Paper ID:15269

Activation Map Compression through Tensor Decomposition for Deep Learning

Le-Trung Nguyen Aël Quélennec Enzo Tartaglione Samuel Tardieu Van-Tam Nguyen

LTCI, Télécom Paris, Institut Polytechnique de Paris

{name.surname}@telecom-paris.fr

How to save up to 98.4% activation memory at training time with tensor decomposition

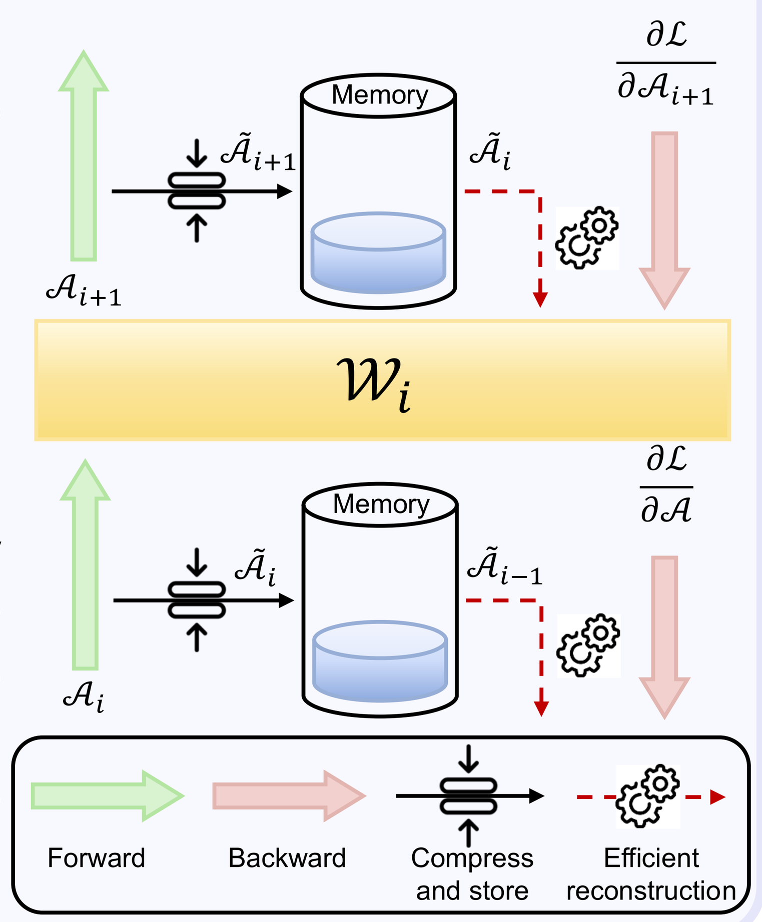
The Memory Bottleneck of Backpropagation

Considering a convolutional neural network, during the backward pass at the i^{th} layer, the following two values must be calculated:

$$\frac{\partial \mathcal{L}}{\partial \mathcal{W}_i} = \frac{\partial \mathcal{L}}{\partial \mathcal{A}_{i+1}} \cdot \frac{\partial \mathcal{A}_{i+1}}{\mathcal{W}_i} = \text{conv}\left(\mathcal{A}_i, \frac{\partial \mathcal{L}}{\partial \mathcal{A}_{i+1}}\right)$$

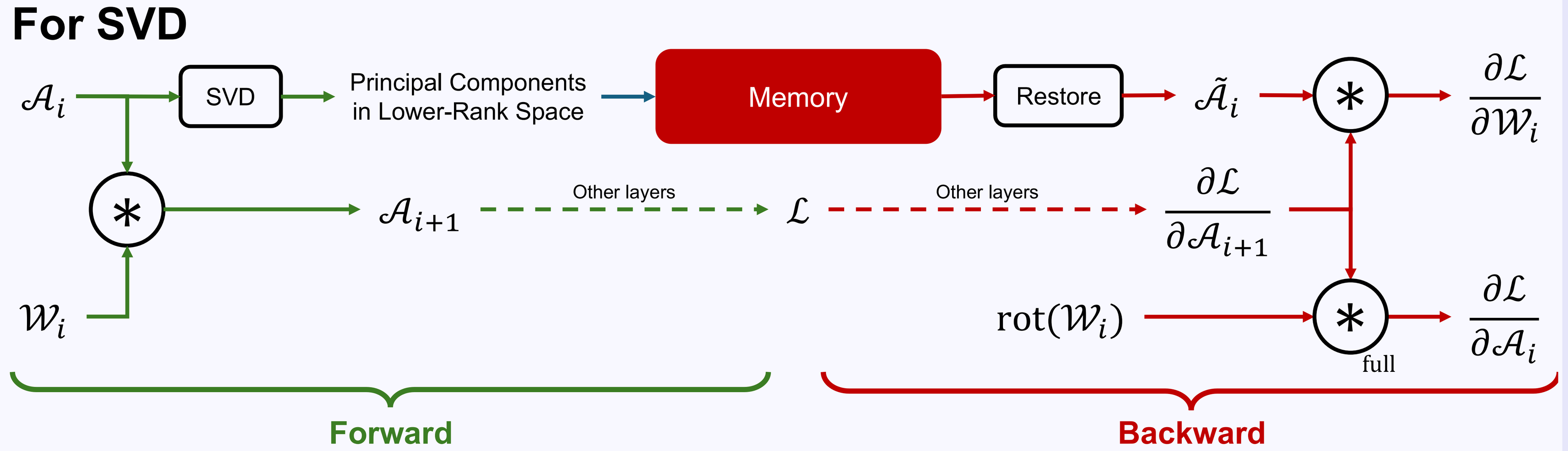
$$\frac{\partial \mathcal{L}}{\partial \mathcal{A}_i} = \frac{\partial \mathcal{L}}{\partial \mathcal{A}_{i+1}} \cdot \frac{\partial \mathcal{A}_{i+1}}{\partial \mathcal{A}_i} = \text{conv}_{\text{full}}\left[\frac{\partial \mathcal{L}}{\partial \mathcal{A}_{i+1}}, \text{rot}(\mathcal{W}_i)\right]$$

Storing $\mathcal{A}_i$ and $\mathcal{W}_i$ is the main cause of memory occupancy during backpropagation. In this work, we propose decomposing $\mathcal{A}_i$ during the forward pass using SVD and HOSVD, based on the desired amount of information to retain (noted ε). $\mathcal{W}_i$ is untouched to avoid error propagation through $\frac{\partial \mathcal{L}}{\partial \mathcal{A}_i}$.

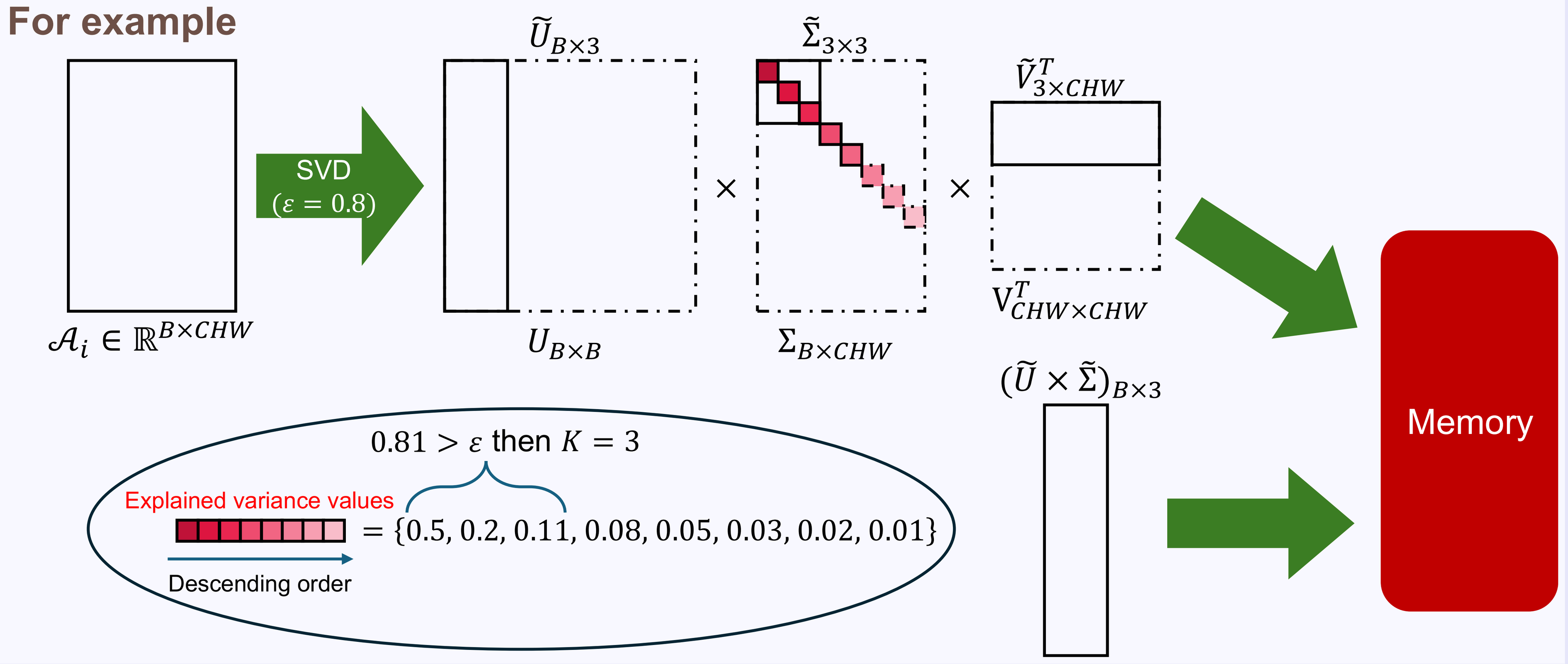


Method

For SVD



For example

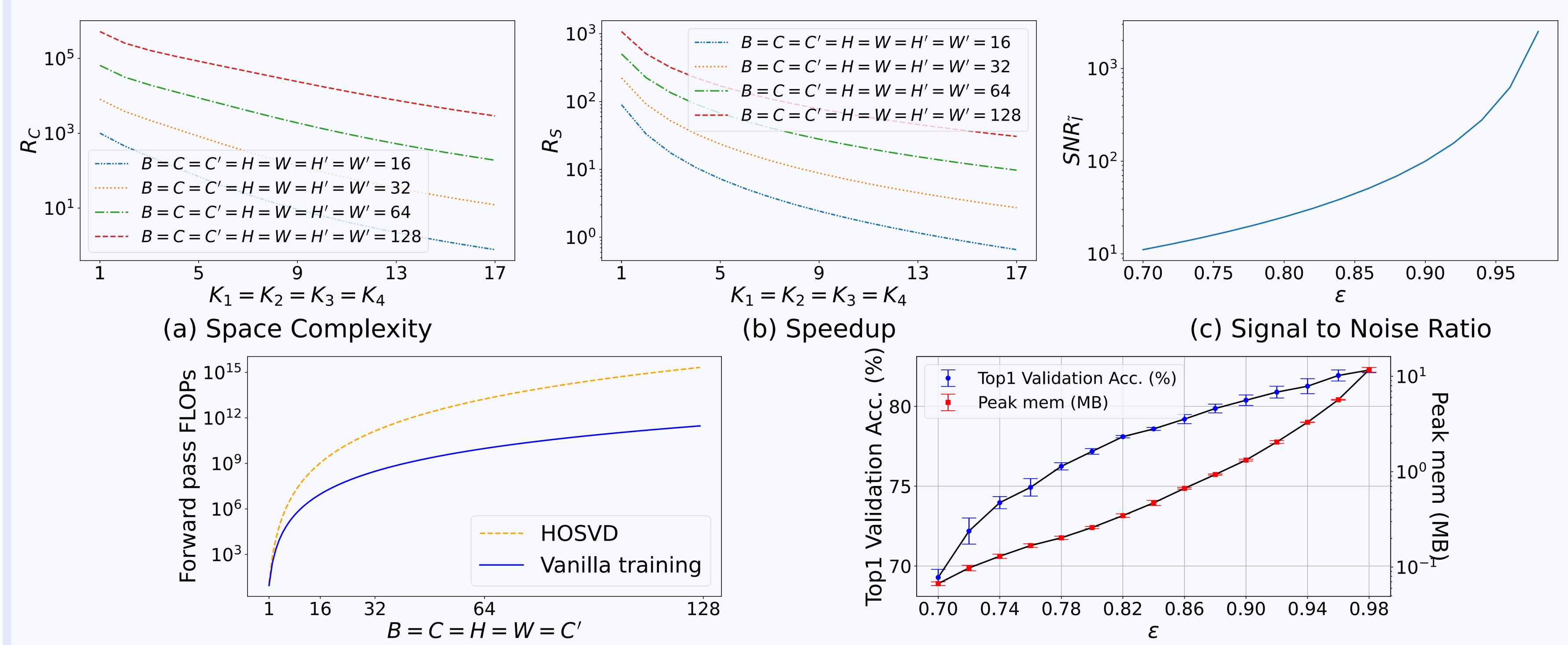


For HOSVD

- Performing SVD on all 4 modes of the activation map.
- Replacing "Restore" step and Frobenius Inner Product by 5 convolutions in lower-rank space.

Projected and Experimental Results

Projected results



(a) Space Complexity (b) Speedup (c) Signal to Noise Ratio

(d) Overhead FLOPs (e) Peak memory and Accuracy Vs. ε

- (a), (b), and (c) are theoretical predicted value for a single convolutional layer with one minibatch of data with size B.
- (d) is the result when applying HOSVD with different ε to finetune the last four layers of MCUNet.

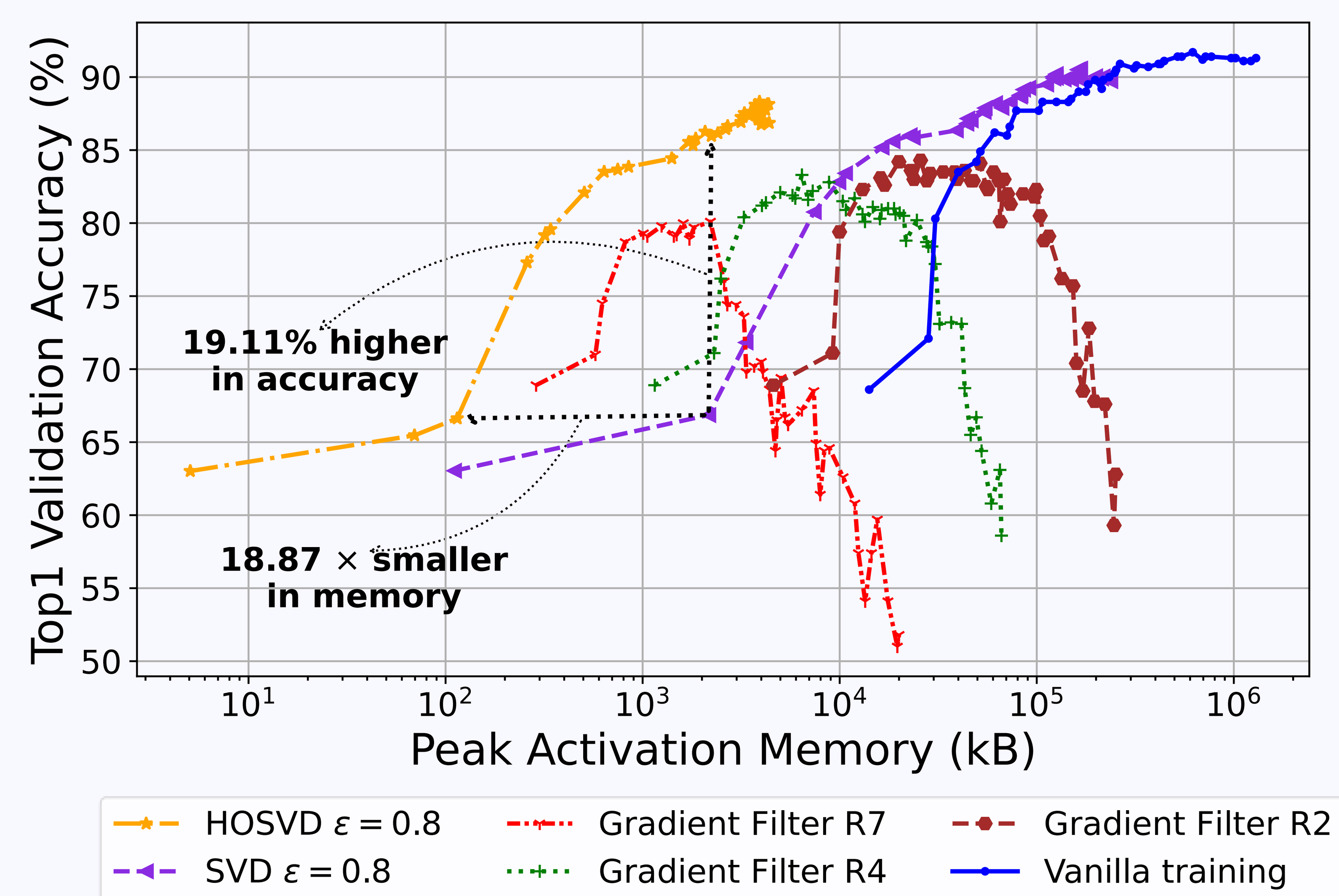
Experiment on ImageNet classification with different models

Method	MobileNetV2 #Layers	Acc ↑	Peak Mem (MB) ↓	Mean Mem (MB) ↓	Method	ResNet18 #Layers	Acc ↑	Peak Mem (MB) ↓	Mean Mem (MB) ↓
Vanilla training	All	74.0	1651.84	1651.84 ± 0.00	Vanilla training	All	72.8	532.88	532.88 ± 0.00
	2	62.6	15.31	15.31 ± 0.00		2	69.9	12.25	12.25 ± 0.00
	4	65.8	28.71	28.71 ± 0.00		4	71.5	30.63	30.63 ± 0.00
Gradient Filter R2	2	62.6	5.00	5.00 ± 0.00	Gradient Filter R2	2	68.7	4.00	4.00 ± 0.00
	4	65.2	9.38	9.38 ± 0.00		4	69.3	7.00	7.00 ± 0.00
SVD ($\varepsilon = 0.8$)	2	61.7	4.97	4.92 ± 0.08	SVD ($\varepsilon = 0.8$)	2	69.5	7.88	7.71 ± 0.21
	4	65.2	14.76	14.61 ± 0.09		4	71.1	19.98	19.72 ± 0.28
SVD ($\varepsilon = 0.9$)	2	62.3	8.97	8.91 ± 0.08	SVD ($\varepsilon = 0.9$)	2	69.7	9.86	9.77 ± 0.13
	4	65.5	20.35	20.20 ± 0.07		4	71.3	24.81	24.66 ± 0.17
HOSVD ($\varepsilon = 0.8$)	2	61.1	0.15	0.15 ± 0.00	HOSVD ($\varepsilon = 0.8$)	2	69.2	0.97	0.91 ± 0.05
	4	63.9	0.73	0.68 ± 0.03		4	70.5	2.89	2.74 ± 0.12
HOSVD ($\varepsilon = 0.9$)	2	61.8	0.43	0.43 ± 0.01	HOSVD ($\varepsilon = 0.9$)	2	69.5	2.73	2.63 ± 0.10
	4	64.8	1.92	1.76 ± 0.08		4	71.1	7.96	7.66 ± 0.21

Method	MCUNet #Layers	Acc ↑	Peak Mem (MB) ↓	Mean Mem (MB) ↓	Method	ResNet34 #Layers	Acc ↑	Peak Mem (MB) ↓	Mean Mem (MB) ↓
Vanilla training	All	67.4	632.98	632.98 ± 0.00	Vanilla training	All	75.6	839.04	839.04 ± 0.00
	2	62.1	13.78	13.78 ± 0.00		2	69.6	12.25	12.25 ± 0.00
	4	64.7	19.52	19.52 ± 0.00		4	72.2	24.50	24.50 ± 0.00
Gradient Filter R2	2	61.8	4.50	4.50 ± 0.00	Gradient Filter R2	2	68.8	4.00	4.00 ± 0.00
	4	64.4	6.38	6.38 ± 0.00		4	70.9	8.00	8.00 ± 0.00
SVD ($\varepsilon = 0.8$)	2	62.0	7.62	7.51 ± 0.12	SVD ($\varepsilon = 0.8$)	2	69.2	6.70	6.49 ± 0.29
	4	64.5	10.59	10.37 ± 0.20		4	71.8	14.68	14.24 ± 0.50
SVD ($\varepsilon = 0.9$)	2	62.1	10.32	10.26 ± 0.08	SVD ($\varepsilon = 0.9$)	2	69.4	9.10	8.96 ± 0.23
	4	64.6	14.39	14.26 ± 0.13		4	72.0	19.11	18.83 ± 0.37
HOSVD ($\varepsilon = 0.8$)	2	61.7	0.48	0.43 ± 0.04	HOSVD ($\varepsilon = 0.8$)	2	68.7	0.30	0.27 ± 0.02
	4	63.9	0.88	0.78 ± 0.07		4	71.1	1.11	1.02 ± 0.07
HOSVD ($\varepsilon = 0.9$)	2	62.0	1.32	1.27 ± 0.06	HOSVD ($\varepsilon = 0.9$)	2	69.2	0.71	0.65 ± 0.05
	4	64.4	2.52	2.36 ± 0.15		4	71.9	3.24	3.09 ± 0.13

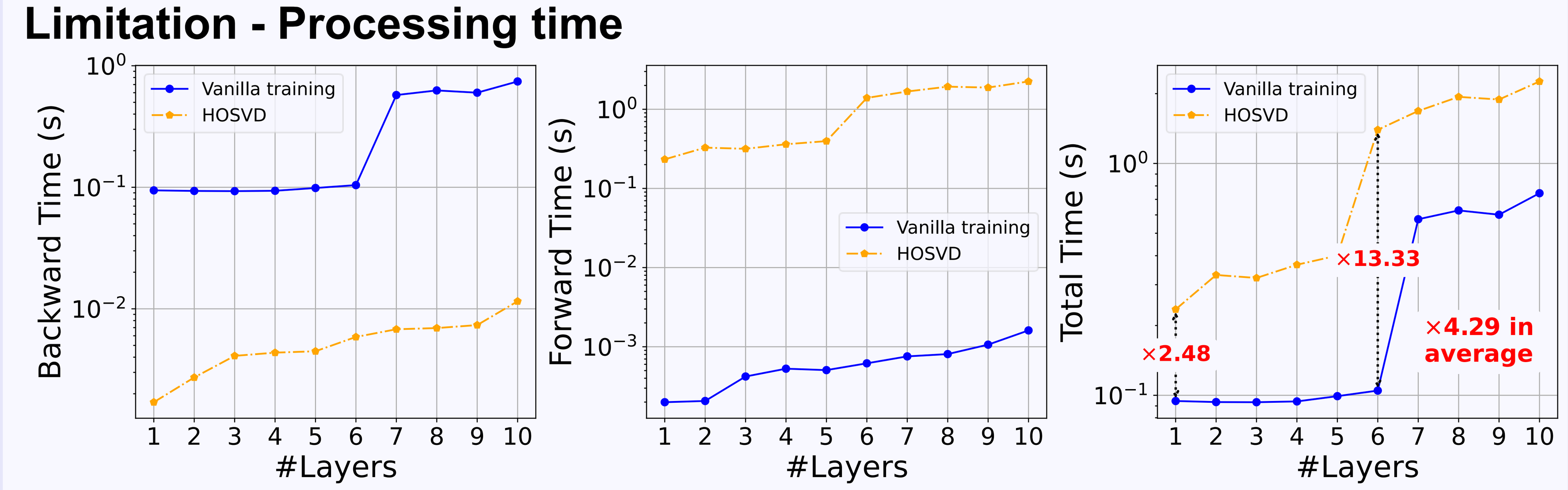
For the same #Layers, our techniques save significantly more memory than the others

Fine-tuning different number of layers of MCUNet pretrained on ImageNet with CIFAR-10 as downstream dataset



- Fine-tuning all layers with HOSVD costs less memory than fine-tuning only one layer with vanilla training.
- HOSVD and SVD do not accumulate error contrary to Gradient Filter.
- In a same context, HOSVD performs better than SVD.

Limitation - Processing time



(a) Backward (b) Forward (c) total

- Backward processing of HOSVD is tens of times faster than vanilla training, while forward is up to a thousand times slower.
- HOSVD is on average 4.29 times slower than vanilla training overall.

References

Yang, Y., Li, G., & Marculescu, R. (2023). Efficient on-device training via gradient filtering. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (pp. 3811-3820).

Lin, J., Zhu, L., Chen, W. M., Wang, W. C., Gan, C., & Han, S. (2022). On-device training under 256kb memory. Advances in Neural Information Processing Systems, 35, 22941-22954.

Acknowledgement: Part of this work was funded by Hi!PARIS Center on Data Analytics and Artificial Intelligence, by the European Union's HORIZON Research and Innovation Programme under grant agreement No 101120657, project ENFIELD (European Lighthouse to Manifest Trustworthy and Green AI) and by French National Research Agency (ANR-22-PEFT-0003 and ANR-22-PEFT-0007) as part of France 2030, the NF-NAI project and NF-FITNESS project.

Paper Code