Continual 3D Convolutional Neural Networks for Real-time Processing of Videos

Lukas Hedegaard and Alexandros Iosifidis

Department of Electrical and Computer Engineering, Aarhus University, Denmark
{lhm,ai}@ece.au.dk

Appendix

A Worst-case memory for *Co*X3D-M

In this section, we provide a detailed overview of the memory consumption incurred by the internal state in a Continual X3D-M (*Co*X3D-M) model. For Continual 3D CNNs, there is no need to store input frames between time steps, though this is the case for regular 3D CNNs applied in an online processing scenario. Intermediary computations from prior frames are kept in the continual layers as state if a layer has a temporal receptive field larger than 1. A continual $k_T \times k_H \times k_W = 1 \times 3 \times 3$ convolution is equivalent to a regular convolution, while a $3 \times 1 \times 1$ is not. The same principle holds for pooling layers. As a design decision, the temporal component of the average pooling of Squeeze-and-Excitation (SE) blocks is discarded. Hence, SE blocks do not incur a memory overhead or delay. Keeping the temporal pooling of the SE block would have increased memory consumption by a modest 85.050 (+1.4%). We can compute the total state overhead using Eqs. (2), (8), and (9) of the main paper by adding up the state size of each applicable layer shown in Tab. 5. An overview of the resulting computations can be found in Tab. 4. The total memory overhead for the network state is 4,771,632 floating point operations. In addition to the state memory, the worst-case transient memory must be taken into account. The largest intermediary feature-map is produced after the first convolution in conv_1 and has a size of $24 \times 112 \times 112 = 301{,}056$ floats. The total worst-case memory consumption for *Co*X3D-M (excluding models weights) is thus **5,072,688** floats.

If we were to reduce the model clip size from 16 to 4, this would result in a memory reduction of 5,184 floats (only pool_5 is affected) for a total worst-case memory of 5,067,504 floats (−0.1%). Increasing the clip size to 64 would yield an increased state memory of 20,736 floats giving a total worst-case memory of 5,093,424 floats (+0.4%).

Stage	**Layer**		**Mem.** (floats)
conv_1	conv_T	$(5-1)\times 24\times 112\times 112 =$	1,204,224
res_2	residual_1	$(3-1-1)\times 24\times 112\times 112 =$	301,056
	residual_{2-3}	$[(3-1-1)\times 24\times 56\times 56]\times 2 =$	150,528
	conv_{1-3}	$[(3-1-1)\times 54\times 56\times 56]\times 3 =$	508,032
res_3	residual_1	$(3-1-1)\times 24\times 56\times 56 =$	75,264
	residual_{2-5}	$[(3-1-1)\times 48\times 28\times 28]\times 4 =$	150,528
	conv_{1-5}	$[(3-1)\times 108\times 28\times 28]\times 5 =$	846,720
res_4	residual_1	$(3-1-1)\times 48\times 28\times 28 =$	37,632
	residual_{2-11}	$[(3-1-1)\times 96\times 14\times 14]\times 10 =$	188,160
	conv_{1-11}	$[(3-1)\times 216\times 14\times 14]\times 11 =$	931,392
res_5	residual_1	$(3-1-1)\times 96\times 14\times 14 =$	18,816
	residual_{2-3}	$[(3-1-1)\times 192\times 7\times 7]\times 6 =$	56,448
	conv_{1-3}	$[(3-1)\times 432\times 7\times 7]\times 7 =$	296,352
pool_5	-	$(16-1)\times 432 =$	6,480
Total			**4,771,632**

Table 4: ***Co*X3D-M** state memory consumption by layer.

Stage	**Filters**	**Output size** $(T \times H \times W)$
input	-	$16 \times 224 \times 224$
conv_1	$1 \times 3^2, 24$ $5^* \times 1^2, 24$	$16 \times 112 \times 112$
res_2	res $\begin{bmatrix} 1 \times 1^2, 54 \\ 3 \times 3^2, 54 \\ \text{SE} \\ 1 \times 1^2, 24 \end{bmatrix} \times 3$	$16 \times 56 \times 56$
res_3	res $\begin{bmatrix} 1 \times 1^2, 108 \\ 3 \times 3^2, 108 \\ \text{SE} \\ 1 \times 1^2, 48 \end{bmatrix} \times 5$	$16 \times 28 \times 28$
res_4	res $\begin{bmatrix} 1 \times 1^2, 216 \\ 3 \times 3^2, 216 \\ \text{SE} \\ 1 \times 1^2, 96 \end{bmatrix} \times 11$	$16 \times 14 \times 14$
res_5	res $\begin{bmatrix} 1 \times 1^2, 432 \\ 3 \times 3^2, 432 \\ \text{SE} \\ 1 \times 1^2, 192 \end{bmatrix} \times 7$	$16 \times 7 \times 7$
conv_5	$1 \times 1^2, 432$	$16 \times 7 \times 7$
pool_5	16×7^2	$1 \times 1 \times 1$
fc_1	$1 \times 1^2, 2048$	$1 \times 1 \times 1$
fc_2	$1 \times 1^2, \#\text{classes}$	$1 \times 1 \times 1$

Table 5: **X3D-M model architecture**. When converted to a continual CNN, the highlighted components carry an internal state which results in a memory overhead. *Temporal kernel size in conv_1 is set to 5 as found in the official X3D source code [1].

B Benchmarking details

This section should be read in conjunction with Sec 4.3 of the main paper. To gauge the achievable on-hardware speeds of clip and frame predictions, a benchmark was performed on the following four system: A CPU core of a MacBook Pro (16-inch 2019 2.6 GHz Intel Core i7); Nvidia Jetson TX2; Nvidia Jetson Xavier; and a Nvidia RTX 2080 Ti GPU (on server with Intel XEON Gold processors). A batch size of 1 was used for testing on CPU, while the largest fitting multiple of $2^{\mathbb{N}}$ up to 64 was used for the other hardware platforms which have GPUs and lend themselves better to parallelisation. Thus, the speeds noted for GPU platforms in Tab. 1 of the main paper should not be interpreted as the number of processed clips/frames from a single (high-speed) video stream, but rather as the aggregated number of clips/frames from multiple streams using the available hardware. The exact batch size and input resolutions can be found in Tab. 6. In conducting the measurements, we assume the input data is readily available on the CPU and measure the time it takes for it to transfer from the CPU to GPU (if applicable), process, and transfer back to the CPU. A precision of 16 bits was used for the embedded platforms TX2 and Xavier, while a 32 bit precision was employed for CPU and RTX 2080 Ti. All networks were implemented and tested using PyTorch, and neither Nvidia TensorRT nor ONNX Runtime were used to speed up inference.

Model	**Input shape**	**Batch size**			
	$(T \times S^2)$	**CPU**	**TX2**	**Xavier**	**RTX**
I3D-R50	8×224^2	1	16	16	32
R(2+1)D-18_8	8×112^2	1	16	16	32
R(2+1)D-18_{16}	16×112^2	1	8	16	32
Slow-8×8-R50	8×256^2	1	8	8	8
SlowFast-8×8-R50	8×256^2	1	8	32	32
SlowFast-4×16-R50	16×256^2	1	16	32	32
X3D-L	16×312^2	1	16	32	32
X3D-M	16×224^2	1	32	64	64
X3D-S	13×160^2	1	64	64	64
X3D-XS	4×160^2	1	64	64	64
*Co*I3D	1×224^2	1	8	8	8
*Co*Slow	1×224^2	1	8	8	8
*Co*X3D-L	1×312^2	1	8	16	32
*Co*X3D-M	1×224^2	1	32	64	64
*Co*X3D-S	1×160^2	1	32	64	64

Table 6: **Benchmark model configurations**. For each model, the input shape is noted as $T \times S^2$, where T and S are the temporal and spatial input shape.

Model	FLOPs	Throughput (evaluations/s)			
		CPU	TX2	Xavier	RTX
(*Co*)I3D	5.04×	3.39×	0.95×	1.62×	1.64×
(*Co*)Slow	7.95×	7.68×	1.19×	1.44×	1.65×
(*Co*)X3D-L	15.34×	9.20×	5.21×	5.77×	5.98×
(*Co*)X3D-M	15.06×	9.05×	4.79×	4.95×	6.86×
(*Co*)X3D-S	12.11×	5.91×	4.15×	4.98×	3.41×

Table 7: **Relative improvements** in frame-by-frame inference in Continual 3D CNN relative to regular 3D CNN counterparts. The improvements (× lower for FLOPs and × higher for throughput) correspond to the results in Tab. 1 of the main paper.

C A note on RCU FLOPs

In Tab. 1 of the main paper, we have approximated the FLOPs for RCU [2] as follows: We use a different measure of FLOPs (the one from the `ptflops` [3]) than the RCU authors and therefore employ a translation factor of 28.6/41.0, which is our measured FLOPs for I3D (28.6) divided by theirs (41.0), multiplied with their reported 54.0 for RCU. Considering that their method used 8 frames and can be applied per frame, we also divide by 8. Note that the this approximation lacks the repeat classification layer and may thus be considered on the low side. The resulting computation becomes $28.6/41.0 \cdot 54.0/8 = 4.71$.

D Supplemental visualisations of benchmark

As a supplement to the results presented in the main paper, this appendix supplies additional views of the benchmarking results in Tab. 1. Accordingly, graphical representations of the accuracy versus speed trade-offs from Tab. 1 are shown in Figures 6-9. As in Fig. 1 of the main paper, the noted accuracies on Kinetics-400 were achieved using 1-clip/frame testing on publicly available pretrained models, the *Co*X3D models utilised X3D weights without further fine-tuning, and the numbers noted in each point represent the size of the global average pooling layer. Likewise, Tab. 7 shows the improvements in continual inference relative to the regular models. In general, the FLOPs improvements are higher than on-hardware speed evaluations, with relatively lower improvements on hardware platforms with GPUs. We attribute these differences to a memory operations overhead, which does not enjoy the same computational improvement as multiply-accumulate operations do on massively parallel hardware.

From Figures 6-9 we likewise observe, that the I3D, R(2+1)D and SlowFast models perform relatively better on hardware compared to the X3D and *Co*X3D models, which utilise computation-saving approaches such as 1D-convolutions and grouped 3D-convolutions at the price of increasing memory access cost.

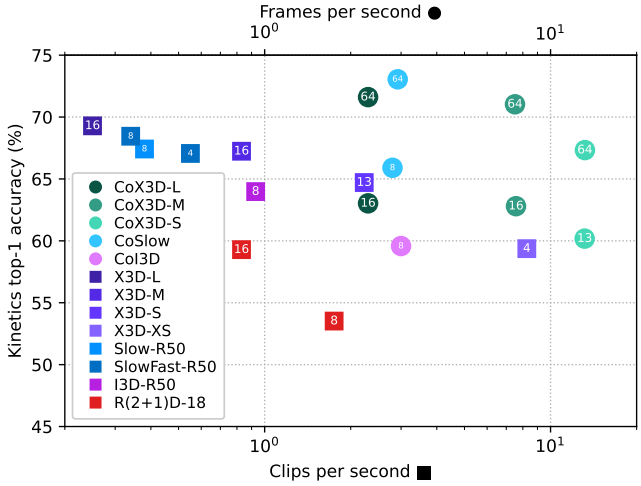

Fig. 6: **CPU** inference throughput versus top-1 accuracy on Kinetics-400.

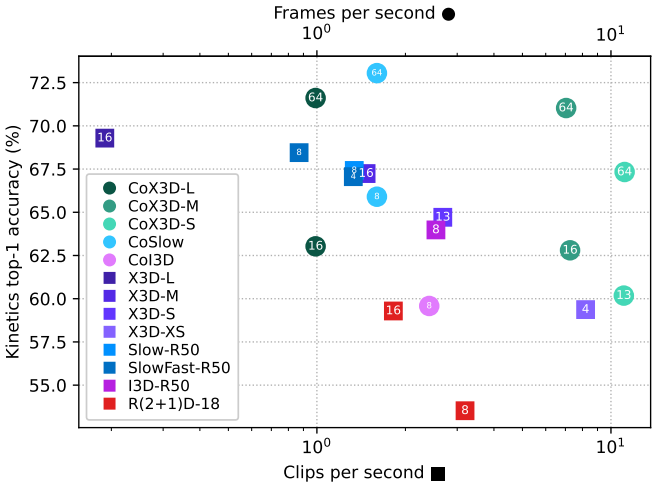

Fig. 7: **TX2** inference throughput versus top-1 accuracy on Kinetics-400.

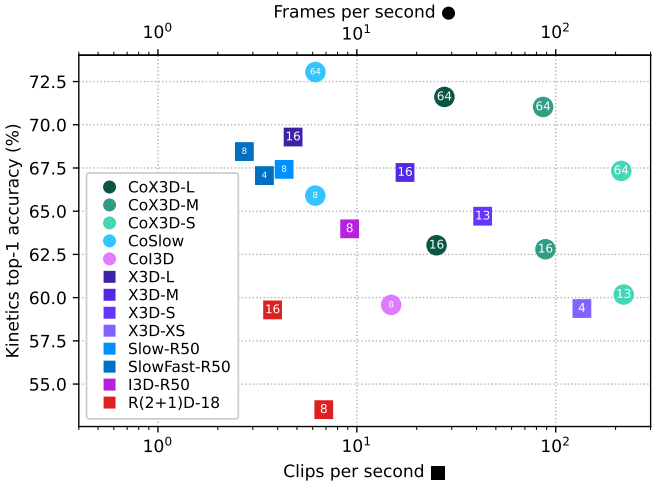

Fig. 8: **Xavier** inference throughput versus top-1 accuracy on Kinetics-400.

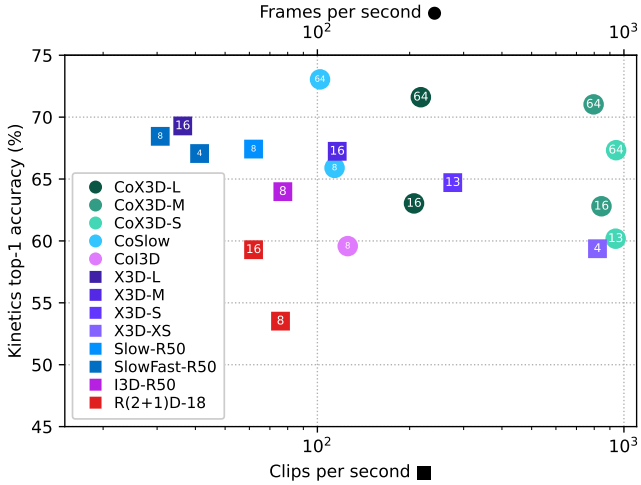

Fig. 9: **RTX2080Ti** inference throughput versus top-1 acc. on Kinetics-400.

References

1. Feichtenhofer, C.: X3D: Expanding architectures for efficient video recognition. IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) (2020), `https://github.com/facebookresearch/SlowFast`. Apache 2.0 Licence.
2. Singh, G., Cuzzolin, F.: Recurrent convolutions for causal 3d cnns. In: 2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW). pp. 1456–1465 (2019)
3. Sovrasov, V.: Ptflops, `https://github.com/sovrasov/flops-counter.pytorch`. MIT License. Last visited on 2021/03/02