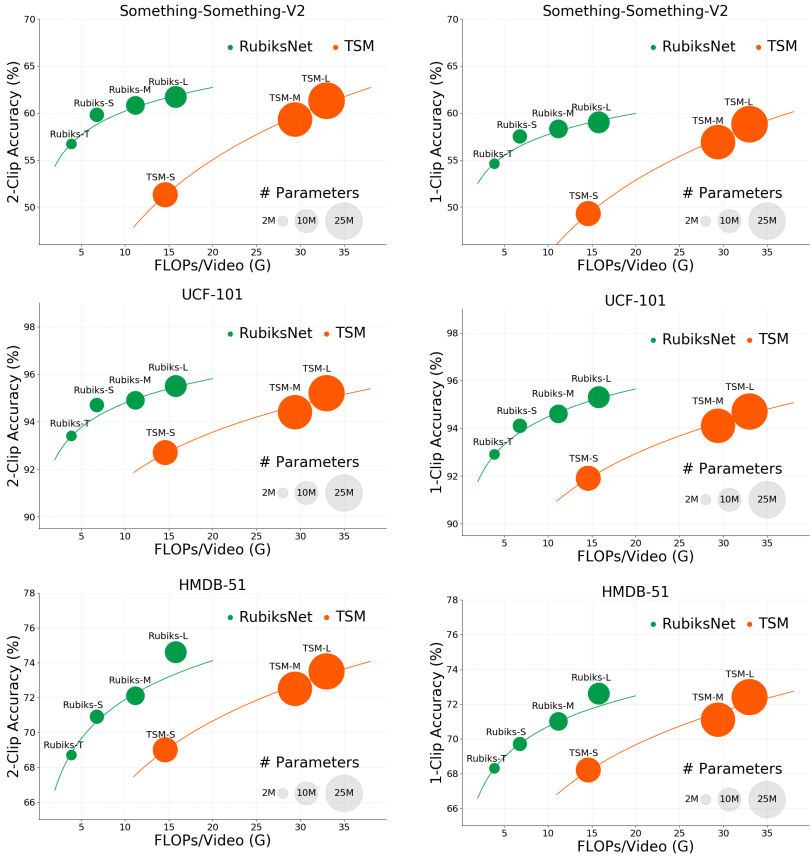

Fig. A5: We report the Pareto curves for our method compared with prior work [16], with size of the circle corresponding to the number of model parameters. Results are reported as both 2-clip accuracy (left) and 1-clip accuracy (right).

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.

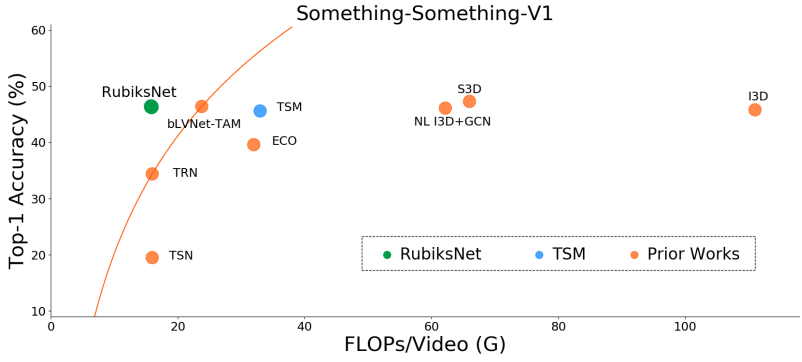

Fig. A6: We also show efficiency-accuracy comparison for Something-Something-V1 (1-clip, top-1 accuracy as per prior work); for visual clarity among multiple prior works we show analogous prior work models with a single point. TSM [16] is our main prior work comparison from ICCV 2019, and is shown in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

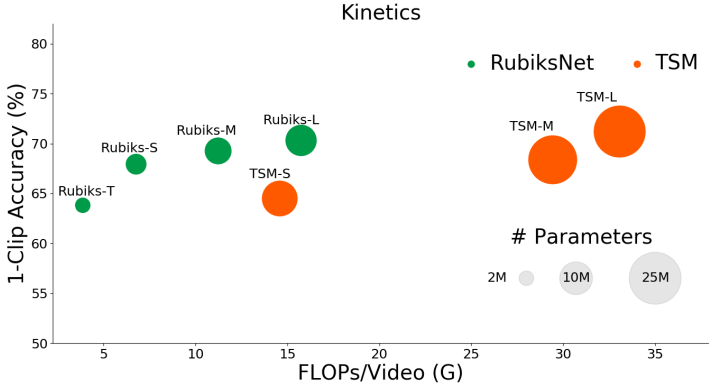

Fig. A7: We also report the Pareto curves for our method compared with prior work [16], with size of the circle corresponding to the number of model parameters. Results are reported as 1-clip, top-1 accuracy. We highlight that in the high-efficiency regime, Rubiks-Small is able to show large efficiency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