XiNet: Efficient Neural Networks for tinyML

Alberto Ancilotto*, Francesco Paissan*, Elisabetta Farella
Energy Efficient Embedded Digital Architectures (E3DA)
Fondazione Bruno Kessler
{aancilotto,fpaissan,efarella}@fbk.eu

Abstract

The recent interest in the edge-to-cloud continuum paradigm has emphasized the need for simple and scalable architectures to deliver optimal performance on computationally constrained devices. However, resource-efficient neural networks usually optimize for parameter count and thus use operators such as depthwise convolutions, which do not maximally exploit the efficiency of resource-constrained devices. In this article, we propose XiNet, a novel convolutional neural architecture that targets edge devices. We derived the XiNet architecture from an extensive real-world efficiency analysis of various neural network operators (e.g., standard, depthwise, and pointwise convolutions). Compared to other mobile architectures, our approach substantially improves the performance-complexity trade-off by optimizing the number of operations, parameters, and working memory (RAM). Moreover, we show how XiNet can be easily adapted to different devices thanks to Hardware Aware Scaling (HAS), which enables disjoint optimization of RAM, FLASH, and operations count. We analyze the scaling properties of our architecture under different hardware constraints and validate it on the image classification task. Finally, we evaluate the performance of XiNet for object detection on the MS-COCO and VOC-2012 benchmarks and compare it with state-of-the-art mobile neural networks, achieving a 70% reduction in energy requirements with similar performance.

1. Introduction

Designing and scaling convolutional neural networks (ConvNets) for mobile and embedded inference is challenging. Approaches like Xception [1], MobileNetv2 [16], EfficientNet [17], and ShuffleNet [6] aim at optimizing the trade-off between computational cost and performance. The design of these architectures relies on the common assumption that indirect metrics (usually the number of multiply-accumulate operations - MAC) can accurately express the overall efficiency of the ConvNet when deployed on a mobile device. However, this is usually not the case, as demonstrated in many works [16, 18] that analyze the practical efficiency of different operators when deployed on hardware platforms. Factors such as memory accesses, Arithmetic Intensity [4], and RAM usage are crucial in the efficiency of the inference and thus should be considered in the development of resource-efficient ConvNets.

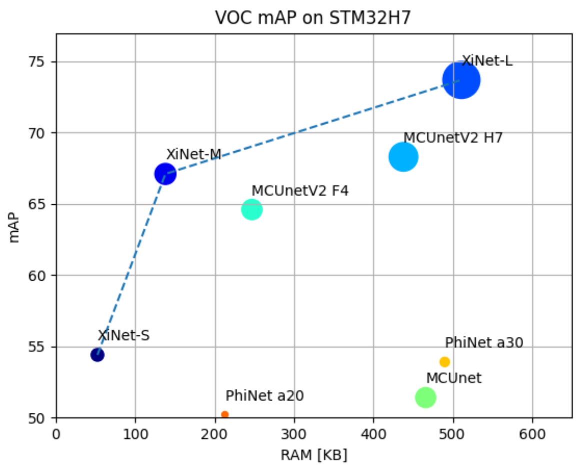

Figure 1. Our proposed architecture allows for a $9\times$ reduction in RAM and a $2\times$ reduction in operations compared to other tinyML approaches for the same level of performance. Circle size is proportional to the number of operations.

Other studies [12, 11] consider these factors and specifically focus on neural network deployment on microcontroller units (MCUs). Nonetheless, these works rely on custom hardware accelerators, and compilers [12, 11] to optimize the inference step; thus, they can not quickly scale to the variety of embedded platforms available on the market. Another limiting factor of these approaches is the effort required to adapt the neural networks to new tasks and devices with different computational constraints since the architecture design results from a Neural Architecture

*These authors contributed equally to this research.
This work has been supported by the European Union's Horizon 2020 research and innovation program under grant agreement no. 957337 (MARVEL project).

Search (NAS) approach. In [15], the authors propose a novel scaling paradigm to optimize neural networks given the changing hardware requirements by independently tuning the amount of RAM, FLASH, and energy required.

Based on the conclusions of [15], we set out to study the direct cost of the most common neural operators used as building blocks of ConvNets. Accordingly, we use the presented results to guide the development of a novel family of convolutional neural networks called XiNets. We demonstrate their state-of-the-art performance-complexity trade-off. Figure 1 summarizes one of the results of XiNet: a significant reduction in RAM usage by almost one order of magnitude compared to other similarly performing detectors.

The contribution of our paper is three-fold and can be summarized in the following bullet points:

- we propose the use of direct metrics to guide the design of neural networks for edge processing after highlighting the main drawbacks of current operators - Sections 2 and 3;
- we derive a novel convolutional block from the efficiency analysis and exploit it to design a novel neural architecture that maximizes the performance-energy trade-off - Sections 4, 5, and 6;
- we validate the proposed approach on object detection and image classification on a Raspberry Pi 4 single board computer and an `STM32H7` microcontroller (MCU) - Section 7.

2. Direct metrics for efficiency estimation

As already anticipated in Section 1, the number of multiply-adds (MACs) is not a good estimator of inference efficiency. The tendency to optimize towards these indirect metrics yielded a set of networks [1, 16, 17, 6], which use depthwise convolutions (DWConvs) as basic building blocks. While these networks achieve low MACs and parameter counts, their latency and thus, energy consumption, could be optimized. The optimization margin results from the inefficiency of indirect metrics, which do not consider important factors such as memory accesses. For this reason, we propose to use direct metrics to guide the design of neural architectures for the edge. Following is a description of the direct metrics we considered in our benchmarking and how they transfer to inference on edge devices.

RAM usage is one of the most constraining factors when deploying neural networks on edge devices. It directly relates to the size of the largest tensors required for operations within the network's inference step. It also depends on the chosen inference engine. Nonetheless, relative improvements in RAM usage considering a fixed inference engine remain significant; thus, also the comparison between operators is valid and generalizes among engines.

Parameter count, similarly to RAM usage, poses a hard constraint on which platforms can host a specific neural network, as this directly correlates to the amount of FLASH memory used.

Arithmetic intensity [4] is a metric that measures the average number of arithmetic operations executed per each byte loaded in a CPU registry. For a fixed amount of computing, higher arithmetic intensity corresponds to lower memory access costs, thus making the operation less memory-bound and, finally, more efficient.

To better understand why depthwise convolutions (DWConvs) are not - directly - the best option for edge inference, we hereafter analyze the cost of using DWConvs, especially in ConvNets based on inverted residual blocks. During the inference step of an inverted residual block that processes tensors of size $W \times H \times C$ (width, height, channels), after the expansion convolution, the RAM contains a $W \times H \times Ct$ tensor - where t is the expansion factor - and another $W \times H \times C$ tensor for the skip connection. To put this in perspective, let us consider an application that requires deploying an inverted residual block with an expansion factor $t = 6$ on a high-end MCU (e.g., `STM32F7`), with $R = 320KB$ of RAM. If we consider an input tensor with a spatial resolution $W, H = 128$, the RAM constraints of the device impose a maximum of C_{max}^{DW} feature channels that can be stored in memory, where:

$$C_{max}^{DW} = \lfloor \frac{R}{W \times H \times (t+1)} \rfloor = 3. \qquad (1)$$

In contrast, for a standard convolutional block, the required RAM only depends on the input tensor size. Therefore, in the same experimental conditions, it leads to a maximum number of feature channels for a standard convolution (C_{max}^{S}) being

$$C_{max}^{S} = \lfloor \frac{R}{W \times H \times 2} \rfloor = 10. \qquad (2)$$

Therefore, for a fixed amount of RAM, the standard convolutional block allows for a significant increase in output channels.

Another crucial shortcoming of DWConvs is highlighted when considering the memory accesses, as executing a DWConvs requires many more load and store operations than arithmetic operations, as also demonstrated in [26]. Running a DWConv on a SIMD-equipped ARM core, with inference in `uint8`, requires that for each multiply-add operation correspond three SIMD load operations (for input, kernel, and output) and one SIMD store operation (for the output). As the registers can not maintain the kernel loaded, due to it being different for each channel, this process leads to inefficiency. In fact, new values will need to be loaded for the next channel, leading to a low arithmetic intensity and a memory-bound inference rate. On the other hand, higher feature reuse in standard and pointwise convolutions leads to higher arithmetic intensity, thanks to the fewer memory accesses of these operators for a similar number of operations.

Despite the possible issues of DWConvs, when considering the RAM usage and arithmetic intensity, they benefit from significantly reducing the number of parameters required to store the kernel compared to a standard convolution. In fact, a standard convolution with kernel size $K \times K$ and C input and output channels requires K^2C^2 parameters, whereas a depthwise convolution with the exact specifications requires only K^2C parameters. However, DWConvs are generally used in inverted residual blocks, thus preceded by a pointwise convolution with an expansion factor t and followed by the corresponding projection convolution. In this design, an inverted residual block with $C = 32$ channels, expansion factor $t = 6$, and a kernel size $K = 3$ results in an increased parameters (P) count of:

$$\frac{P^{DW+2PW}}{P^S} = \frac{2tC + K^2}{K^2C} \approx 1.4 \tag{3}$$

Finally, we demonstrated how using depthwise convolutions in inverted residual blocks is not the Pareto-optimal solution to the performance-energy and, thus performance-complexity trade-offs. A direct assessment of the efficiency metrics might guide novel, more efficient ConvNets.

3. Energy requirements for common operators

When optimizing the performance-complexity trade-off with indirect complexity metrics, DWConvs are usually the best choice. However, as we have thoroughly demonstrated in Section 2, it does not imply that the energy efficiency of such operators is optimal. In this section, we propose an efficiency metric, η, that enables a direct comparison of neural operators in different experimental setups.

3.1. Efficiency metric

To measure the actual energy efficiency of different operators directly on devices with low computational capabilities, we benchmarked different operators by analyzing the power consumption, latency, number of clock cycles, and energy required on seven devices, ranging from low-power microcontrollers (e.g., STM32 Cortex-M series of processors) to higher-performance platforms (e.g., Raspberry Pi, with Cortex-A processor series). We also included in our analysis a processor equipped with a Tensor-Processing Unit (TPU) and a 9-core-32bit RISC V ultra-low power microcontroller. We assessed the actual efficiency of each operator (η_{op}) by calculating the ratio between the energy needed for a standard convolution (E_S) and the energy of the chosen operator (E_{op}) to perform an equivalent number of MACs.

$$\eta_{op} = \frac{E_S}{E_{op}} \tag{4}$$

In particular, this means that the higher the efficiency of an operator, the less power it requires for the same amount of MACs at inference time. This allows us to compare the efficiency of different operators while taking into account the variability of runtimes and different hardware platforms. To establish a baseline, we used standard convolutions as the reference operator, as it is the most commonly optimized operator on hardware platforms. While our efficiency metric was specifically designed for edge devices, the results can also be generalized to architectures such as GP-GPUs.

3.2. Operator efficiency

Following is an analysis of the efficiency of neural operators identified among those used in different ConvNets that target diverse tasks, ranging from image classification to image generation. In particular, we tested standard, pointwise, depthwise, and grouped convolutions as candidates for the convolutional block's operators. Moreover, we benchmarked transposed convolutions, nearest-neighbour, bilinear, and depth-to-space interpolation as upsampling operators. The latter is characteristic of generative models and used in upsampling necks for object detection. More details on the experimental choices are in the supplementary material, Section 4.

Figure 2 shows that the standard convolution operator is, as expected, the most efficient on average. Pointwise convolutions, being a specific case of the standard convolution, are similarly efficient. However, depthwise convolutions and convolutions based on groups or patches require approximately three times the energy needed for an equivalent standard convolution. The high efficiency of grouped convolutions on Greenwaves' `GAP8` is due to the processor being optimized for parallel execution. However, since we aim to propose a general approach applicable to most embedded systems, we consider `GAP8` as an outlier for this particular operator to avoid losing generality on other platforms.

We also evaluated the efficiency of different downsampling techniques (convolutions with stride, max-pooling, and average-pooling) usually present inside convolutional blocks. However, these variants are almost equivalent from an efficiency perspective. We did not consider batch normalization in this benchmarking, as it can be computed together with the preceding convolution at inference time.

For the benchmarking of the upsampling operators, all the operators tested demonstrated much lower efficiency than the standard convolution, as expected. Among them, bilinear upsampling has marginally higher efficiency, particularly when deployed on microcontrollers.

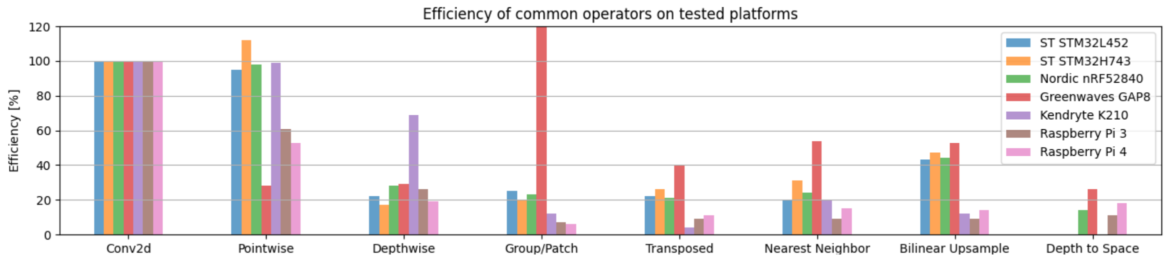

Figure 2. Measured real-world efficiency on different platforms for the benchmarked operators. Each color corresponds to a different hardware platform, each bar corresponds to the efficiency (defined in equation 4) of the tested operator, indicated in the horizontal axis.

4. XiNet design

Exploiting the analysis performed in Section 3.2, we propose a novel convolutional block based on the set of maximally efficient operators. In particular, our convolutional block comprises standard convolutions as the primary operator and targets embedded inference. Given the broad range of computational constraints among different embedded devices, we designed our convolutional block to easily adjust the trade-off between parameter count, RAM usage, energy, and performance.

4.1. Efficiency of standard convolutions

Standard convolutions' parameters and MAC count depend on the kernel size K, the number of input channels C_{in}, and the number of output channels C_{out}. To maximize efficiency and compatibility with different platforms (as seen in the provided additional materials, Section 4), we fix the kernel size to $K = 3$. Therefore, the MAC and parameter counts depend on C_{in} and C_{out}. In particular, we can:

- scale both C_{in} and C_{out} by multiplying them by a constant factor, α as in [16, 17];
- change γ, defined as the ratio between C_{in} and C_{out}, obtaining a reduction of parameters and MAC by a factor γ.

Using a standard convolution scaled with the described techniques has two advantages: first, we use a maximally efficient operator; thus, for the same MAC, we will get a network with lower power consumption. Secondly, we have a significant advantage in terms of RAM usage. For example, in an architecture based on inverted residual blocks, the dynamic memory needed by an operator is equal to the sum of the input and output tensor:

$$RAM^{DW} = 2tWHC_{out} \tag{5}$$

Instead, with an architecture based on the convolution pattern we propose, this is reduced to

$$RAM^{S} = WHC_{out} + \frac{WHC_{out}}{\gamma} \tag{6}$$

Therefore, for the same number of channels, we obtain a reduction in the required memory of

$$\frac{RAM^{DW}}{RAM^{S}} = 2t\frac{\gamma}{\gamma+1} \tag{7}$$

Considering $t = 6$ as in Mobilenetv2 and MCUNet and $\gamma = 4$ translates to a memory saving of almost $10\times$.

4.2. Convolutional block

To complete the design of the convolutional block, we insert a pointwise convolution before our optimized convolutional pattern to project the number of feature maps from C_{in} to C_{out}/γ as shown in Figure 3.

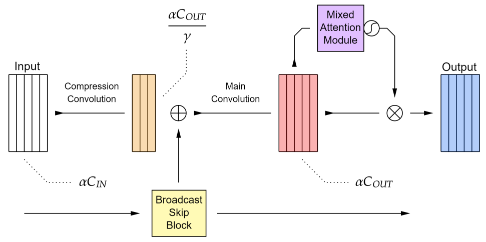

Figure 3. Structure of the proposed convolutional block. In the illustration, the output of the compression pointwise convolution is orange, the output of the main (3×3) convolution is red, and the output of the XiNet convolutional block is blue.

Since the architecture is designed to be quantized with minimal performance loss, we use max pooling as it allows us to achieve a higher signal-to-quantization noise ratio by increasing the dynamic range of the activations. For the same reason, we complete the sequence of operators by inserting a ReLU6 nonlinearity [7] instead of Swish/SiLU. As in MobilenetV2 [16], we verified that nonlinearity in the compressed subspace leads to the destruction of helpful information and a consequent decrease in performance. For this reason, we use ReLU6 only after the main convolution and not after the pointwise convolution.

Finally, we add an attention mechanism and skip connections to the convolutional block, described in Sections 5 and 6.

4.3. Hardware aware scaling

When designing XiNet, a ConvNet based on a sequence of our convolutional blocks, we wanted to resemble the hardware-aware scaling (HAS) properties of [15]. HAS is a novel scaling paradigm that focuses on tightly matching hardware requirements for different embedded platforms while allowing for a one-shot network architecture design without requiring additional training complexity. HAS independently optimizes the number of operations, number of parameters, and RAM usage. For this, we add β, a hyper-parameter that controls the trade-off between the number of operations and parameters. In practice, β linearly modifies the number of input and output channels in each block depending on its relative position in the network. Finally, the number of channels is rounded to the closest multiple of 4 to increase efficiency on platforms performing 4-way SIMD operations on uint8 data in 32-bit registers. To summarize, the number of channels for the i-th convolutional block is:

$$C^i_{out} = 4\lceil \alpha 2^{D_i - 2}\left(1 + \frac{(\beta-1)i}{N}\right) C^0_{out} \rceil \tag{8}$$

where D_i is the number of downsampling blocks before the i-th block, N is the total number of convolutional blocks, and C^0_{out} is the number of filters of the first convolutional block. As in ResNets [5], and as described in Eq. 8, the number of feature maps is doubled after each downsampling block.

In conclusion, the scaling properties of XiNets, and their computational complexity are completely defined by a set of three hyper-parameters:

- α, the width multiplier that linearly scales the number of convolutional filters within the whole architecture and affects the number of parameters, number of operations, and RAM usage quadratically;
- β, the shape factor, scales the number of filters within the architecture linearly, from αC_{out} in the first block to $\alpha\beta C_{out}$ in the last block. Varying α and β allows for a different trade-off between computational complexity and the number of parameters. $\beta < 1$ decreases the number of parameters at the expense of computational complexity. Conversely, $\beta > 1$ allows for the generation of computationally simpler networks but with a higher number of parameters;
- γ, the compression factor, scales the ratio between input and output filters of the compressed Conv2D. The number of parameters, operations, and RAM usage depends linearly on γ.

Modifying the XiNet hyper-parameters, as described in [15], enables scaling our neural architecture for different hardware platforms.

5. Efficient attention

Attention mechanisms play a crucial role in models' performance, as proved by many recent works in literature [23, 24]. However, this usually comes at the cost of higher computational complexity with respect to simple ConvNets. In this section, we describe an efficient attention module that can be plugged into any convolutional block and is currently integrated into ours as depicted in Figure 3. Attention mechanisms for image processing can be clustered into three different categories: channel attention [21, 8], spatial attention [14, 10, 21] and self-attention [19, 25].

In channel attention [21, 8], the attention mechanism extracts a one-dimensional vector encoding the relative channel importance of each input filter. Practically, the latter operation is achieved via a matrix-vector multiplication at the end of the module between the one-dimensional vector and the original feature map. This operation brings significant performance advantages while requiring a very low number of operations and parameters. However, these operations are very inefficient when used on embedded devices. Furthermore, they often bring compatibility issues due to the non-standard operations used within the block. Embedded device toolchains seldom support matrix-vector multiplication, and the global average pooling operations must be executed on the CPU even when using a purposefully designed NN engine. In fact, [15] uses a similar attention approach with Squeeze-and-Excitation blocks [8], leading to decreased error rates on `CIFAR100` of over 8% with a negligible 0.03% MAC increase. However, this comes at the cost of a 32% increase in execution latency and, consequently, energy consumption.

Spatial attention (SA) [14, 10, 21] instead is generally more efficient since it requires an element-wise multiplication of the attention module output and its input for which hardware accelerators are usually optimized. The main limitation of SA is generating the attention map, usually requiring non-standard operations and a high fragmentation (e.g., [21]).

Self-attention [19, 25] is the most problematic approach to bring to embedded devices. As it relies on computing a high dimensional correlation matrix between each pixel in the input feature map, this results in a tensor with spatial resolution $W^2 \times H^2$, i.e., $270MB$ of RAM when used on a 128×128 tensor, making this approach unfeasible for small embedded devices with $< 1MB$ of available RAM. Some variants of self-attention work on image patches instead of full-resolution images [19]. However, with the low amount of RAM usually available in off-the-shelf MCUs, the input resolution does not allow for meaningful patch generation.

5.1. Mixed channel and spatial attention

Given the results of Section 3.2, we propose an attention model that approximates a mixed channel- and spatial-attention mechanism while relying on the most efficient operators. In particular, we tested different configurations that generate a feature map with the same $W_{out} \times H_{out} \times C_{out}$ dimensions of the output feature map of a convolutional block. Thus, it is possible to perform an element-wise multiplication between the feature maps and the attention mask instead of the costly and usually non-optimized tensor-vector and tensor-matrix multiplication. A detailed breakdown of the different attention mechanisms benchmarked is available in the supplementary materials, Section 2.

To converge on a good trade-off between performance and computational cost, we tested six different attention mechanisms derived from our analysis and inspired by the literature:

- a modified Spatial Attention Module (SAM), similar to [10] based on a standard Conv2D (B in Figure 4);
- a lighter version of the SAM based on a striding operation performed on the channels axis (C in Figure 4);
- a lighter SAM relying on a pointwise operation, with and without nonlinearity (D in Figure 4);
- a variant of SAM where the main block convolution is split into 2 operations, using only one to compute the attention map. (E in Figure 4);
- a lightweight implementation of the first block, acting on the compressed representation (F in Figure 4).

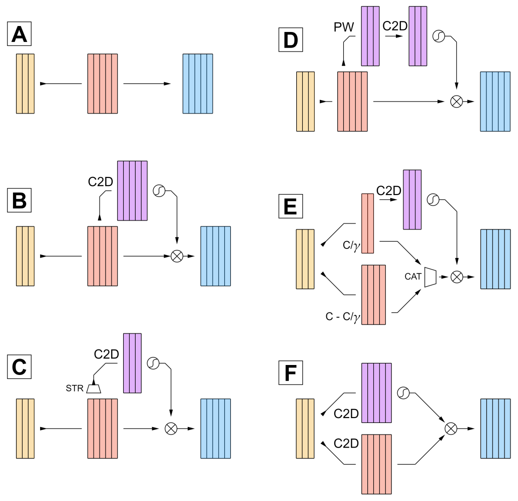

Figure 4. Tested attention modules: A - Base convolutional block without attention module. B - Naive convolution implementation as in YOLOv5. C - Striding-based. D - Pointwise convolution-based. E - Split-based. F - attention applied on compressed tensor.

We tested the listed attention mechanisms in a XiNet with five convolutional blocks on an image classification task, namely CIFAR-100. The relative accuracy and latency

Module	Energy	MAC/s	Acc Gain@1
None	1.68 mJ	137.7M	+0%
Conv2D	10.2 mJ	143.9M	+11.4%
Stride	3.54 mJ	115.7M	+6.1%
Pointwise	**3.46 mJ**	**122.4M**	**+10.1%**
Pointwise (relu)	3.46 mJ	122.1M	-0.5%
Split	**3.93 mJ**	**104.4M**	**+10.5%**
Compression	3.41 mJ	120.5M	+8.3%

Table 1. Relative accuracy on CIFAR-100 and energy consumption for different attention mechanisms measured on an STM32H743 MCU.

(measured on a STM32H743 MCU) for each of the attention variants are reported in Table 1.

As expected, implementing the block with standard convolutions provides the highest performance and MAC/s. However, operation count and, thus, energy consumption becomes too high for an MCU-targeted ConvNet. Lighter variants, like the pointwise- and split-based attention mechanisms, show a better performance-to-computation trade-off without compromising performance too much.

In XiNet, we use the pointwise-based attention module, as it is more straightforward, requires less fragmentation, and does not require a concatenation operation to merge multiple patches.

6. Broadcasted skip connections

Following the same protocol that guided the development of the convolutional block, we use only maximally efficient operators from Section 3.2 to design a broadcasting mechanism for skip connections. It is similar to the one used in [27] to connect layers that work on tensors with different spatial resolutions and feature maps. Our broadcasting skip connection performs an average pooling on the input tensor to match the spatial dimension of the output of the convolutional block. The downsampled tensor is then processed by a pointwise convolution to match the number of channels. Finally, we perform an element-wise sum, which is more efficient than concatenation.

Performing the two broadcasting operations in this particular order has two main advantages:

- in architectures with downsampling, previous convolutional blocks usually have fewer channels than the current block, and thus, performing pooling before increasing the channel count reduces the RAM needed to store the skip tensor;
- as deeper layers have lower spatial resolutions, after a pooling operation, we can discard the original high-resolution tensor and only store the pooled version for later layers, optimizing RAM usage even more.

To optimize the skip connections inside XiNet, we defined a ConvNet in which each layer receives information from all previous layers through the broadcasting skip block, similar to a DenseNet [9]. We assigned a trainable weight to each skip tensor and used a softmax operator to normalize the weight of all skip connections. Given a network with N blocks, we approximate a super-network containing the $N!$ sub-nets composed of all skip connection combinations, a process similar to that employed in [22]. We trained this architecture on the CIFAR-100 dataset and checked the scores assigned to each skip connection at convergence to find the optimal data path.

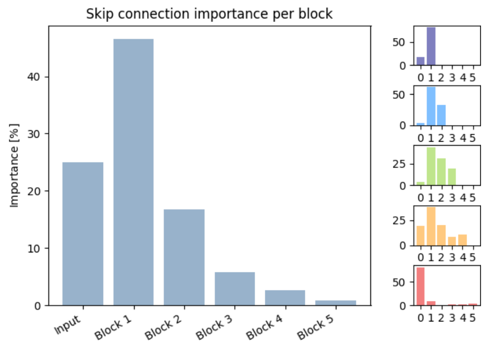

Figure 5. Average skip connection importance - defined as weight magnitude - per layer. The layer ID on the x-axis represents the input tensor of the broadcast skip module. The subplots on the right represent the single importances per layer.

Figure 5 demonstrates a notable preference in each convolutional block for tensors originating from the first convolutional block's output. This could be due to neighboring blocks extensively reusing the same features, as verified in [3], rendering skip connections unnecessary. Utilizing features from distant blocks maximizes feature diversity, leading to improved performance. These findings also enable effective exploitation of the broadcasting module's structure, as the first block has the fewest channels, reducing RAM usage for skip connections. For deeper blocks, this implies a $5\times$ reduction in the skip tensor's size.

6.1. Block design ablation study

Table 2 provides a detailed analysis of the behavior of the reference architecture with and without the enhancements suggested in Sections 5 and 6. Results indicate that the two proposed blocks improve the network performance without significantly altering block efficiency.

	Base			XiNet
Skip connections		✓		✓
Attention blocks			✓	✓
mAP	72.3	72.7	74.4	**74.9**
Efficiency	68%	64%	70%	67%

Table 2. Effects of the presence of the different building blocks on network performance and efficiency. '**Base**' is a sequence of convolutional blocks without attention and skip connections. '**XiNet**' is instead the complete architecture.

7. Experimental results

To demonstrate the effectiveness of our proposed architecture on image processing tasks, we evaluated it on three benchmark datasets: CIFAR-100, MS-COCO [13] and Pascal VOC [2]. We compare the performance of neural networks running on embedded platforms at up to 30 fps. Details on the baseline implementations and training procedure of XiNet are reported in the supplementary materials.

7.1. Microcontroller-scale image classification

To evaluate XiNet's performance on image classification tasks, we measured its accuracy, energy consumption, and frames per second (fps) on an STM32H7 MCU running at 280MHz and a Raspberry Pi 4. The computational cost of the network is summarized in Table 3. Table 4 compares XiNet's performance to other state-of-the-art approaches.

	res	α	β	γ	**RAM**	**MAC**
XiNet-Class	160	1	1.5	4.0	204KB	259M

Table 3. Hyperparameters of the benchmarked classification architecture, targeting an `STM32H7` MCU.

XiNet significantly outperforms depthwise-convolution based neural networks, achieving more than $3\times$ the fps while maintaining similar levels of classification accuracy. Furthermore, results demonstrate how XiNet can operate efficiently on resource-constrained microcontrollers without compromising accuracy or performance.

		STM32H7A3		**Raspberry Pi 4**	
	Acc	**fps**	**Energy**	**fps**	**Energy**
MCUNet in3	69.62%	1.34	53.0mJ	10.8	469mJ
MCUNet in4	72.86%	1.11	71.9mJ	7.70	657mJ
PhiNet	69.17%	2.40	31.6mJ	15.6	324mJ
XiNet-Class	72.27%	3.40	22.3mJ	30.6	165mJ

Table 4. Performance of our proposed method on the CIFAR-100 benchmark compared to other state-of-the-art approaches. These neural networks were benchmarked on a `STM32H7` MCU and a RaspberryPi 4.

7.2. Object Detection

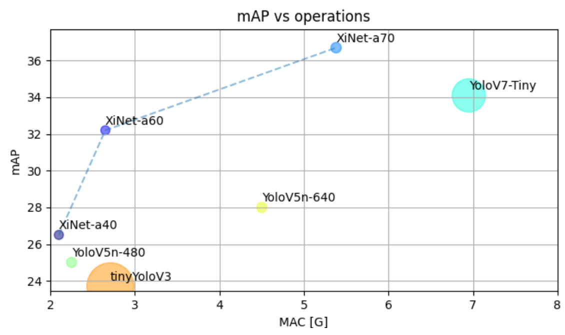

Figure 6. Comparison of our proposed architecture and different generations of the smallest YOLO detectors. Circle size is proportional to RAM usage.

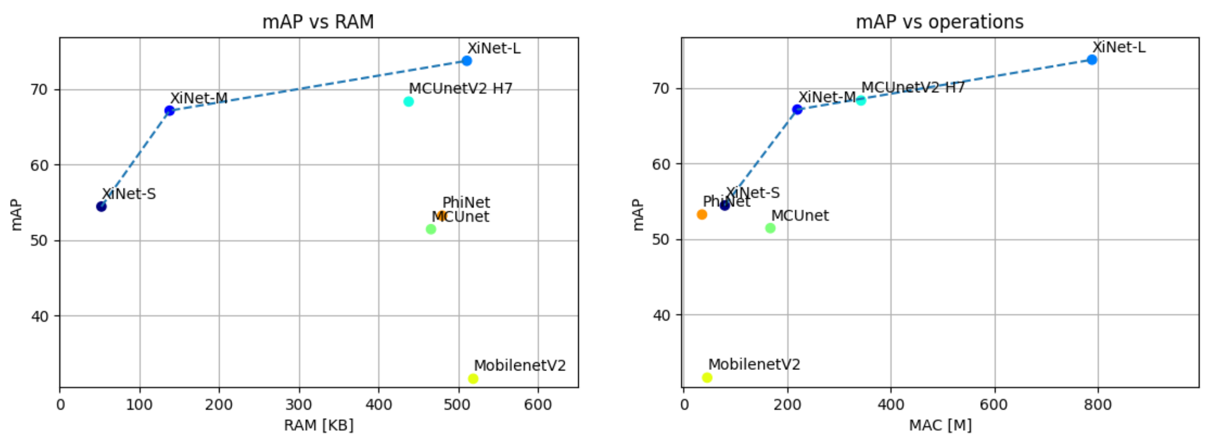

Figure 7. Comparison with other edge-oriented neural architectures showing the trade-offs between memory, operations, and object detection performance. XiNet outperforms MCUNet and PhiNet both in terms of RAM (left) and operation (right). Each color represents a different architecture (type or complexity).

To develop our tiny object detector, we started with the architecture of YoloV7 [20], offering state-of-the-art performance with relatively small models. We modified the original YoloV7 architecture replacing the backbone feature extractor with XiNet, and changed the network head to use XiNet convolutional blocks to optimize resource usage compared to standard convolutions.

As we are not targeting any specific platform, we use by default a compression factor $\gamma = 4.0$, a shape factor $\beta = 1.4$, and varied the width multiplier α to obtain networks with different operation counts. To evaluate our approach, we tested our object detector on the COCO object detection dataset [13] and compared it to other tiny object detectors in the YOLO family. Figure 6 shows each model's trade-off between mean average precision (mAP) and network complexity. Our results indicate that XiNet outperforms all other alternatives in terms of performance-complexity trade-off.

7.3. Microcontroller-scale object detection

This section evaluates XiNet's performance on computationally constrained embedded devices, specifically microcontrollers. We analyze the object detection performance of XiNet using the VOC2012 [2] dataset, comparing our results with the official results of the MCUNet [12, 11] family of architectures. We target a multimedia-oriented STM32H743 microcontroller with 2MB of available FLASH memory and 512KB of RAM that can be used for the network.

We propose three different architectures obtained by scaling different aspects of the reference architecture presented before. These architectures are detailed in Table 5:

- XiNet-L is the biggest architecture that can fit the target platform. It demonstrates the best possible performance achievable by scaling the network using HAS to use all available FLASH and RAM and as many operations as possible.
- XiNet-S is optimized for energy consumption and memory usage, showcasing the flexibility of trading operations and parameters to reduce energy consumption significantly.
- XiNet-M is a balanced architecture, leveraging the scaling principles used in HAS to use all available FLASH to optimize the trade-offs between accuracy, memory, and energy.

	res	α	β	γ	**RAM**	**MAC**
XiNet-S	256	0.4	2.0	4.0	53KB	80M
XiNet-M	384	0.4	2.0	4.0	138KB	220M
XiNet-L	416	0.45	1.8	4.0	511KB	789M

Table 5. Hyperparameters of the benchmarked architectures targeting an `STM32H7` MCU, with corresponding computational requirements.

Results for object detection on the VOC-2012 dataset are shown in Figure 7. In particular, these experiments prove how independently scaling parameters, RAM, and number of operations allows XiNet-based object detectors to significantly outperform other methods by making the best use of all the available FLASH offered by the platform, thus allowing for a significant increase in mAP. When scaling XiNets to match the performance of current state-of-the-art models (e.g. `MCUnetV2 H7`), we obtain a reduction in the number of operations of $2\times$ and a reduction in RAM usage of $9\times$. XiNet-S, for example, can achieve $8fps$ on 320×240 images on an `STM32H743` MCU, while XiNet-M can run on a Raspberry Pi 4 at $3fps$ when working on 640×480 images.

8. Conclusion

This paper proposes XiNet, a novel neural network architecture developed to optimize compatibility among hardware platforms and energy efficiency. We designed the network building blocks, including convolutional blocks, attention mechanisms, and broadcasting skip connections, to exploit the hardware's capabilities maximally. XiNet offers advanced scalability features that allow the network to be scaled for specific computational requirements (FLASH, RAM, and energy budget) using Hardware Aware Scaling. Our results demonstrate that XiNet has superior performance in object detection and image classification for embedded platforms with the ability to run under real-time constraints. XiNet outperforms other state-of-the-art networks in object detection with a fixed complexity budget, achieving more than twice the frames per second for the same performance on various hardware platforms.

References

[1] François Chollet. Xception: Deep learning with depthwise separable convolutions. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1800–1807, 2017. 1, 2

[2] Mark Everingham, Luc Van Gool, Christopher K. I. Williams, John M. Winn, and Andrew Zisserman. The pascal visual object classes (VOC) challenge. *International Journal of Computer Vision*, 88:303–338, 2009. 7, 8

[3] Kai Han, Yunhe Wang, Qi Tian, Jianyuan Guo, Chunjing Xu, and Chang Xu. Ghostnet: More features from cheap operations. *CoRR*, abs/1911.11907, 2019. 7

[4] Mark Harris. Mapping Computational Concepts to GPUs. In *ACM SIGGRAPH 2005 Courses*, SIGGRAPH '05, page 50–es, New York, NY, USA, 2005. Association for Computing Machinery. 1, 2

[5] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. pages 770–778, 06 2016. 5

[6] Andrew G. Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, Quoc V. Le, and Hartwig Adam. Searching for mobilenetv3. *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 1314–1324, 2019. 1, 2

[7] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *CoRR*, abs/1704.04861, 2017. 4

[8] 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. 5

[9] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q. Weinberger. Densely connected convolutional networks. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2261–2269, 2017. 7

[10] Glenn Jocher, Ayush Chaurasia, Alex Stoken, Jirka Borovec, NanoCode012, Yonghye Kwon, TaoXie, Kalen Michael, Jiacong Fang, imyhxy, Lorna, Colin Wong, (Zeng Yifu), Abhiram V, Diego Montes, Zhiqiang Wang, Cristi Fati, Jebastin Nadar, Laughing, UnglvKitDe, tkianai, yxNONG, Piotr Skalski, Adam Hogan, Max Strobel, Mrinal Jain, Lorenzo Mammana, and xylieong. ultralytics/yolov5: v6.2 - YOLOv5 Classification Models, Apple M1, Reproducibility, ClearML and Deci.ai integrations, Aug. 2022. 5, 6

[11] Ji Lin, Wei-Ming Chen, Han Cai, Chuang Gan, and Song Han. Mcunetv2: Memory-efficient patch-based inference for tiny deep learning. In *NeurIPS*, 2021. 1, 8

[12] Ji Lin, Wei-Ming Chen, Yujun Lin, John Cohn, Chuang Gan, and Song Han. Mcunet: Tiny deep learning on IoT devices. *ArXiv*, abs/2007.10319, 2020. 1, 8

[13] Tsung-Yi Lin, Michael Maire, Serge J. Belongie, Lubomir D. Bourdev, Ross B. Girshick, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft COCO: common objects in context. *CoRR*, abs/1405.0312, 2014. 7, 8

[14] Mang Ning, Yao Lu, Wenyuan Hou, and Mihhail Matskin. Yolov4-object: an efficient model and method for object discovery. In *2021 IEEE 45th Annual Computers, Software, and Applications Conference (COMPSAC)*, pages 31–36, 2021. 5

[15] Francesco Paissan, Alberto Ancilotto, and Elisabetta Farella. PhiNets: A Scalable Backbone for Low-Power AI at the Edge. 2022. 2, 5

[16] Mark Sandler, Andrew G. Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4510–4520, 2018. 1, 2, 4

[17] Mingxing Tan and Quoc V. Le. Efficientnetv2: Smaller models and faster training. *ArXiv*, abs/2104.00298, 2021. 1, 2, 4

[18] Xiaohu Tang, Shihao Han, L. Zhang, Ting Cao, and Yunxin Liu. To bridge neural network design and real-world performance: A behaviour study for neural networks. In *MLSys*, 2021. 1

[19] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. 5

[20] Chien-Yao Wang, Alexey Bochkovskiy, and Hong-yuan Liao. Yolov7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors, 07 2022. 8

[21] Sanghyun Woo, Jongchan Park, Joon-Young Lee, and In So Kweon. Cbam: Convolutional block attention module. Berlin, Heidelberg, 2018. Springer-Verlag. 5

[22] Tien-Ju Yang, Andrew G. Howard, Bo Chen, Xiao Zhang, Alec Go, Vivienne Sze, and Hartwig Adam. Netadapt: Platform-aware neural network adaptation for mobile applications. *ArXiv*, abs/1804.03230, 2018. 7

[23] Li Yuan, Yunpeng Chen, Tao Wang, Weihao Yu, Yujun Shi, Francis E. H. Tay, Jiashi Feng, and Shuicheng Yan. Tokens-to-token vit: Training vision transformers from scratch on imagenet. *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 538–547, 2021. 5

[24] Li Yuan, Qibin Hou, Zihang Jiang, Jiashi Feng, and Shuicheng Yan. Volo: Vision outlooker for visual recognition. *IEEE transactions on pattern analysis and machine intelligence*, PP, 2022. 5

[25] Han Zhang, Ian Goodfellow, Dimitris Metaxas, and Augustus Odena. Self-attention generative adversarial networks. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 7354–7363. PMLR, 09–15 Jun 2019. 5

[26] Pengfei Zhang, Eric Lo, and Baotong Lu. High performance depthwise and pointwise convolutions on mobile devices. In *AAAI*, 2020. 2

[27] Zongwei Zhou, Md Mahfuzur Rahman Siddiquee, Nima Tajbakhsh, and Jianming Liang. Unet++: Redesigning skip connections to exploit multiscale features in image segmentation. *IEEE Transactions on Medical Imaging*, 39:1856–1867, 2020. 6