

# An Ultra-low Power TinyML System for Real-time Visual Processing at Edge

Kunran Xu<sup>†</sup>, Huawei Zhang<sup>†</sup>, Yishi Li, Yuhao Zhang, Rui Lai, *Member, IEEE*, and Yi Liu

**Abstract**—Tiny machine learning (TinyML), executing AI workloads on resource and power strictly restricted systems, is an important and challenging topic. This brief firstly presents an extremely tiny backbone to construct high efficiency CNN models for various visual tasks. Then, a specially designed neural co-processor (NCP) is interconnected with MCU to build an ultra-low power TinyML system, which stores all features and weights on chip and completely removes both of latency and power consumption in off-chip memory access. Furthermore, an application specific instruction-set is further presented for realizing agile development and rapid deployment. Extensive experiments demonstrate that the proposed TinyML system based on our model, NCP and instruction set yields considerable accuracy and achieves a record ultra-low power of 160mW while implementing object detection and recognition at 30FPS. The demo video is available on <https://www.youtube.com/watch?v=mIZPxtJ-9EY>.

**Index Terms**—Convolutional neural network, tiny machine learning, internet of things, application specific instruction-set

## I. INTRODUCTION

Running machine learning inference on the resource and power limited environments, also known as Tiny Machine Learning (TinyML), has grown rapidly in recent years. It is promising to drastically expand the application domain of healthcare, surveillance and IoT, etc [1], [2]. However, TinyML presents severe challenges due to large computational load, memory demand and energy budget of AI models, especially in vision applications. For example, being a classical Convolutional Neural Network (CNN) model for object classification, AlexNet [3] requires about 1.4GOPS and 50MB weights. However, a typical TinyML system based on microcontroller unit (MCU) usually has only < 512KB on-chip SRAM, <2MB Flash and <1GOP/s computing ability. Meanwhile, because of the strict power limitation (<1W) [2], [4], TinyML system has no off-chip memory, e.g. DRAM, showing a huge gap between the desired and available hardware capacity.

Recently, the continuously emerging studies on TinyML achieve to deploy CNNs on MCUs by introducing memory-

<sup>†</sup> Authors contributed equally to this work.

This work was supported in part by the National Key R&D Program of China under Grant 2018YF070202800, Natural Science Foundation of China (NSFC) under Grant 61674120. (Corresponding author: Rui Lai).

Kunran Xu, Huawei Zhang, Yishi Li, Yuhao Zhang and Rui Lai are with the School of Microelectronics, Xidian University, Xi'an 710071, and also with the Chongqing Innovation Research Institute of Integrated Circuits, Xidian University, Chongqing 400031, China. (e-mail: aazztcc@gmail.com; myyzhww@gmail.com; yshlee1994@outlook.com; stuyuh@163.com; rlaix@mail.xidian.edu.cn).

Yi Liu is with the School of Microelectronics, Xidian University, Xi'an 710071, and also with the Guangzhou Institute of Technology, Xidian University, Guangzhou 510555, China. (yiliu@mail.xidian.edu.cn).

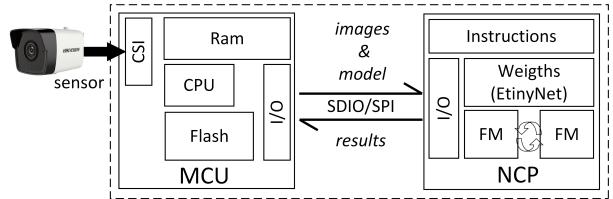


Fig. 1. The overview of the proposed TinyML system for visual processing.

efficient inference engines [1], [4], [5] and more compact CNN models [6], [7]. However, the existing TinyML systems still struggle to implement high-accuracy and real-time inference with ultra-low power consumption. Such as the state-of-the-art MCUNet [1] obtains 5FPS on STM32F746 but only achieves 49.9% top-1 accuracy on ImageNet. When the frame rate is increased to 10FPS, the accuracy of MCUNet further drops to 40.5%. What's more, running CNNs on MCUs is still not a extremely power-efficient solution due to the low efficiency of general purpose CPU in intensive convolution computing and massive weight data transmission.

Considering this, we propose to greatly promote TinyML system by jointly designing more efficient CNN models and specific CNN co-processor. Specifically, we firstly design an extremely tiny CNN backbone EtinyNet aiming at TinyML applications, which has only 477KB model weights and maximum feature map size of 128KB as well as yields remarkable 66.5% ImageNet Top-1 accuracy. Then, an ASIC-based neural co-processor (NCP) is specially designed for accelerating the inference. Since implementing CNN inference in a fully on-chip memory access manner, the proposed NCP achieves up to 180FPS throughput with 73.6mW ultra-low power consumption. On this basis, we propose a state-of-the-art TinyML system shown in Fig.2 for visual processing, which yields a record low power of 160mW in object detecting and recognizing at 30FPS.

In summary, we make the following contributions:

- 1) An extremely tiny CNN backbone named EtinyNet is specially designed for TinyML. It is far more efficient than existing lightweight CNN models.
- 2) An efficient neural co-processor (NCP) with specific designs for tiny CNNs is proposed. While running EtinyNet, NCP provides remarkable processing efficiency and convenient interface with extensive MCUs via SDIO/SPI.
- 3) Building upon the proposed EtinyNet and NCP, we promote the visual processing TinyML system to achieve a record ultra-low power and real-time processing efficiency, greatly advancing the TinyML community.

## II. SOLUTION OF OUR TINYML SYSTEM

Fig.1 shows the overview of the proposed TinyML system. Different from existing TinyML system, our system integrates MCU with its NCP on a compact board to achieve superior efficiency in a collaborative work manner.

Before performing inference, MCU sends the model weights and instructions to NCP who has sufficient on-chip SRAM to cache all these data. During inference, NCP executes the intensive CNN backbone workloads while MCU only performs the light-load pre-processing (color normalization) and post-processing (fully-connected layer, non-maximum suppression, etc). Running the proposed EtinyNet, the specially designed NCP can work in a single-chip manner, which reduces the system complexity as well as the memory access power and latency to the greatest extent. We will demonstrate that aforesaid division of labor greatly improves the processing efficiency in Section VI.

Considering the requirements of real-time communication, we interconnects NCP and MCU with SDIO/SPI interface. Since the interface is mainly utilized to frequently transmit input images and output results, the bandwidth of SDIO and SPI is sufficient for real-time data transmission. For instance, SDIO could provide up to 500Mbps bandwidth, which can transmit about 300FPS for  $256 \times 256 \times 3$  image and 1200FPS for  $128 \times 128 \times 3$  image. As for relatively slower SPI, it still reaches 100Mbps, or an equivalent throughput of 60FPS for  $256 \times 256 \times 3$  image. These two buses are widely supported by MCUs available in the market, which makes NCP can be applied in a wide range of TinyML systems.

Fig.2 shows the prototype verification system only consisting of STM32L4R9 MCU and our proposed NCP. Thanks to the innovative model (EtinyNet), co-processor (NCP) and application specific instruction-set, the entire system yields both of efficiency and flexibility.

## III. DETAILS OF PROPOSED ETINYNET MODEL

Since NCP handles CNN worksloads on-chip for pursuing extreme efficiency, the model size must be reduced as small as possible. By presenting Linear Depthwise Block (LB) and Dense Linear Depthwise Block (DLB), we derive an extremely tiny CNN backbone EtinyNet shown in Fig.3.

### A. Linear Depthwise Block

Depthwise-separable convolution [8], the key building block for lightweight CNNs only makes up of two operations: depthwise convolution (*DWConv*) and pointwise convolution (*PWConv*). Let  $\mathbf{I} \in \mathcal{R}^{C \times H \times W}$  and  $\mathbf{O} \in \mathcal{R}^{D \times H \times W}$  respectively represent the input feature maps and output feature maps, depthwise-separable convolution can be computed as

$$\mathbf{O} = \sigma(\phi_p(\sigma(\phi_d(\mathbf{I})))) \quad (1)$$

where  $\phi_d$ ,  $\phi_p$  represent the depthwise convolution and pointwise convolution while  $\sigma$  denotes the non-linearity activation function, e.g. ReLU. It has been demonstrated in [8] that ReLU in bottleneck block would prevent the flow of information and thus impair the capacity as well as expressiveness of model. In

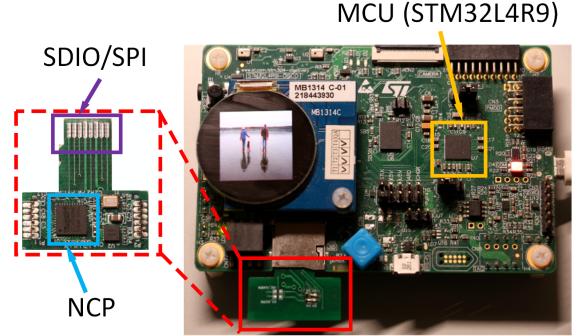


Fig. 2. The TinyML system for verification.

addition, we further observed that the ReLU behind depthwise convolution also declines the model accuracy. In view of this, we remove the ReLU behind *DWConv* and present the linear depthwise-separable convolution formulated as

$$\mathbf{O} = \sigma(\phi_p(\phi_d(\mathbf{I}))) \quad (2)$$

Since  $\phi_d$  and  $\phi_p$  are both linear, there exists a standard convolution  $\phi_s$  linearly combined of  $\phi_d$  and  $\phi_p$ , which is a specific case of sparse coding [9]. In existing lightweight models, depthwise convolution layers generally possess about only 5% of the total parameters but contribute greatly to model accuracy, which indicates depthwise convolution is with high parameter efficiency. Taking advantage of this, we further introduce additional *DWConv* of  $\phi_{d2}$  behind *PWConv* and build a novel linear depthwise block defined as

$$\mathbf{O} = \sigma(\phi_{d2}(\sigma(\phi_p(\phi_{d1}(\mathbf{I}))))) \quad (3)$$

As shown in Fig 3(a), the structure of proposed linear depthwise block (LB) can be represented as *DWConv-PWConv-DWConv*, which is apparently different from the commonly used bottleneck block of *PWConv-DWConv-PWConv* in other lightweight models. As for the reason, increasing the proportion of *DWConv* in model parameters is beneficial to improve the model accuracy.

### B. Dense Linear Depthwise Block

Restricted by the total number of parameters and size of feature maps, the width of network can not be too large. However, width of CNNs is important for achieving higher accuracy [10]. As suggested by [11], the structure with shortcut connection could be regarded as a wider network consisting of sub-networks. Therefore, we introduce the dense connection into LB for increasing its equivalent width. We refer the resulting block to dense linear depthwise block (DLB), which is depicted in Fig 3(b). Note that we take the  $\phi_{d1}$  and  $\phi_p$  as a whole due to the removal of ReLU, and add the shortcut connection at the ends of these two layers.

### C. Architecture of EtinyNet Backbone

By stacking the LB and DLB, we configure the EtinyNet backbone as indicated in Fig 3(c). Each line of which describes the block repeated  $n$  times. All layers in the same block have the same number  $c$  of output channels. The first layer of each

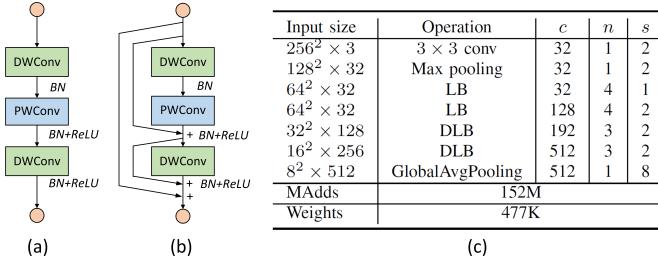


Fig. 3. The proposed building blocks that make up the EtinyNet. (a) is the linear depthwise block (LB) and (b) is the dense linear depthwise block (DLB). (c) is the configurations of the proposed EtinyNet backbone.

functional unit has a stride  $s$  and all others adopt stride 1. Since dense connection consumes more memory space, we only utilize DLB at the high level stages with much smaller feature maps. It's encouraging that EtinyNet backbone has only 477KB parameters (quantized in 8-bit) and still achieves 66.5% ImageNet Top-1 accuracy. The extreme compactness of EtinyNet makes it possible to design small footprint NCP that could run without off-chip DRAM.

#### IV. APPLICATION SPECIFIC INSTRUCTION-SET FOR NCP

For easily deploying tiny CNN models on NCP, we define a application specific instruction-set. As shown in Table I, the set contains 13 instructions respectively belonging to neural operation type (N type) and Control type (C type), which include basic operations for tiny CNN models widely used in image classification, object detection, *etc*. Each instruction encodes a network layer and consists of 128 bits: 5 bits are reserved for operation code, and the remaining 123 bits represent the attributes of operations and operands. Program 1 illustrates the assembly code of a LB in EtinyNet using our instruction set, it can be observed that LB can be easily built with only three instructions. The proposed instruction set has a relatively coarser granularity. Hence, general model can be built with fewer instructions ( $\sim 100$ ), which effectively reduces on-chip instruction memory and makes a good trade-off between efficiency and flexibility.

TABLE I  
INSTRUCTION SET FOR PROPOSED NCP

Instruction format	Description	Type
<b>bn</b>	batch normalization	N
<b>relu</b>	non-linear activation operation	N
<b>conv</b>	$1 \times 1$ and $3 \times 3$ convolution & bn, relu	N
<b>dwconv</b>	$3 \times 3$ depthwise conv & bn, relu	N
<b>add</b>	elementwise addition	N
<b>move</b>	move tensor to target address	N
<b>dsam</b>	down-sampling by factor of 2	N
<b>usam</b>	up-sampling by factor of 2	N
<b>maxp</b>	max pooling by factor of 2	N
<b>gap</b>	global average pooling	N
<b>jump</b>	set program counter (PC) to target	C
<b>sup</b>	suspend processor	C
<b>end</b>	suspend processor and reset PC	C

#### V. DESIGN OF NEURAL CO-PROCESSOR

As shown in Fig.4, the proposed NCP consists of five main components: Neural Operation Unit (NOU), Tensor Memory

#### Program 1. LB Program Using Our Instruction Set

```
dwconv $b1 $b0 $w0 $w1 #b1=bn(b0 ⊕ w0,w1))
conv $b2 $b1 $w2 $w3 #b2=relu(bn(b1 ⊗ w2,w3))
dwconv $b1 $b2 $w4 $w5 #b1=relu(bn(b2 ⊕ w4,w5))
```

(TM), Instruction Memory (IM), I/O and System Controller (SC). When NCP works, SC firstly decodes one instruction fetched from IM and informs the NOU to start computing with decoded signal. The computing process takes multiple cycles, during which NOU reads operands from TM and writes results back automatically. Once completing the writing back process, SC continues to process the next instruction until an **end** or **suspend** instruction is encountered. When NOU is idle, TM is accessible through I/O. We will fully describe each component in the following parts.

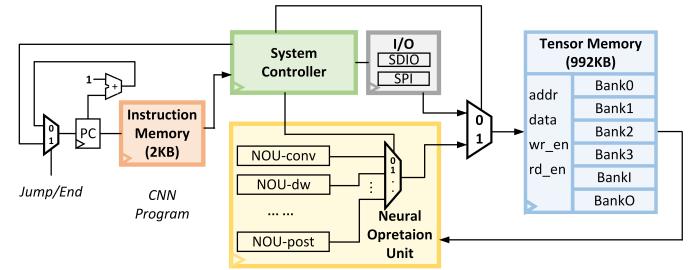


Fig. 4. The overall block diagram of the proposed processor NCP. NCP consists of Neural Operation Unit, Tensor Memory, Instruction Memory, I/O and System Controller.

#### A. Neural Operation Unit

NOU contains three sub-modules termed NOU-conv, NOU-dw and NOU-post for supporting the corresponding neural operations of **conv**, **dwconv** and **bn**.

1) NOU-conv processes the  $3 \times 3$  convolution and **PWConv** in the EtinyNet backbone. It firstly converts input feature maps (IF) and kernels (KL) to matrixes with *im2col* operation and then performs matrix multiply-accumulate operation [12], [13] with a  $T_{oc} \times T_{hw}$  8-bit MAC array to realize convolution. Different from other ASIPs [14], [15] with fine grained instructions, the hardwired computing control logic of our NOU-conv considerably improves the processing efficiency.

2) NOU-dw employs shift registers, multipliers and adder trees in classic convolution processing pipeline [16] to perform **dwconv** operation. It arranges 9 multipliers and 8 adders in each processing pipeline to handle **DWConv**. With the help of shift registers, each pipeline could cache neighborhood pixels and produce convolution result in a output channel every cycle. For accelerating the convolution computing, we arrange total  $T_{oc}$  of 2D convolution processing pipelines in NOU-dw module to implement parallel computation in  $N_{oc}$  dimensionality.

3) NOU-post implements BN, ReLU and element addition operations. It applies single-precision floating-point computing with  $T_{oc}$  postprocess units. Each postprocess unit contains interg2float module, floating-point MAC, ReLU module and float2integer module. The input of postprocess unit comes from NOU-conv, NOU-dw or TM, selected by a multiplexer.

Therefore, results computed by **conv** or **dwconv** could be directly sent to NOU-post, which allows BN, ReLU to be fused with **conv** and **dwconv**, considerably cutting down the number of memory access.

### B. Tensor Memory

TM is a single-port SRAM consists of 6 banks, whose width is  $T_{tm} \times 8$  bits, as shown in Fig 4. Thanks to the compactness of EtinyNet, NCP only requires totally 992KB on-chip SRAM. The BankI (192KB) is responsible for caching input  $256 \times 256 \times 3$  sized color images. The 128KB sized Bank0 and Bank1 are arranged for caching feature maps, while Bank2 and Bank3 with larger size of 256KB are used for storing model weights. The 32KB sized BankO is used to store computing results, such as feature vectors, heatmaps [17] and bonding boxes [18], etc. TM's small capacity and simple structure yield our NCP a small footprint.

### C. Configurable Tensor Layout for Parallelism

For memory access efficiency and processing parallelism, we specially design pixel-major layout and interleaved layout. As shown in Fig.5(a), for the pixel-major layout, all pixels of the first channel are sequentially mapped to TM in a row-major order. Then, the next channels are arranged in the same pattern until all channels in a tensor are stored. Pixel-major layout is convenient for operations that need to obtain continuous column data in one memory access but is efficiency for those operations requiring to access continuous channel data in one cycle, like **dwconv**, **usam**, etc. In this situation, the **move** instruction is employed to transform tensor into interleaved layout. In this layout, as shown in Fig.5(b), the whole tensor is divided into  $N_c // T_{tm}$  tiles and are placed in TM sequentially, while each tile is arranged in a channel-major order. With these two tensor layout, NCP can efficiently utilize TM bandwidth, greatly reducing memory access latency.

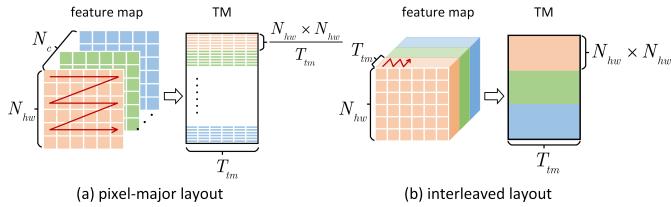


Fig. 5. Illustration of different tensor layouts. (a) Pixel-major layout. (b) Interleaved layout.

### D. Characteristics

We implement our NCP using TSMC 65nm low power technology. While  $T_{oc} = 16$  and  $T_{hw} = 32$ , NCP contains 512 of 8-bit MACs in NOU-conv, 144 of 8-bit multipliers and 16 of adder trees in NOU-dw, and 16 of single precision floating-point MACs in NOU-post. The  $T_{tm}$  is set to 32, so TM has a data width of 256. When working at 100Mhz, NOU-conv and NOU-post are active every cycle so that NCP achieves a peak activity of 105.6 GOP/s.

## VI. EXPERIMENTAL RESULTS

We respectively assess the effectiveness of our proposed EtinyNet, NCP and the corresponding TinyML system.

### A. EtinyNet Evaluation

Table II lists the ImageNet 1000 categories classification results of the most well-known lightweight CNN architectures, including MobileNet series [8], ShuffleNet [19], MnasNet [20] and MicroNet [21], which have backbone (except fully-connected layer) size between of 0.5MB to 1MB. We pay more attention to the backbone because the fully-connected layer is not involved in all visual processing models and its parameter size is linearly related to the number of categories. Among all these results, our EtinyNet achieves the highest accuracy, reaching 66.5% top-1 accuracy and 87.2% top-5 accuracy. It outperforms the most competitive models of MobileNeXt-0.35 with significant 2.7%. Meanwhile, the proposed EtinyNet has the smallest model size, only about 58% of MobileNeXt-0.35, which demonstrates its high parameter efficiency. In addition, the more compact version EtinyNet-0.75 and EtinyNet-0.5 (the width of each layer shrunk by the factor of 0.75 and 0.5) still obtain competitive accuracy of 64.4% and 59.3%, respectively. Obviously, EtinyNet simultaneously yields higher accuracy and lower storage consumption for TinyML system.

TABLE II  
COMPARISON OF STATE-OF-THE-ART TINY MODELS OVER ACCURACY ON IMAGENET-1000 DATASET. "F" AND "B" DENOTE THE PARAMS. FOR FULLY-CONNECTED LAYER AND BACKBONE RESPECTIVELY.

Model	Params. (KB)	Top-1 Acc.	Top-5 Acc.
MobileNeXt-0.35	1024(F) / 812(B)	64.7	85.7
MnasNet-A1-0.35	1024(F) / 756(B)	64.4	85.1
MobileNetV2-0.35	1024(F) / 740(B)	60.3	82.9
MicroNet-M3	864(F) / 840(B)	61.3	82.9
ShuffleNetV2-0.5	1024(F) / 566(B)	61.1	82.6
EtinyNet	512(F) / 477(B)	<b>66.5</b>	<b>86.8</b>
EtinyNet-0.75	384(F) / 296(B)	64.4	85.2
EtinyNet-0.5	320(F) / 126(B)	59.3	81.2

### B. NCP Evaluation

We test our NCP of running EtinyNet backbone and compare it with other state-of-the-art CNNs accelerator in terms of latency and power consumption. As shown in Table III, the proposed NCP only takes 5.5ms to process one frame or yields an equivalent processing throughput of 180FPS, about  $2.6\times$  faster than the second fastest ConvAix [15] and  $4.7\times$  faster than Eyeriss [14]. What's more, NCP consumes only 73.6mW and achieves prominent high energy efficiency of 611.4 GOP/s/W, which is superior to other designs except for NullHop [22] manufactured with more advanced technology. When considering processing efficiency, that is, the number of frames that can be processed per unit time and per unit power consumption, NCP reaches extremely high of 449.1 Frames/s/mJ, at least  $29\times$  higher than other designs. As for the reason, it can be explained by 1) NCP performs inference without accessing off-chip memory, significantly reducing the data transmission power consumption and latency; 2) Our coarse-grained instruction set shrinks the control logic overhead.

TABLE III  
COMPARISON WITH STATE-OF-THE-ART NEURAL PROCESSORS.

Component	Eyeriss	NullHop	ConvAix	NCP
Technology	65nm	28nm	28nm	65nm
Core area [mm <sup>2</sup> ]	12.3	6.3	3.53	10.88
DRAM used	yes	yes	yes	none
FC support	none	none	yes	none
CNN Model	AlexNet	VGG16	MobilenetV1	EtinyNet
ImageNet Acc. [%]	59.3	68.3	70.6	66.5
Latency [ms]	25.9	72.9	14.2	5.5
Power [mW]	277.0	155.0	313.1	73.6
Energy efficiency (GOP/s/W)	433.8	2714.8	256.3	611.4
Processing efficiency (Frames/s/mJ)	5.38	1.21	15.18	449.1

### C. TinyML System Verification

We compare our proposed system with existing prominent TinyML systems based on MCUs. As shown in Table IV, the state-of-the-art CMSIS-NN [4] only obtains 59.5% Imagenet top-1 accuracy at 2FPS. MCUNet promotes the throughput to 5FPS, but pays the cost of accuracy dropping to 49.9%. In comparison, our solution reaches up to 66.5% accuracy and 30FPS, achieving the goal of real-time processing at edge. Furthermore, since existing methods take MCUs to complete all the CNN workloads, they must use high-performance MCUs (STM32H743, STM32F746) and run at the upper-limit frequency (480MHz for H732 and 216MHz for F746), which results in considerable power consumption of about 600mW. On the contrary, the proposed solution allows us to perform the same task only with a low-end MCU (STM32L4R9) running at 120MHz, which boosts the energy efficiency of the entire system and achieves an ultra-low power of 160mW.

TABLE IV  
COMPARISON WITH MCU-BASED DESIGNS ON IMAGE CLASSIFICATION (CLS) AND OBJECT DETECTION (DET) TASKS. \* DENOTES OUR REPRODUCED RESULTS.

	Methods	Hardware	Acc/mAP	FPS	Power
Cls	CMSIS-NN	H743	59.5%	2	*675 mW
	MCUNet	F746	49.9%	5	*525 mW
	Ours	L4R9+NCP	66.5%	30	160 mW
Det	CMSIS-NN	H743	31.6%	10	*640 mW
	MCUNet	H743	51.4%	3	*650 mW
	Ours	L4R9+NCP	56.4%	30	160 mW

In addition, we benchmark the object detection performance of our MCU+NCP and other SOTA MCU-based designs on Pascal VOC dataset. The mAP and throughput results shown in Table IV indicate that our system also greatly improves the performance in object detection task, which makes AIoT more promising to be applied in extensive applications.

## VII. CONCLUSION

In this paper, we propose an ultra-low power TinyML system for real-time visual processing by designing 1) a extremely tiny CNN backbone EtinyNet, 2) an ASIC-based neural co-processor and 3) an application specific instruction set. Our study greatly advances the TinyML community and promises to drastically expand the application scope of AIoT.

## REFERENCES

- [1] J. Lin, W. Chen, Y. Lin, J. Cohn, C. Gan, and S. Han, “Mcunet: Tiny deep learning on iot devices,” in *Annual Conference on Neural Information Processing Systems, December 6-12, 2020, virtual*.
- [2] M. Shafique, T. Theocharides, V. J. Reddy, and B. Murmann, “Tinyml: Current progress, research challenges, and future roadmap,” in *58th ACM/IEEE Design Automation Conference, DAC 2021, San Francisco, CA, USA, December 5-9, 2021*. IEEE, 2021, pp. 1303–1306.
- [3] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” *Commun. ACM*, vol. 60, no. 6, pp. 84–90, 2017.
- [4] L. Lai, N. Suda, and V. Chandra, “CMSIS-NN: efficient neural network kernels for arm cortex-m cpus,” *CoRR*, vol. abs/1801.06601, 2018.
- [5] A. Capotondi, M. Rusci, M. Fariselli, and L. Benini, “Cmix-nn: Mixed low-precision CNN library for memory-constrained edge devices,” *IEEE Trans. Circuits Syst. II Express Briefs*, vol. 67-II, no. 5, pp. 871–875, 2020.
- [6] C. R. Banbury, C. Zhou, I. Fedorov, R. M. Navarro, U. Thakker, D. Gope, V. J. Reddi, M. Mattina, and P. N. Whatmough, “Micronets: Neural network architectures for deploying tinyml applications on commodity microcontrollers,” *CoRR*, vol. abs/2010.11267, 2020.
- [7] R. T. N. Chappa and M. El-Sharkawy, “Deployment of se-squeezezenext on nxp bluebox 2.0 and nxp i.mx rt1060 mcu,” in *2020 IEEE Midwest Industry Conference (MIC)*, vol. 1, 2020, pp. 1–4.
- [8] M. Sandler, A. G. Howard, M. Zhu, A. Zhmoginov, and L. Chen, “Mobilenetv2: Inverted residuals and linear bottlenecks,” in *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR, Salt Lake City, UT, USA, June 18-22, 2018*, pp. 4510–4520.
- [9] H. Lee, A. Battle, R. Raina, and A. Ng, “Efficient sparse coding algorithms,” in *NIPS*, 2006.
- [10] S. Zagoruyko and N. Komodakis, “Wide residual networks,” *ArXiv*, vol. abs/1605.07146, 2016.
- [11] Z. Wu, C. Shen, and A. V. Hengel, “Wider or deeper: Revisiting the resnet model for visual recognition,” *Pattern Recognit.*, vol. 90, pp. 119–133, 2019.
- [12] M. E. Nojehdeh, S. Parvin, and M. Altun, “Efficient hardware implementation of convolution layers using multiply-accumulate blocks,” in *IEEE Computer Society Annual Symposium on VLSI, ISVLSI 2021, Tampa, FL, USA, July 7-9, 2021*. IEEE, 2021, pp. 402–405.
- [13] S. Zhang, V. Karihaloo, and P. Wu, “Basic linear algebra operations on tensorcore GPU,” in *11th IEEE/ACM Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems, ScalA@SC 2020, Atlanta, GA, USA, November 13, 2020*. IEEE, 2020, pp. 44–52.
- [14] Y. Chen, T. Krishna, J. S. Emer, and V. Sze, “Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks,” *IEEE J. Solid State Circuits*, vol. 52, no. 1, pp. 127–138, 2017.
- [15] A. Bytyn, R. Leupers, and G. Ascheid, “Convax: An application-specific instruction-set processor for the efficient acceleration of cnns,” *IEEE Open J. Circuits Syst.*, vol. 2, pp. 3–15, 2021.
- [16] T. Chen, Z. Du, N. Sun, J. Wang, C. Wu, Y. Chen, and O. Temam, “Diannao: a small-footprint high-throughput accelerator for ubiquitous machine-learning,” in *ASPLOS, Salt Lake City, UT, USA, March 1-5, 2014*. ACM, pp. 269–284.
- [17] K. Xu, R. Lai, L. Gu, and Y. Li, “Multiresolution discriminative mixup network for fine-grained visual categorization,” *IEEE Transactions on Neural Networks and Learning Systems*, Early Access, doi:10.1109/TNNLS.2021.3112768.
- [18] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. E. Reed, C.-Y. Fu, and A. Berg, “Ssd: Single shot multibox detector,” in *ECCV*, 2016.
- [19] X. Zhang, X. Zhou, M. Lin, and J. Sun, “Shufflenet: An extremely efficient convolutional neural network for mobile devices,” in *2018 IEEE Conference on Computer Vision and Pattern Recognition, Salt Lake City, UT, USA, June 18-22, 2018*. IEEE Computer Society, pp. 6848–6856.
- [20] M. Tan, B. Chen, R. Pang, V. Vasudevan, and Q. V. Le, “Mnasnet: Platform-aware neural architecture search for mobile,” *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2815–2823, 2019.
- [21] Y. Li, Y. Chen, X. Dai, D. Chen, M. Liu, L. Yuan, Z. Liu, L. Zhang, and N. Vasconcelos, “Micronet: Improving image recognition with extremely low flops,” *ArXiv*, vol. abs/2108.05894, 2021.
- [22] A. Aimar, H. Mostafa, E. Calabrese, A. Rios-Navarro, R. Tapiador-Morales, I. Lungu, M. B. Milde, F. Corradi, A. Linares-Barranco, S. Liu, and T. Delbrück, “Nullhop: A flexible convolutional neural network accelerator based on sparse representations of feature maps,” *IEEE Trans. Neural Networks Learn. Syst.*, vol. 30, no. 3, pp. 644–656, 2019.