XiNets: Efficient Neural Networks for tinyML Supplementary Material

1. Benchmarking details

1.1. Training Procedure

On CIFAR-100, we trained XiNet for 150 epochs using the LAMB optimizer [5] with a learning rate of 0.004. As augmentation strategies, we used mixup ($p = 0.05$) [7], cutmix ($p = 0.05$) [6], and auto-augment ($M = 5 \pm 0.55$) [2]. Training scripts are available in the supplementary materials, in the folder `classification`. Refer to the `README` for instructions on how to run the experiments.

On VOC-2012 and COCO, we trained XiNet for 100 and 50 epochs respectively, using the SGD optimizer with a learning rate of 0.01. The augmentation strategy is the same as proposed in Yolov7 [4]. Training scripts and configurations used are again available in the supplementary materials, in the `object detection` folder.

1.2. Comparison with MCUNet

The main goal of our comparison with the baselines is to highlight the benefits of the neural architecture's energy efficiency and computational resources. Therefore, we want to isolate the training procedure as much as possible and focus on the architectural design only, thus, using the same experimental protocol for all networks. Moreover, applying a novel training strategy to all the baselines should lead to similar relative improvements in performance, thus not compromising the conclusions drawn in the paper. The training procedure for MCUNet requires multiple stages. As the code for this training strategy is unavailable, and we want to avoid incorrectly representing results due to unnoticed reproducibility errors, we trained all the baselines from scratch after carefully optimizing the training hyper-parameters (Table 1 shows the used search space) using the Orìon toolkit [1].

1.3. Dataset choice

To benchmark XiNet on image classification, we chose to use CIFAR-100 due to its suitability for highlighting the backbone's capabilities in low-resource regimes. While training mobile-oriented networks on ImageNet could improve their performance on other benchmarks through transfer learning, we opted for CIFAR-100 to ensure that the classification head did not overshadow the backbone's modelling power. Imagenet would require a significant portion of the network to map embeddings to its large number of classes, limiting the potential of the whole pipeline to fit into a tiny hardware platform without recurring to specialized neural architecture search strategies or training loops. To further validate XiNet's image processing design, we benchmarked its performance on the VOC-2012 and MS-COCO benchmarks.

2. Attention mechanism ablation study

XiNets leverages an attention mechanism that combines channel and spatial attention while minimizing complex convolutional operations. Specifically, we adapt the mixed attention module to generate a feature map with identical dimensions (width, height, and output channels $W_{out} \times H_{out} \times C_{out}$) as the output feature map of a block. Thus, it is possible to perform an element-wise multiplication between feature maps, avoiding computationally expensive tensor-vector and tensor-matrix multiplication operations that may not be optimized on embedded devices.

We benchmarked different ways to generate the attention map to find the best trade-off between computational cost and performance. Following are the different techniques and an analysis of their structures:

- The most naive implementation for this kind of mixed attention relies on a single 2D convolution from $W_{out} \times H_{out} \times C_{out}$ to the same dimensionality, followed by a softmax activation and an element-wise product with the generating tensor, as depicted in Fig. 1.

CutMix	MixUp	Weight decay	Learning rate	Optimizer
uniform(0, 0.2)	uniform(0, 0.2)	loguniform(0.01, 0.001)	loguniform(0.1, 0001)	lamb, sgd, adam

Table 1. Search space used for the different hyperameters optimized using Bayesian optimization. Notation follows the one from the Orìon toolkit.

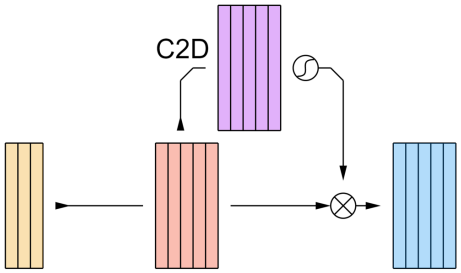

Figure 1. Naive attention module implementation

This approach approximates channel attention with a single convolution (as in the case of a squeeze and excitation block [3]) and spatial attention by providing a $K \times K$ higher receptive field to the convolutional block. The main drawback of this approach is the considerable parameter count resulting from using the same K as the convolutional block. It would result in

$$\frac{MAC_{att}}{MAC_{block}} = \frac{\gamma K^2}{K^2 + 1}$$

A plausible range for the compression factor γ is $[3; 6]$; therefore, this attention implementation would increase the number of operations of the block by 2.7× to 5.4×, making it non-practical in an actual network design.

- To reduce the number of operations required by the naive implementation, one potential solution is to compute the approximated attention map using only a subset of channels (e.g., C_{out}/γ) as illustrated in Figure 2.

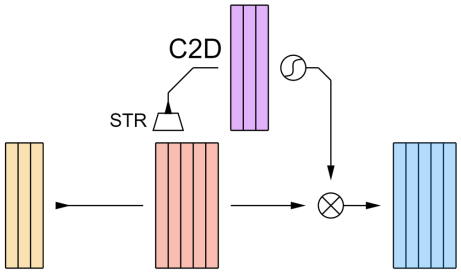

Figure 2. Stride-based attention module implementation

This approach may result in a lower increase in accuracy, as it is equivalent to computing an SE [3] tensor over a small subset of channels. Nonetheless, it would only require

$$\frac{MAC_{att}}{MAC_{block}} = \frac{K^2}{K^2 + 1}$$

While this increases the block operations of the attention module by 90%, it also introduces a potentially non-optimized or unsupported striding operation into the module.

- A pointwise convolution can be used as a substitute for the striding operation to reduce the number of channels before computing the attention map (Fig. 3). This choice has the advantage that all input channels are used in the attention module computation. Additionally, the channel attention mechanism mirrors what happens with an SE block [3], where a high number of channels gets reduced using an FC layer.

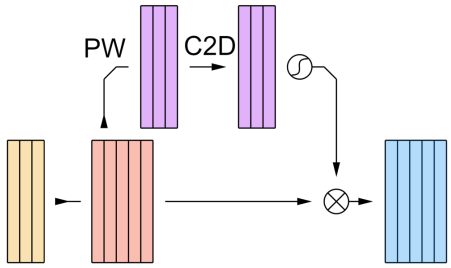

Figure 3. PW-based attention module implementation

Another advantage of this implementation is that a non-linearity can be inserted after the compression PW convolution. Operation count is only marginally increasing compared to the stride-based implementation at

$$\frac{MAC_{att}}{MAC_{block}} = \frac{W_{out} \times H_{out} \frac{C_{out}^2}{\gamma} \times (1 + K^2)}{W_{out} \times H_{out} \times \frac{C_{out}^2}{\gamma}(1 + K^2)} = 1$$

It is precisely doubling the cost of the base convolutional block. While this approach shows a slightly higher operation count than the stride-based implementation, it avoids using a striding operator, which could slow down execution depending on the target platform. Moreover, RAM

usage is significantly reduced, as all operations in the block target one tensor (input or output) of full size $W_{out} \times H_{out} \times C_{out}$, and one of compressed size $W_{out} \times H_{out} \times C_{out}/\gamma$. At a certain point, other implementations will require the presence in RAM of two full-size tensors for input and output.

- It is possible to remove the additional operations required by the previous implementation by splitting the computation of the first $W_{out} \times H_{out} \times C_{out}$ tensor into two partial tensors (Fig. 4), one with size $W_{out} \times H_{out} \times C_{out}/\gamma$, which will be used to compute the attention map, and the remaining one of size $W_{out} \times H_{out} \times C_{out} \times (\gamma - 1)/\gamma$.

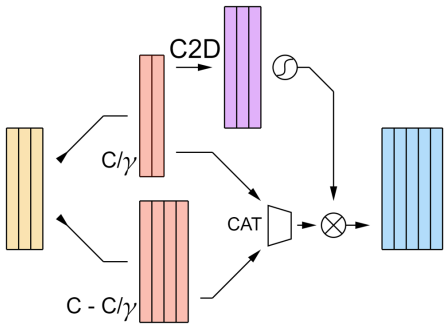

Figure 4. Concatenation-based attention module implementation

This implementation has the same number of operations as the stride-based one while trading a higher fragmentation (number of sub-tensors computed per block) for the striding process. While the number of operations is lower, this approach again suffers from the inability to consider all feature map channels in the attention map computation.

- Another approach that has been tested relies instead on computing the attention map directly on the compressed tensor in the block instead of using the output tensor (Fig 5).

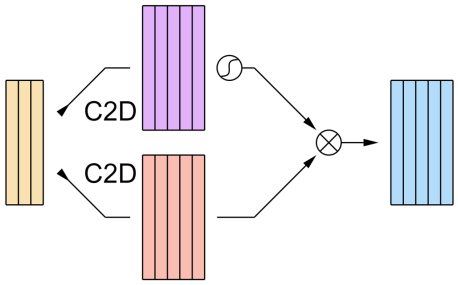

Figure 5. Compression-based attention module implementation

While this is optimal from the point of view of operation count, operation type, and network fragmentation, this approach does not provide a K^2 increase in the receptive field of the block.

We test the described attention modules with a 5-block network, first trained without any attention module, and report the relative accuracy and latency (on an `STM32F7` MCU).

Module	MAC	latency	MAC/s	Gain
None	1.8M	13.1ms	137.7M	+0%
Conv2D	11.5M	79.9ms	143.9M	+24%
Stride	3.2M	27.7ms	115.7M	+13%
Pointwise	**3.3M**	**27.0ms**	**122.4M**	**+20%**
Pointwise (relu)	3.3M	27.0ms	122.1M	-1%
Concatenation	**3.2M**	**30.7ms**	**104.4M**	**+21%**
Compression	3.2M	26.6ms	120.5M	+17%

3. Kernel size and aligned accesses

As we explained in the main paper, the efficiency of an operator is determined by several factors, including Arithmetic Intensity, the ratio of the number of operations to the number of memory accesses related to an operation. It follows that optimizing the number of memory accesses can improve efficiency. One effective way to achieve this, especially on platforms with SIMD operations for multiple byte load/store, is to use a kernel size K multiple of the number of bytes loaded by a single SIMD operation. For instance, if a platform can load 4 kernel values in uint8 with a single 32-bit load operation, then K must be a multiple of 4, e.g., 4×2.

We tested different configurations on a subset of the platforms to see if this optimization translated into an actual increase in efficiency in practice.

As we can observe in Table 3, this optimization leads to actual benefits on some platforms but does not significantly benefit others, and in some cases, the performance was even degraded. Particularly informative is the case of the Kendryte K210, which shows how some platforms and/or runtimes can be largely optimized to perform convolutions only with commonly used kernels. For these reasons, we chose not to vary the kernel size and to keep it fixed to 3×3 throughout the network to maximize compatibility with as many platforms as possible.

4. Full efficiency results for considered platforms

Tables 2 and 4 report experimental results for efficiency measures for the tested platforms. We show the number of operations per second performed by each platform, the number of operations completed per mJ of energy, and the relative efficiency obtained. The

notation "NS" in the following tables specifies that the compiler or runtime did not support the operator being tested.

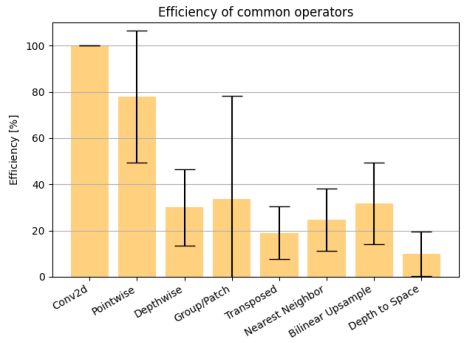

Figure 6. Average efficiency across all platforms for tested operators

	3x3 MOPS/ms	4x2 MOPS/ms	Gain
RPI4	91.45	107.95	18.04%
RPI3	35.55	43.98	23.72%
H7	2.54	2.24	-11.63%
L4	0.54	0.49	-9.80%
K210	4.55	0.26	-94.27%

Table 3. Comparison of efficiency using SIMD-optimized kernels across different edge devices.

References

[1] Xavier Bouthillier, Christos Tsirigotis, François Corneau-Tremblay, Thomas Schweizer, Lin Dong, Pierre Delaunay, Fabrice Normandin, Mirko Bronzi, Dendi Suhubdy, Reyhane Askari, Michael Noukhovitch, Chao Xue, Satya Ortiz-Gagné, Olivier Breuleux, Arnaud Bergeron, Olexa Bilaniuk, Steven Bocco, Hadrien Bertrand, Guillaume Alain, Dmitriy Serdyuk, Peter Henderson, Pascal Lamblin, and Christopher Beckham. Epistimio/orion: Asynchronous Distributed Hyperparameter Optimization, 2022. 1

[2] Ekin Dogus Cubuk, Barret Zoph, Dandelion Mané, Vijay Vasudevan, and Quoc V. Le. Autoaugment: Learning augmentation strategies from data. *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 113–123, 2019. 1

[3] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7132–7141, 2018. 2

[4] Chien-Yao Wang, Alexey Bochkovskiy, and Hong-Yuan Mark Liao. Yolov7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors, 2022. 1

[5] Yang You, Jing Li, Sashank J. Reddi, Jonathan Hseu, Sanjiv Kumar, Srinadh Bhojanapalli, Xiaodan Song, James Demmel, Kurt Keutzer, and Cho-Jui Hsieh. Large batch optimization for deep learning: Training bert in 76 minutes. *arXiv: Learning*, 2019. 1

[6] Sangdoo Yun, Dongyoon Han, Seong Joon Oh, Sanghyuk Chun, Junsuk Choe, and Young Joon Yoo. Cutmix: Regularization strategy to train strong classifiers with localizable features. *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 6022–6031, 2019. 1

[7] Hongyi Zhang, Moustapha Cissé, Yann Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. *ArXiv*, abs/1710.09412, 2017. 1

	2D Convolution			Pointwise		
Platform	**MAC/s**	**MAC/mJ**	**Eff**	**MAC/s**	**MAC/mJ**	**Eff**
L452	31.3M	1.40M	100%	29.87M	1.34M	95%
H743	238.4M	1.86M	100%	267.3M	2.08M	112%
nRF52	19.5M	2.71M	100%	19.24M	2.67M	98%
GAP8	4823M	16.55M	100%	1332M	4.57M	28%
K210	1314M	2.09M	100%	1301M	2.07M	99%
rPi 3	5732M	1.61M	100%	3514M	0.99M	61%
rPi 4	15820M	3.13M	100%	8309M	1.64M	53%
AVG:			100%			78% %
	Depthwise			**Patch / Grouped**		
Platform	**MAC/s**	**MAC/mJ**	**Eff**	**MAC/s**	**MAC/mJ**	**Eff**
L452	7.07M	0.32M	22%	5.30M	0.41M	25%
H743	41.59M	0.32M	17%	48.35M	0.38M	20%
nRF52	5.52M	0.77M	28%	4.53M	0.63M	23%
GAP8	1445M	4.96M	29%	6840M	23.47M	142%
K210	902M	1.44M	69%	152M	0.24M	12%
rPi 3	1478M	0.42M	26%	381M	0.11M	7%
rPi 4	3122M	0.62M	19%	882M	0.17M	6%
AVG:			30%			35%

Table 2. Convolutional operators efficiency

	Transposed Convolution			NN Interpolation		
Platform	**MAC/s**	**MAC/mJ**	**Eff**	**MAC/s**	**MAC/mJ**	**Eff**
L452	6.82M	0.31M	22%	6.26M	0.28M	20%
H743	62.86M	0.49M	26%	74.80M	0.58M	31%
nRF52	4.10M	0.57M	21%	4.68M	0.65M	24%
GAP8	1952M	6.70M	40%	2601M	8.92M	54%
K210	54M	0.09M	4%	269M	0.43M	20%
rPi 3	541M	0.15M	9%	524M	0.15M	9%
rPi 4	1677M	0.33M	11%	2372M	0.47M	15%
AVG:			19%			25%
	Bilinear Interpolation			**PixelShuffle/ Depth2Space**		
Platform	**MAC/s**	**MAC/mJ**	**Eff**	**MAC/s**	**MAC/mJ**	**Eff**
L452	13.46M	0.60M	43%	NS	NS	
H743	111.86M	0.87M	47%	NS	NS	
nRF52	8.58M	1.19M	44%	2.73M	0.38M	14%
GAP8	2550M	8.75M	53%	341M	4.30M	26%
K210	157M	0.25M	12%	NS	NS	
rPi 3	494M	0.14M	9%	622M	0.18M	11%
rPi 4	2203M	0.44M	14%	2804M	0.55M	18%
AVG:			30%			17%

Table 4. Upsampling operators efficiency