Distilling Bit-level Sparsity Parallelism for General Purpose Deep Learning Acceleration

Hang Lu
luhang@ict.ac.cn
State Key Laboratory of Computer Architecture, Institute of Computing Technology, CAS
Beijing, China

Liang Chang
liangchang@uestc.edu.cn
University of Electronic Science and Technology of China
Chengdu, China

Chenglong Li
Chenglong_Li@std.uestc.edu.cn
University of Electronic Science and Technology of China
Chengdu, China

Zixuan Zhu
zxzhu@std.uestc.edu.cn
University of Electronic Science and Technology of China
Chengdu, China

Shengjian Lu
lushengjian2021@gmail.com
State Key Laboratory of Computer Architecture, Institute of Computing Technology, CAS
Beijing, China

Yanhuan Liu
gulingshiyans@163.com
State Key Laboratory of Computer Architecture, Institute of Computing Technology, CAS
Beijing, China

Mingzhe Zhang
zhangmingzhe@iie.ac.cn
State Key Laboratory of Information Security, Institute of Information Engineering, CAS
Beijing, China

ABSTRACT

Along with the rapid evolution of deep neural networks, the ever-increasing complexity imposes formidable computation intensity to the hardware accelerator. In this paper, we propose a novel computing philosophy called "bit interleaving" and the associate accelerator design called "*Bitlet*" to maximally exploit the bit-level sparsity. Apart from existing bit-serial/parallel accelerators, *Bitlet* leverages the abundant "sparsity parallelism" in the parameters to enforce the inference acceleration. *Bitlet* is versatile by supporting diverse precisions on a single platform, including floating-point 32 and fixed-point from $1b$ to $24b$. The versatility enables *Bitlet* feasible for both efficient inference and training. Empirical studies on 12 domain-specific deep learning applications highlight the following results: (1) up to 81×/21× energy efficiency improvement for training/inference over recent high performance GPUs; (2) up to 15×/8× higher speedup/efficiency over state-of-the-art fixed-point accelerators; (3) $1.5mm^2$ area and scalable power consumption from $570mW$ ($float32$) to $432mW$ ($16b$) and $365mW$ ($8b$) @$28nm$ TSMC; (4) highly configurable justified by ablation and sensitivity studies.

Hang Lu and Liang Chang contributed equally to this research. Mingzhe Zhang is the corresponding author (zhangmingzhe@iie.ac.cn).

MICRO '21, October 18–22, 2021, Virtual Event, Greece

ACM ISBN 978-1-4503-8557-2/21/10…$15.00
https://doi.org/10.1145/3466752.3480123

CCS CONCEPTS

• **Computer systems organization → Special purpose systems**.

KEYWORDS

neural network, accelerator, bit-level sparsity

ACM Reference Format:
Hang Lu, Liang Chang, Chenglong Li, Zixuan Zhu, Shengjian Lu, Yanhuan Liu, and Mingzhe Zhang. 2021. Distilling Bit-level Sparsity Parallelism for General Purpose Deep Learning Acceleration. In *MICRO'21: 54th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO '21), October 18–22, 2021, Virtual Event, Greece*. ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3466752.3480123

1 INTRODUCTION

The deep learning accelerator performance is supposed to catch up with the ever increasing model size designed for higher accuracy, but on the other hand, the hardware designers are very reluctant to empower more computational resources in line with the evolution of deep neural networks (DNNs), due to some tight limits like battery life and power budget or even cost especially on embedded devices like autonomous robotics, drones and smartphones and so on. Boosting the accelerator efficiency is hence very desirable in both high-performance and power-efficient use cases. This work focuses on leveraging the abundant *bit-level sparsity parallelism* to accelerate both *training* and *inference* phases to serve the cloud/edge general-purpose deep learning.

The Landscape of Prior Sparsity-aware Work: Plenty of mechanisms [7, 31, 43, 49] and prototypes [16, 17, 28, 32, 47] in the literature seek to mine the maximum potential of weight/activation sparsity and execute the effectual multiply-and-accumulations (MACs)

in parallel for as many as possible. However, the sparsity is not always abundant but varies upon different models or even the individual layers within the same model. For instance, activation sparsity is more possible due to some non-linear activation functions, but weights, on the contrary, usually show scarce sparsity except for training with L1 norm. Even for the activations, the zero values are *only* likely to be spawned when they are passed through functions like ReLU or PReLu. The same situation exists in the model employing activation functions, such as sigmoid, ELU, leakyRelu, Gelu or Tanh. To resolve this challenge, some approaches therefore intend to identify the near "zero values" in the operand set [29, 30] or implement tedious sparse (re)training to create more sparsity for pruning [27, 31, 43, 48].

Recently, the community has noticed that headroom of exploiting the value-based sparsity has hit an end. On one hand, there exits a visible margin the compression ratio cannot overstep if the lossless accuracy is the first-order design constraint. No matter what pruning methodology is employed, plenty of time is spent on exploring such margin to balance the accuracy and the model size. On the other hand exploiting the value sparsity also inevitably induces more complex accelerator design. For example, enlarging its storage system to accommodate the growing size of the indices[17, 32, 46, 47] with the cost of the increased memory accesses and compromised peak computation throughput.

The "bit-level sparsity", on the contrary, is the inherently more fine-grained sparsity that targets the "zero bits" in each operand instead of the coarse-grained zero values. Using either floating-point or fixed-point precision to represent each weight or activation, the zero-bit percentage could attain 45% ∼ 77% in different vanilla DNNs as will be shown in the next section. Skipping zero bits in the operand will not affect the result, which also implies that the acceleration could be directly obtained without any software effort if effectual computation at the bit level is strictly enforced.

A series of bit-serial accelerators [8, 22, 26, 37] have been proposed to leverage the abundant bit-level sparsity, albeit to varying degrees. Figure 1 presents a high-level example to compare the computing paradigm of three types of accelerator PE. Apart from the early-stage bit-parallel accelerators [9, 12, 13, 19, 21] (Figure 1a), bit-serial prototypes use numerically identical bit-level arithmetic to compute the inner product. For example, decomposing one $8b \times 8b$ product into eight $1b \times 8b$ products yields identical result, by organizing and input the weights for MAC both in serial (Step ❶ in Figure 1b). However, this simple example also showcases an overwhelming problem. To release the maximum potential of the bit sparsity, it better skips the zero bits as many as possible. Whereas, the locality of the zero bits in each $8b$ operand is highly unpredictable, especially after fixed-point quantization. The reason is that quantization will fully utilize the limited bit width to represent the value range, making zero bits arbitrarily interleaved with the essential bit 1s. In order to fully leverage the inherently exposed bit sparsity, synchronization is imperative and must be carefully implemented as the Step ❷ shows in Figure 1b, before finalizing the bit-serial MACs in Step ❸.

Prior used synchronization methods include middleware-level dense scheduling (*i.e.*, Bit-tactical [26]) and hardware-level direct Booth Encoding (*i.e.* Laconic [36] and Pragmatic [8]) and so on. However, the key weakness of these approaches stems from the difficulty in determining a uniform pattern to describe the locality of the sparsity for synchronization. A direct consequence is that the ongoing MAC operation must be halted for aligning the bit significance with the cost of crippled throughput compared with the bit-parallel counterparts. For example in Figure 1(b), ${w_0}^2$ must wait until ${w_1}^0$ and ${w_1}^1$ have accomplished MAC, while ${w_1}^3$ must wait until ${w_0}^2$ has accomplished MAC. In terms of the hardware implementation, the complexity is also increased because Booth encoding needs additional circuits for encoding and storing the weight bits. As the incidental weakness, such serialized organization manner cannot support the floating-point arithmetic, or in other words, bit-serial accelerators cannot be deployed for general-purpose cases.

Our Focus: this work proposes a novel method to exploit the bit-level sparsity called "bit interleaving." It does not focus on the internal sparsity of an individual weight; instead, it leverages the *sparsity parallelism* exhibited by a series of weights to accelerate DNN inference. As shown in Figure 1c, it organizes the same number of weights in parallel (Step ❶), but interleaves the weights and implements bit-level MAC in serial (Step ❷) This is different from the existing bit serial/parallel accelerator design concept, from two dimensions: (1) compared with bit parallel accelerators, the actual bits used for computing the product are not original weights but interleaved weights; (2) apart from the bit serial accelerators, the serialization procedure is not limited within one weight, but extended to a series of interleaved weights along with each independent bit significance. Due to the above two features, bit interleaving is particularly appealing to DNNs in three aspects: (1) MAC computations exhibit behaviors that favor bit interleaving: the accumulation targets each independent bit significance, and no synchronization mechanism is necessary as in bit-serial accelerators. (2) it can be conveniently configured to support either floating-point or fixed-point precision, and is orthogonal to any quantization/pruning methodology. This becomes more important when quantization to lower fixed-point or integer precision does not work, and the inference has to turn back to floating-point arithmetic; (3) the "bit interleaving"-directed accelerator design could be used for both training and inference.

The contributions of this work are listed as follows:

- **We propose a novel DNN acceleration philosophy that fully exploits the bit-level sparsity, termed as *bit interleaving*.** Section 2 demonstrates the bit sparsity is not only high, as proved in recent literature[36], but also uniform at each bit significance (around 50%). Our findings corroborate that such even distribution persists: ① in both fixed-point & floating-point weights, and ② across big & little models. This *sparsity parallelism* motivates the feasibility and the necessity of the proposed bit interleaving concept.
- **We propose *Bitlet*, the associate hardware accelerator that maximally mines the potential of bit interleaving.** *Bitlet* serves as a general-purpose deep learning accelerator supporting both floating-point (fp 32/16) and fixed-point ($1 \sim 24b$) on one single platform. No matter what precision is used in practice, the sparsity parallelism could all be sufficiently exploited at each bit significance. Such versatility renders *Bitlet* could bring satiable efficiency not only in inference but also in training. As will be shown in Section 5, the maximum speedup of *Bitlet* could attain 15× over existing bit parallel/serial accelerators, and 81× over GPUs.

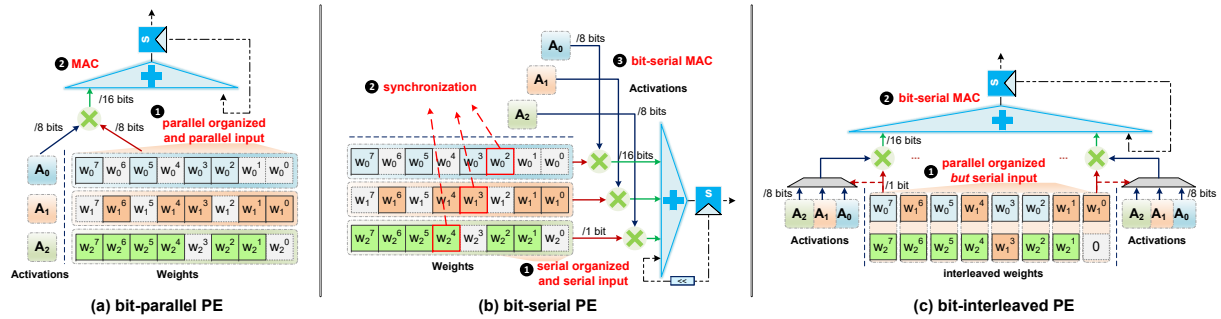


Figure 1: High-level step-by-step example comparing the bit-interleaved PE with prior bit-parallel/serial PE in the fixed-point mode. The w_i^j marked in grey is the non-essential bit (0 bit). In (a) bit-parallel PE, Step ❶ organizes the weights for MAC in parallel; Step ❷ issues MAC. In (b) bit-serial PE, Step ❶ organizes the weights in serial; Step ❷ synchronizes the significance of the essential bits; Step ❸ issues the "bit-serial" MAC. In (c) bit-interleaved PE, Step ❶ organizes the weights in parallel, but Step ❷ issues the bit-serial MAC along each bit significance, excluding the synchronization operation. Note that bit interleaving also supports the floating-point MAC, as will be specified in Section 3.

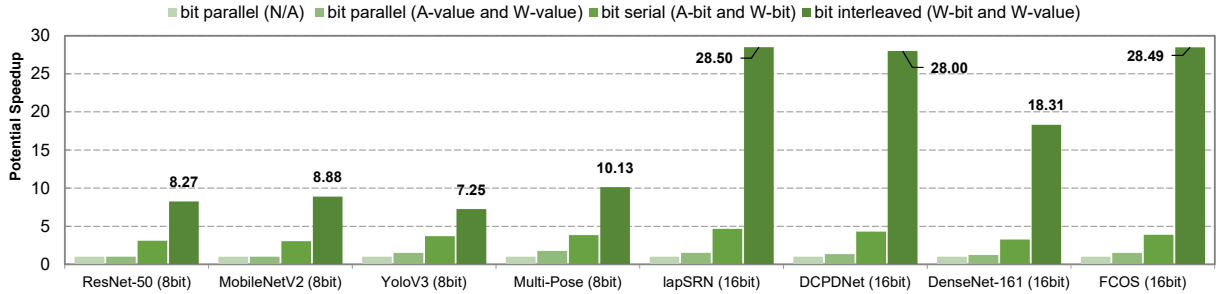


Figure 2: Potentials of bit interleaving. The baseline design is "bit parallel (N/A)". Most existing sparsity-aware accelerators only target fixed-point precision, so we only compare 16b and 8b DNNs in Table 2. In Section 5, we will evaluate the floating-point applications over GPUs.

Table 1: Accelerator design philosophies.

Philos.	Design	Sparsity Exploited	Preci. V.	Training Support
bit parallel	Eyeriss[10], DaDianNao[12]	N/A	16b	No
	Cambricon -S[47], EIE [17]	A-/W-value	16b	No
	SCNN[32]	A-&W-value	16b	No
bit serial	UNPU[27], Stripes[22]	N/A	1 ~ 16b	No
	Bit Fusion [37]	N/A	2,4,8,16b	No
	Pragmatic [8]	A-/W-bit	1 ~ 16b	No
	Bit Tactical[26]	A-bit&W-value	1 ~ 16b	No
	Laconic[36]	A-&W-bit	1 ~ 16b	No
bit inter -leaving	**Bitlet (this work)**	W-bit &W-value, (or A-bit&A-value)	$fp32/16$, 1 ~ 24b	Yes

2 MOTIVATION AND RELATED WORK

2.1 Rethinking Sparsity-aware Accelerators

In Table 1, we categorize the state-of-the-art sparsity-aware accelerators. In early-stage bit-parallel accelerators, *i.e.,* Cambricon series [46, 47] and SCNN [32], the sparsity is only focused on values. By collaborating with software-level pruning techniques, more headroom of zero values are created to release the potential of these accelerators. More recent bit-serial accelerators place the emphasis on the bit-level sparsity, considering the zero bits are inherently abundant in genetic weights or activations. The most recent Laconic[36] uses "terms" after Booth Coding to extract the essential bits serially and proposes a low-cost LPE to minimize the power increment caused by frequent encoding/decoding. Tactical[26] resolves the sparsity on the level of weight value and activation bit. The design concept is similar to Pragmatic[8], both of which uses zero-bit skipping ability to optimize the ineffectual product, but Tactical relies on a datatype agnostic front-end for skipping the zero weights and a software scheduler for maximizing the possibilities of weight skipping. There are also sparsity-agnostic prototypes that follow bit-serial computing manner. For example, Stripes[22] and UNPU[27] directly implements bit serialization for the fixed-point operands without sparsity avoidance. Bit-fusion[37] supports faster spatial and temporal composition to accelerate bit serialization but still does not exploit bit sparsity as well.

The strength of bit serial accelerators is the effectiveness of exploiting the sparsity in bits. However, the bit-serial accelerators provide comparatively lower throughput than its bit-parallel counterpart. On the top of two design philosophies, bit interleaving seeks to combine their pros and avoid their cons. Figure 2 shows the potential of *Bitlet* could attain 8× ~ 29× by exploiting the weight bit and value sparsity (W-bit and W-value in Table 1) for various AI tasks. Exploiting the activation sparsity is also applicable within *Bitlet*, depending on the on-spot data reuse policy.

Although fixed-point precision succeeds in efficient DNN inference, it leads that the accelerators designed for fixed precision can only implement inference, which makes these prototypes can hardly work for general purpose perspective. For example, the DNN training still relies on floating-point back propagation to guarantee

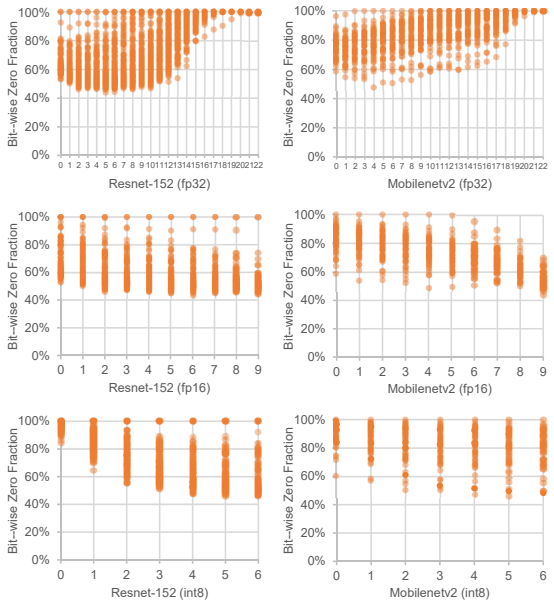


Figure 3: Sparsity parallelism. Each dot indicates the fraction of zeros on this bit lane across all the weights of this kernel. It shows ~ 50% bits are 0s for all kernels. On X-axis in the figure, the sparsity only entails the mantissa (23/10 bits for float 32/16), and 7 significant bits excluding the sign bit for int8 precision. We do not need to consider the sparsity of the exponential bits.

the model accuracy. Even in some edge-level use cases, the precision must be tuned back to the floating point but the real-time requirement still needs to be fulfilled, especially when the fixed-point precision cannot satisfy the accuracy for the in-progress AI application. Ideally, the accelerator should be applicable to most of the use cases, providing enough convenience and flexibility for the end-users in synergy. *Bitlet*, due to the bit interleaving design concept, could not only tackle sparsity effectively, but also support versatile precisions including both floating point and fixed point. The configurable feature renders it suitable for both high-performance and power-efficient scenario.

2.2 Leveraging the Sparsity Parallelism

Previous work has proved that the bit level sparsity is abundant. However, they solely focus on exploring the tactics of skipping zero bits inside a particular weight, while none of them explored the inter-weight sparsity, that is, the *sparsity parallelism* based on which bit interleaving could significantly outperform the bit serial/parallel prototypes.

As shown in Figure 3, we trace the bit sparsity for different convolution kernels, and observe that the weight sparsity at each significance is uniform. The X-axis indicates the bit significance of the mantissa, so we have 23 bits in total excluding the hidden bit 1 in standard fp32 format [15]. Each orange dot indicates the zero fraction on this bit index within one kernel. For the two DNNs `ResNet152` and `MobileNetV2`, it shows obvious aggregation at the first half of the mantissa (bit0~bit16), which means the amount of 0s and 1s that lie on this bit significance is nearly comparable. This provides a favorable condition to read the weights into the accelerator in parallel but compute the produce in serial along each bit lane. The independence of bit lanes is helpful to get rid of the cost of synchronization. As will be shown in Section 5, the computation of 64 MACs could be accomplished within one cycle in our FPGA platform for most of the evaluated DNNs.

Besides, from bit lane 17~23, the dots mostly overlap at 100% on the Y-axis (the long tail in the fp32 figures), which means most of the bits are 0. The floating-point multiplier does not distinguish this suboptimal scenario, because it is designed for covering any corner case of the operand. It is also the root reason that floating point MAC can hardly be accelerated. Whereas in our proposed *Bitlet* accelerator, such high sparsity could be easily exploited by bit interleaving. The details will be illustrated in the next section.

3 BIT INTERLEAVING

3.1 Theorem

Without losing generality, a floating-point operand is composed of three portions: signed bit (S), mantissa (M), and exponent (E), following IEEE-754[15], which is the most commonly used design standard in the industry. If we employ float-32 format (fp32 hereafter), the mantissa comprises 23 bits and the exponent occupies 8 bits with the last bit for the sign. A floating-point operand fp could be expressed as $fp = (-1)^s 1.m \times 2^{e-127}$, in which e is the actual position of the "binary point" plus 127. We consider a series of fp32 MACs in computing the partial sum of convolutions:

$$\sum_{i=0}^{N-1} A_i \times W_i = \sum_{i=0}^{N-1} (-1)^{S_{W_i}} A_i \times M_{W_i} \times 2^{E_{W_i}} \tag{1}$$

Explain Eq.1 first: we translate W_i into the normalized fp32 representation, in which M_{W_i} and E_{W_i} stands for $1.m_{W_i}$ and $e_{W_i} - 127$ for simplified expressions. Note that M_{W_i} includes the hidden bit – the first bit '1' in the mantissa, while for the actual storage in memory, this bit is hidden complied with IEEE-754. M_{W_i} is the mantissa and its width is fixed – 24 bits in total, so if we further decompose this M_{W_i}, we get the bit-represented partial sum:

$$\sum_{i=0}^{N-1} A_i \times W_i = \sum_{i=0}^{N-1} \sum_{b=0}^{-23} \left[(-1)^{S}_{W_i} A_i\right] \times 2^{E_{W_i}+b} \times M^b_{W_i} \tag{2}$$

$$= \sum_{i=0}^{N-1} \sum_{b=0}^{-23} \left[(-1)^{S_{W_i}} \bigoplus S_{A_i} \cdot M_{A_i}\right] \times 2^{E_{W_i}+E_{A_i}+b} \times M^b_{W_i} \tag{3}$$

where $M^b_{W_i}$ is the b-th bit of the binarized M_{W_i}. Replacing A_i with IEEE-754 binary format, Eq.2 could be rewritten as Eq.3. Furthermore, let $E_i = E_{W_i} + E_{A_i}$, then Eq.3 can be transformed as:

$$\sum_{i=0}^{N-1} \sum_{b=0}^{-23} \left[(-1)^{S_{W_i}} \bigoplus S_{A_i} \cdot M_{A_i}\right] \times 2^{E_i - E_{max}} \times 2^{E_{max}+b} M^b_{W_i} \tag{4}$$

$$= \sum_{i=0}^{N-1} \sum_{b=E_i-E_{max}}^{E_i-E_{max}-23} \left[(-1)^{S_{W_i}} \bigoplus S_{A_i} \cdot \left(M_{A_i} \times M^b_{W_i}\right)\right] \times 2^{E_{max}+b} \tag{5}$$

In Eq.5, we can infer that N number of fp32 MACs are equivalent to a series of bit-level operations of the corresponding mantissa. In specific, if $M^b_{W_i} = 1$, then the summation of N MACs is transformed

into the summation of N signed M_{A_i}(denoted by $(-1)^{S_{W_i} \bigoplus S_{A_i}}$) shifting $2^{E_{max}+b}$, contingent on the significance bit b and the E_{max}.

The above analysis conveys a fact: the floating-point partial sum could be turned into bit-level operations, with the sparsity considered. The product is primarily formed by the mantissa M_{A_i}, but whether it contributes to the product is decided by $M^b_{W_i}$ in Eq.5. Such bit sparsity could be also exploited in bit interleaving. There is a significant portion of 0s distributed at each bit significance, so if $M^b_{W_i} = 0$ but another weight W_j at the same significance b is an essential bit 1, we can allow $M^b_{W_j}$ taking the place of $M^b_{W_i}$, making different weight bits *interleaved* in the same lane. The mantissas M_{A_j} and M_{A_i} could contribute to the product within the same cycle, which accelerates the computations by exploiting the sparsity.

The theorem also embraces the fixed-point precision. In Eq.5, the impact of E_{max} and $E_i - E_{max}$ is unnecessary, because fixed-point values have no exponent. In the rest of this section, we describe how bit interleaving works for fp32 weights, and in the next section, we elaborate on how the proposed *Bitlet* accelerator is designed to support versatile precisions.

3.2 Procedures

Figure 1c exemplifies the procedures of bit interleaving for the 8-bit fixed-point MAC in a step-by-step manner. However, the floating-point MAC is not as easily harnessed as its fixed-point counterpart for the practical implementation, because there is an special portion - the exponent residing in the binary operand, and different operands have different exponents. In order to mine the maximum potential of the floating-point sparsity, bit interleaving involves three independent but consecutive steps based on Eq. 5:

3.2.1 Step ❶: Pre-Processing. Figure 4a illustrates as an example of 6 vanilla fp32 weights permuted in rows, each of which has an arbitrary exponent and mantissa. The triangle mark indicates the actual location of the binary point. For simplify, we do not represent the actual fp32 binary format stored in memory, but use the more iconic representations to express the value. For example, the $(0.01)_2$ with $E_5 = -2$ hence stands for 0.25 in decimal (W_5). This step is similar to the Step ❶ in Figure 1c, except that in here it organizes the fp32 weights in parallel for interleaving. It pre-processes these vanilla binary weights, in order to obtain the respective exponent and further determine the "maximum" exponent (E_6 in this example). The mantissa is also interpreted and stored for the later MAC. The rear bits (bit 9 23) of each mantissa is omitted for a simplified representation.

3.2.2 Step ❷: Dynamic Exponent Matching. The exponents denote the position of the binary point. Conventionally, it involves the step called "exponent matching" in the floating-point addition, to align the binary point of the two operands. In bit interleaving , we align a group of floats by matching their exponents uniformly to the maximum (E_6 in this example) instead of handling them one by one. This step is called "dynamic exponent matching", and is not involved in Figure 1c because fixed-point value has no exponent.

Looking back at Eq.5, the two $\sum$s could be executed in parallel during actual implementation: the outmost $\sum$ is on behalf of the vertical dimension in Figure 4a, that is, N number of weights with their associate activations; the innermost $\sum$ stands for the horizontal dimension which represents different bit widths of the mantissa. From this point of view, the key concept of Eq. 5 is to compute all the M_{A_i}s with $M^b_{W_i} = 1$, along the two dimensions in Figure 4a.

Since our final goal is to compute $\sum_{i=0}^{N-1} A_i \times W_i$, and it involves N number of weight and activation MACs. Therefore, this step intends to match all exponents to their maximum at each time instead of the costly one-by-one matching. As can be seen in Figure 4(b), the 6 weights are all aligned to the maximum exponent–w_6. For example, w_5 needs to shift 8 bit positions to the right to align with w_6. As a benefit, the exponent matching is issued only once for all the 6 weights, saving time and energy for efficient hardware implementation.

3.2.3 Step ❸: Essential Bit Distillation. For now, the problem is how to take advantage of the essential bits to obtain the accurate partial sum and further, the satisfied inference speed. Considering the sparsity parallelism as mentioned in Section 2, this step leverages this feature to *distill* the essential bits, and it is exactly identical to the Step ❷ in Figure 1c.

As shown in Figure 4c, if we effectively distill the essential bit 1s, the total computations could be reduced significantly from 6-operand MACs to only 3. Still taking W_6 as an example, its exponent is 6 and the first bit (b=0) is an essential 1. As inspired by Eq.5, the value of $2^{E_{max}+b}$ for this bit equals to 2^6, which means this bit is located in the $7th$ position before the binary point. For $W_1 \sim W_5$, the 2^6 positions are all zeros after exponent matching. If we could ascend the first bit of W_6 replacing the same position in W_1 in the same vertical lane, we are able to compute $A_6 \times 2^6 + A_1 \times 2^3$ simultaneously. The essential bits that belong to other weights could be operated in the same manner, and the distilled weights are finalized in Figure 4(c).

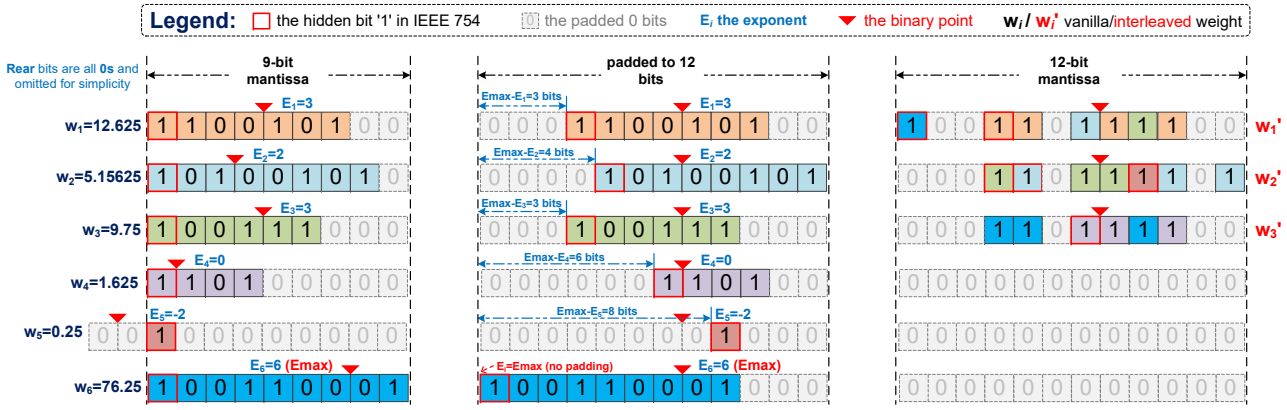
As summary, the two steps accelerate fp32 MACs from two aspects: (1) it eliminates the expensive exponent matching operations; and (2) it eliminates the useless computations caused by 0 bits by exploiting the sparsity parallelism. In Section 5, we will empirically show that the speed of fp32 MAC is even faster than the int8 or fixed-16 quantization on our accelerator platform. As the extra bonus, it directly accelerates the original DNNs without involving low-precision quantization and other software work.

4 BITLET ACCELERATOR

To enforce bit-interleaving, we design a novel accelerator, named as *Bitlet*. In this section, we present the key modules in *Bitlet*, including the micro-architecture of the compute engine with versatile precision support and the overall design for efficient memory access.

4.1 Bitlet Compute Engine

Key Module #1 - Preprocess. First and foremost, we design an engine responsible for the two steps in bit interleaving. *Bitlet* digests multiple input weights and activations, indicated by N in Figure 5. In *Bitlet* Compute Engine (BCE hereafter), W_0 through W_{N-1} are the original weights, while $A_0 \sim A_{N-1}$ are the corresponding activations. The *preprocessing module* decomposes each W_i and A_i into two fractions: mantissa and exponent, and performs $E_i = E_{W_i} + E_{A_i}$ for each A/W pair. Afterwards, the maximum exponent E_{max} is chosen and stored in the register for the subsequent dynamic matching phase. After nailing down the E_{max}, M_{W_i} is shifted by $E_{max} - E_{i_i}$ bits to align its exponent to E_{max}. Still taking the exemplified weights in Figure 4, E_{max} should be $E_6 = 6$ in W_6, other



(a) **Step 1:** preprocessing the floating-point weights

(b) **Step 2:** dynamic exponent matching

(c) **Step 3:** bit distillation

Figure 4: Core concept of "*bit interleaving*". Vanilla fp32 weights are exemplified in Step ❶. It pro-processes the weights by interpreting out the exponent E_i and mantissa M_i. According to the maximum exponent (E_{max}), the weights are shifted and zero padded in Step ❷. Note that the exponent matching is only allowed to the right hand side in case of severe precision loss as standardized in IEEE 754. Step ❸ is responsible for bit distillation. Only 3 interleaved weights are finally involved in the accelerator.

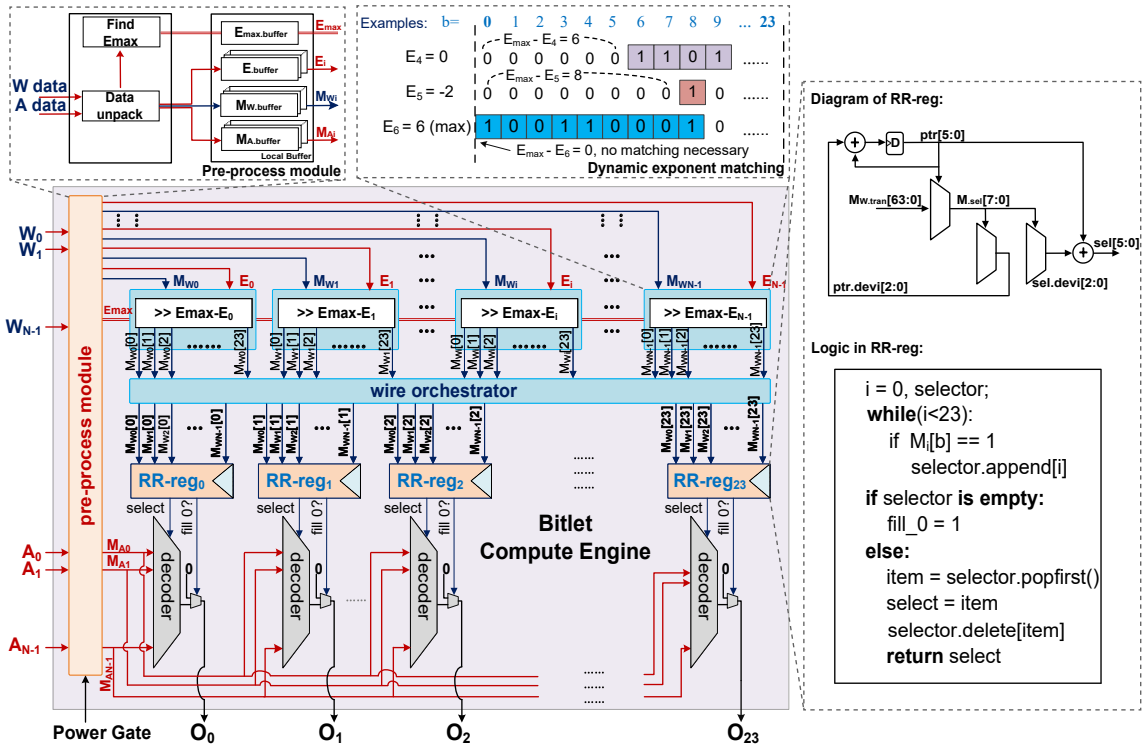


Figure 5: Microarchitecture of the core module – *Bitlet Compute Engine (BCE).*

weights are all aligned to E_6, *i.e.*, M_{W_4} will be shifted by $6 - 0 = 6$ positions to the right hand side as shown Figure 5. The shifted positions on the left hand side are padded by 0s automatically, and the rear bits beyond $b = 23$ are discarded because the mantissa is 24-bit length.

Key Module #2 - Wire Orchestrator. After dynamic matching, we obtain the 24-bit but shifted mantissa, indicated by $M_{W_i}[0] \sim M_{W_i}[23]$. They are further sent into another module called "Wire Orchestrator" in Figure 5, and it is used for or-organizing the wires by aggregating the same bit significance together. In particular, the output of the Orchestrator is represented as $M_{W_0}[b]$, $M_{W_1}[b]$, $\cdots$, $M_{W_{N-1}}[b]$ in which b is in range $0 \sim 23$. Note that this module does not contain any combinatorial logic or sequential logic. It just congregates wires for the aligned mantissas and performs the

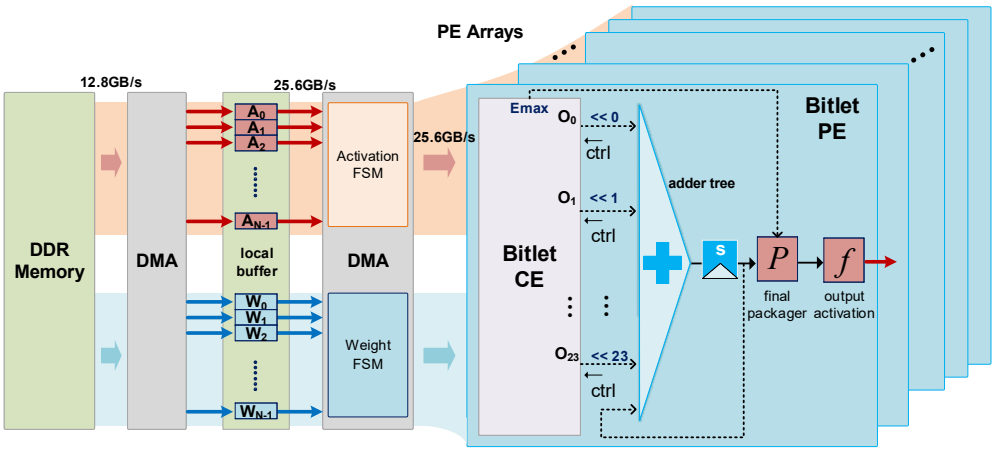


Figure 6: *Bitlet* accelerator. Each *Bitlet PE* is comprised of one BCE and a series of adders used for computing the output activations. *Bitlet* is versatile: for the floating point, Emax is dynamic, while for the fixed-point precisions, Emax is fixed to the target precision (*i.e.,* 16 or 8).

transpose operation, so intuitively this module will not introduce significant power consumption but increase the circuit area in some extents, which will be evaluated in Section 5).

Key Module #3 - RR-reg. $RR\text{-}reg_i$ distills the essence in the interleaved weights and selects the output of BCE from N activation mantissa. Each *RR-reg* has its internal clock connected with the accelerator clock tree. As shown in Figure 5, the pseudo-code represents the working procedure: it firstly wraps around the input bits and distills the essential bit in sequence, one after another. The "select" signal informs the decoder logic to configure the chosen activation path and output O_i. If no essential bit is detected, *RR-reg* activates the "fill 0" signal, and the output O_i will be 0 as well. This operation accommodates the scenario that all bits in a particular bit column are 0s, *i.e.,* $b = 1$ or 2 in Figure 4(c).

Discussions: We highlight three features of the BCE: ① the architecture will not introduce accuracy loss, because the dynamic exponent matching specified in Section 3.2.2 is identical to the floating-point arithmetic in IEEE 754, by discarding the rightmost outlier bits after shifting (">> $E_{max} - E_i$" in Figure 5). These bits are trivial bits with negligible significance so the accuracy is harmless. ② BCE does not require any pre-processing regarding the sparsity knowledge. The pre-processing module in Figure 5 is only responsible for splitting out the mantissa/exponent of each weight and activation. In the practical RTL implementation, we instantiate a sliding window in each RR-reg to automatically interleave and distill the essential bits. Benefited from the sparsity parallelism, the distillation of the essential $M_{W_i}[b]$ could be accomplished nearly simultaneously in each RR-reg. It also equips *Bitlet* with promising inference speed and low cost as proved in Section 5. ③ BCE is mostly composed of combinatorial circuits except for the *RR-reg*s, but it does not involve complex wires that might lead to prolonged critical path delay. Each *RR-reg* spawns one output O_i per clock cycle, but the total cycles spent on the partial sum are substantially optimized compared with the traditional one-by-one MAC. N is the only design parameter in *BCE*, and larger N is conductive to distill more bit 1s. In Section 5 we will empirically study its impact on the overall inference speed.

4.2 Accelerator Architecture

PEs. *Bitlet* is comprised of mesh-connected PEs. As shown in Figure 6, each PE is comprised of one *BCE* and one adder tree. *BCE* bridges the gap between the on-chip buffer and the adder tree. Each PE absorbs N weights and activations in tandem and spawns the partial sum O_i as the input of the adder tree. Since *BCE* outputs 24 bits limited by the mantissa, we also have 24 inputs to the adder tree.

It finalizes the result by multiplying $2^{E_{max}+b}$ (note that b is negative) to ensure the correctness of the result, which can be decomposed into a fixed part b and a common part E_{max} for BCE output. The fixed part of the exponent is computed by fixed amount shifting operated by wire connections in physical circuit. As for the common part, E_{max} is applied to the accumulator result in the final-packager module to produce a formatted result. Obtaining O_i only entails the fixed-point additions of the activation mantissa excluding any multiplications, which also means the arithmetic complexity and power consumption are also optimized accordingly.

Memory System. To achieve high throughput, the *Bitlet* accelerator provides separated DMA channels for the activation and weight data. As shown in Figure 6, the local buffer stores the data fetched from DDR3 memory and provides adequate bandwidth for the accesses from the corresponding *Bitlet PEs*. In our RTL implementation, the bandwidth achieves 12.8 GB/s per channel between the memory and the local buffer, while the PE Array can utilize totally 25.6 GB/s to get the activation and weight data from the local buffer. In terms of the dataflow mode, *Bitlet* leverages weight stationary and activation broadcasting [14] to minimize the main memory accesses.

4.3 Versatility

Bitlet is a versatile accelerator. It could be conveniently configured into fixed-point mode, providing enough flexibility for end-users. If we want to employ fixed-point 16 precision for example, we could handily power gate part of the preprocessing module that performs the exponent matching and shifting ("≫ $E_{max} - E_{W_i}$" in Figure 5), and let the input W_i directly connects to the Wire Orchestrator.

Bitlet is initially designed for the 24-bit mantissa, so if we use 16-bit fixed point for example, only $RR\text{-}reg_0 \sim RR\text{-}reg_{15}$ are involved. Others could be safely powered off or left idle. Similarly for int8 quantization or any other target precision (*i.e.,* int4, int9 *etc.*), *Bitlet* handles it in the same manner. Therefore, end-users no longer have to resort to other precision-specific accelerators to accommodate different use cases. They could freely calibrate their DNNs to fulfill the accuracy target as well as the power/performance tradeoff on one platform.

5 EVALUATION

5.1 Experiment Setup

DNN Models & Application Baselines. In this section, we evaluate the proposed bit interleaving methodology and the *Bitlet* accelerator. *Bitlet* is a general-purpose accelerator that covers mainstream floating-point and $1 \sim 24$bit fixed-point precision, which also benefits the diversity of the applications that could run on *Bitlet*. Therefore, we select 12 domain-specific AI applications with diverse model structures, GFLOPs, parameter sizes and precisions, as listed in Table 2. According to the domain, the DNNs are trained with the corresponding dataset with PyTorch framework[33]. We quantize the initial fp32 weights into fixed-16 and int8 precision. In order to prove the versatility of *Bitlet*, we choose a series of baselines: (1) *GPUs*, including high-performance datacenter product – Titan V (volta) and Titan Xp (pascal) and power-efficient edge-level product – Jetson TX2 (pascal). (2) *Sparsity-agnostic accelerators*, typical accelerator based on the fixed-point MAC - Eyeriss[14], and it employs multipliers and adders both designed for fixed-16 values without considering the sparsity; and typical bit-serial accelerators - Stripes[22]. This design does not exploit the abundant bit-level sparsity. (3) *Sparsity-aware accelerators*, we adopt value-sparsity aware prototype SCNN[32] and bit-sparsity aware prototype Laconic[36] as the representative. For *Bitlet*, we evaluate various flexible configurations, including "*Bitlet (float 32)*", "*bitlet (16b)*" and "*Bitlet (8b)*". For the key design parameter N, we evaluate several typical settings $(2, 4, 8, 16, 32, 64)$ to analyze its impact on the speedup. A particular *hardware ablation study* is presented to test the performance sensitivity to the individual modules in *BCE* (*i.e.,* preprocessing, RR-reg), also under various N settings.

FPGA & ASIC implementation. At the RTL level, we employ Vivado (v2018.2) to conduct post-synthesis simulation on Xilinx Virtex-7 FPGA. The inference time in *frames per second* (fps) is recorded at each run. We instantiate 32 PEs clocked at 200MHz, with 1 *BCE* in each PE. Runtime memory access data of our FPGA platform are recorded and then feed to the DRAMsys tool[23] to estimate the energy consumption of the memory accesses. For ASIC, Synopsis Design Compiler (v2016) is used to measure power and area. The frequency is set to 1 GHz. Our design is synthesized with both TSMC 28*nm* and 65*nm* technology library. We decompose the *Bitlet* PE to report detailed module-level area. The baseline area numbers are directly reported from their publications, but only the total PE area is available. .

5.2 Specifics Comparison

This subsection compares the specs of the SOTA accelerator prototypes and GPUs with *bitlet*. The spec items in Table 3 are directly reported according to the published literatures. Under 32 PEs/BCEs, *Bitlet* obtains a 204.8 GOPs, 372.35 GOPs 744.7 GOPs peak performance and 359.15 GOPs/W, 667.97 GOPs/W, 1335.93 GOPs/W peak efficiency under 28 nm technology (111.97 GOPs/W, 267.87 GOPs/W, 621.10 GOPs/W under 65nm technology) for floating-point 32, 16b and 8b precision, respectively. For GPUs, Titan V has the highest performance but its efficiency is undermined by the high power consumption. Although *Bitlet* has comparatively the least PEs, it achieves the most optimal area and efficiency under 28*nm* technology node. Note that the "PEs/Cores" in the table is the standard configuration, based on which the peak power efficiency and performance are reported. However, in the subsequent performance and energy evaluation, we use the same number of PEs with nearly identical computational resources to give a fair comparison.

5.3 Speedup and Energy Efficiency

Speedup. We first analyze the acceleration. As shown in Figure 7, we use the actual inference time (in fps) measured on our FPGA platform as the representative of the speedup. We record the real values in *frames per second* (fps) for comparison, so the Y-axis has different scales in each sub-figure. For example, DenseNet-161 can reach 2.8 fps on *Bitlet*, Whereas on other accelerator baselines, the results are 0.187 (15.03×), 0.228 (12.33×), 0.426 (6.6×), and 0.611 (4.6×). As the representative of bit-serial accelerator, Stripes supports layer-wise tunable precision from $1 \sim 16b$, tested offline based on the minimum acceptable accuracy loss. SCNN however fixes the precision to 16b in our evaluation and relies highly on the value-sparsity condition. Pruning is not implemented on our benchmarks so it performs worse than Stripes.

A more interesting observation is the fps result of *Bitlet* in float-32 mode. It shows 2.53 fps for DenseNet-161 which is even faster than all the fixed-point baselines. For other applications across 16b and 8b, the results are similar. Such benefit stems from the bit-interleaving philosophy. By distilling the sparsity in parallel not in serial, floating-point MAC also enjoys the acceleration by exploiting the bit-level sparsity. Therefore, a more important conclusion is that the end users could directly acquire abundant acceleration on *Bitlet* using vanilla DNNs after training, instead of having to quantize their models to the fixed-point precision in a labor-intensive and time-consuming manner.

Table 4 presents the performance of executing one MAC operation. The reported data includes the pre-processing and the post-processing time. *Bitlet* representatives $22 \sim 29$ cycles/MAC across 16b and 8b. The 16b *Bitlet* behaves nearly comparative with the 8b *Bitlet*. It is reasonable because the bit sparsity exhibits nearly uniform distribution at each significance, even if the bit length is different. This feature, again, confirms that *Bitlet* serves as a general-purpose accelerator that could cover any fixed-point precision.

Energy Consumption. Figure 8 reports the energy consumption normalized to the "*Bitlet* (float 32)". The largest gap emerges at ResNet-50, who shows 24.31× more energy consumption. Generally speaking, SCNN does not perform well in this set of experiment. The reason is that SCNN tackles value-level sparsity, which means the speedup benefits only when the sparsity is fertile, so the accelerators designed for value sparsity have inherent limitations; it must rely on software approaches to assist the practical use. However, the 12 applications do not go through sparse training or pruning. They are all original DNNs. Therefore, the accelerator designed for

Table 2: Benchmark DNNs and their specs, which are used for motivating bit interleaving and the evaluations.

Models	Type	Precision	Domain	Dataset	GFLOPS	Weights	W-bit Sparsity (%)
ResNet-50[18]	2D Convolution	8 bit	Image Classification	ILSVRC'12[3]	8.21	25.56M	70.15 (fixed point)
MobileNetV2[35]	2D Convolution	8 bit	Image Classification	ILSVRC'12[3]	0.615	3.49M	76.85 (fixed point)
YoloV3[34]	2D Convolution	8 bit	Object Detection	CoCo[1]	25.42	61.95M	77.78 (fixed point)
Multi-Pose[24]	2D Convolution	8 bit	Pose Estimation	CoCo[1]	97.55	59.59M	66.33 (fixed point)
lapSRN[25]	2D De-Convolution	16 bit	Image Super Resolution	SET14[4]	736.73	0.87M	74.31 (fixed point)
DCPDNet[45]	Encoder -Decoder	16 bit	Deraining /Dehazing	NYU-Depth[38]	254.37	66.9M	75.00 (fixed point)
DenseNet-161[20]	2D Convolution	16 bit	Image Classification	ILSVRC'12[3]	15.56	28.68M	68.92 (fixed point)
FCOS[39]	Feature Pyramid	16 bit	Object Detection	CoCo[1]	80.14	32.02M	70.83 (fixed point)
CartoonGAN[11]	GAN	float 32	Style Transfer	flickr[2]	108.98	11.69M	48.49 (floating point)
Transformer[41]	Seq2Seq	float 32	Word Embedding	wmt'14[6]	10.6	176M	45.75 (floating point)
C3D[40]	3D Convolution	float 32	Video Understanding	UCF101[5]	38.57	78.41M	45.83 (floating point)
D3DNet[44]	3D Deformable	float 32	Video Super Resolution	Vimeo-90k[42]	408.82	2.58M	47.69 (floating point)

Table 3: Accelerator specs comparison. The items are directly referred according to the published literatures. We list them here to give a straightforward comparison of the key metrics between bitlet and others, *i.e.,* PEAK Performance in GOPs, GOPs/W etc. "–" means this datum is not able to be referred.

	Accelerator ASICs						GPUs		
Chip	Eyeriss [14]	SCNN [32]	Stripes [22]	Laconic [36]	**Bitlet (Ours)**		Titan V	Titan Xp	Tegra X2
PEs/Cores	168	64	4096	192	**32**		5120	3840	256
Precision	16b	16b	1~16b	1~16b	**fp32/16, 1~24b**		fp32/16, 8b	fp32, 8b	fp32/16
Technology	65nm TSMC	16nm TSMC	65nm TSMC	65nm TSMC	**28nm TSMC**	**65nm TSMC**	12nm TSMC	16nm TSMC	16nm TSMC
Freq. (MHz)	250	1000	980	1000	**1000**		1455	1582	854
PEAK Performance (GOPs)	23.1	2000	–	–	**204.8 (fp32) 372.35 (16b) 744.7 (8b)**		14900 (fp32) 29800 (fp16)	12150 (fp32)	750.1(fp32) 1330 (fp16)
Power	278mW	–	–	–	**570mW(fp32) 432mW(16b) 366mW(8b)**	**1829mW(fp32) 1390mW(16b) 1199mW(8b)**	250W	250W	15W
PEAK Power Efficiency (GOPs/W)	83.09	–	–	441 (16b) 805 (8b)	**359.15 (fp32) 667.97(16b) 1335.93 (8b)**	**111.97 (fp32) 267.87 (16b) 621.10 (8b)**	59.6(fp32) 119.2(fp16)	48.6 (fp32)	50.0 (fp32) 88.7(fp16)
Area (mm^2)	12.25	7.9	122.1	1.59	**1.54**	**5.80**	–	–	–

Table 4: Quantitative comparison for average computation performance (cycles/MAC).

	16b				8b			
	lapSRN	DCPDNet	DenseNet-161	FCOS	ResNet-50	MobileNetV2	YoloV3	Multi-Pose
Eyeriss	339.34	357.39	343.68	346.62	344.08	343.77	345.08	341.71
SCNN	226.22	262.09	281.87	231.08	344.08	343.77	230.05	197.14
Stripes	90.49	106.25	150.86	115.54	158.39	174.07	119.94	110.23
Laconic	71.44	83.64	105.18	89.77	111.13	113.44	93.22	89.92
Bitlet	29.51	16.45	22.85	22.44	21.03	22.22	19.04	24.64

indexing the sparsity for zero skipping cannot fully contribute to the speedup improvement but still consumes significant amount of power. *Bitlet* (16b) has the lowest energy consumption. Selecting some of the data to report: 28.1% for DCPDNet, 28.3% for YoloV3 and 36.7% for MobileNetV2.

Energy Breakdown. Our Xilinx-V7 FPGA platform involves DDR3 memory, we use DRAMsys to estimate the runtime memory access energy together with the PE computation energy. Figure 9 shows the energy breakdown from two aspects. Figure 9(a) shows the full-system energy breakdown and obviously the memory accesses dominate the energy consumption. Especially for Transformer, the data attains nearly 99% and the PE computation energy occupies only 1%. In Figure 9(b), we further decompose the PE-only energy for each DNN. Preprocessing module dominates this time (63.2%), because it involves a large number of buffers to store the mantissa and exponent. For other modules, *Wire Orchestrator* and adder tree consume 14.7% and 6.27% energy on average, respectively.

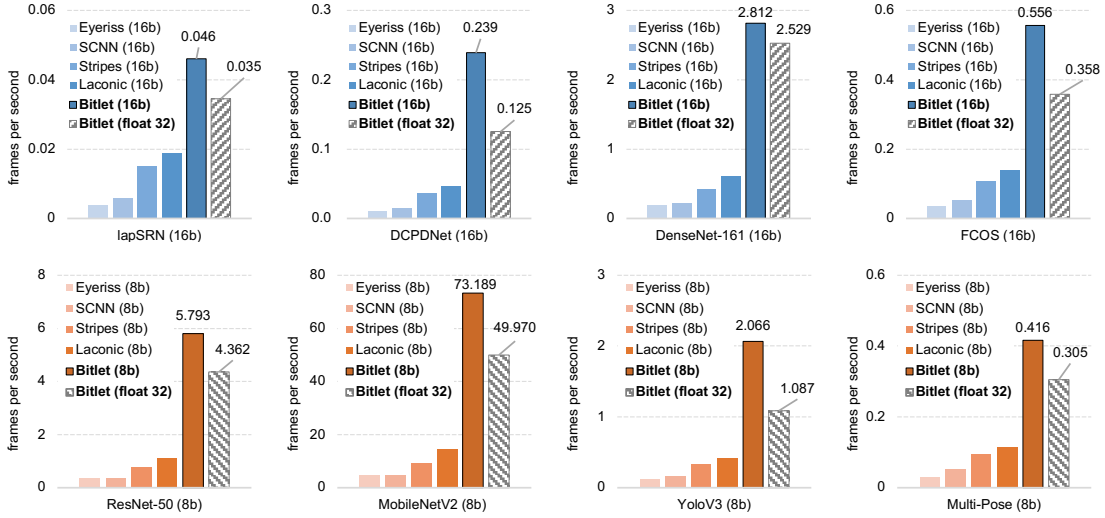


Figure 7: Speedup results. The upper row denotes the 16b DNN benchmarks and the bottom row denotes the 8b benchmarks. We also run the float-32 version on *bitlet* for reference. All the results are real values in frames per second (fps). Higher is better.

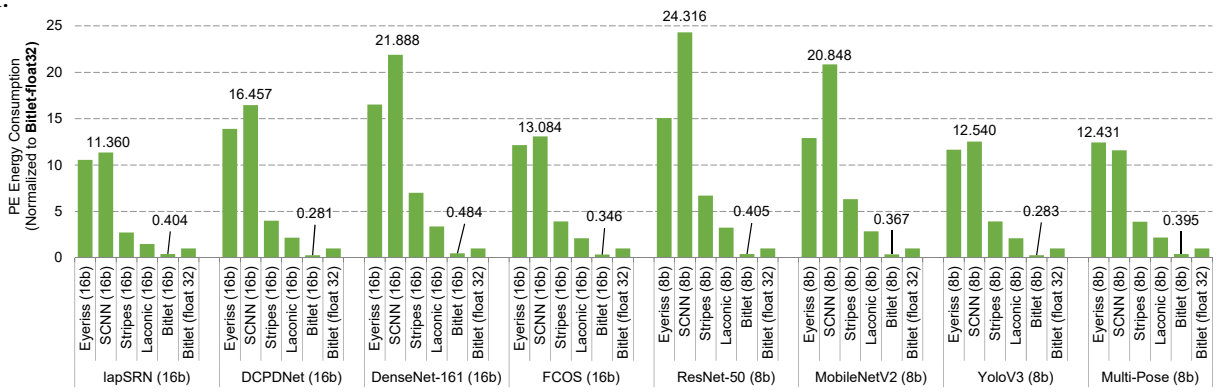


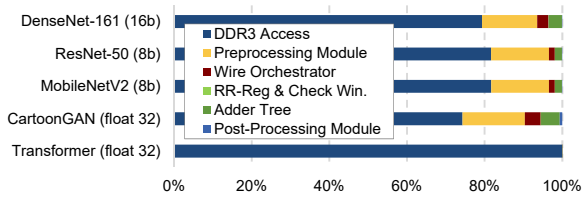
Figure 8: Energy Consumption. We also report the energy result of the float-32 inference, and all the fixed-point results are normalized to it. Lower is better.

Table 5: Training/Inference efficiency of the float-point applications. Since the employed accelerator baselines do not support floating point, we compare *Bitlet* with GPUs. Note: TX2 is not used for training.

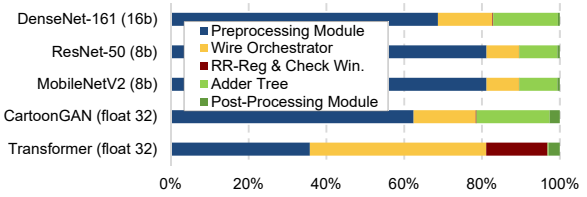
GPU Baselines	Models (**Train** / **Inference** efficiency in GOPs/W)			
	Cartoon -GAN	Trans -former	C3D	D3DNet
Titan V	0.27 / 3.19	4.09 / 21.80	1.82 / 4.32	0.06 / 4.67
Titan Xp	0.20 / 2.36	3.03 / 16.13	1.35 / 3.19	0.04 / 3.46
Jetson TX2	−− / 0.20	−− / 1.39	−− / 0.27	−− / 0.30
***Bitlet* (float 32)**	9.40 / 67.29	7.36 / 69.97	8.59 / 66.04	4.87 / 60.24

Efficiency. *Bitlet* is a versatile, general-purpose accelerator, supporting the floating-point and 1 24b fixed-point precisions. Since the ASIC baselines do not support the floating-point arithmetic, we compare the power efficiency of *Bitlet* with the GPU baselines, in terms of four deep learning applications – CartoonGAN, Transformer, C3D and D3DNet. Table 5 reports the data including both training and inference efficiency. *Bitlet* shows 34.81×, 1.80×, 4.72×, and 81.16× improvement over the datacenter product - Titan V. Under similar evaluation method, the corresponding inference efficiency improvement is 21.09×, 3.21×, 15.29×, and 12.9×. We can observe an obvious gap between the training and inference efficiency in Table 5. That is because the training phase involves forward and backward propagation, and the backward propagation however, is not able to be accelerated by Bitlet. The actually-effective acceleration is targeted and recorded towards the forward propagation only, so the GOPs/Watt data are relatively smaller.

Table 6 and Table 7 show the results compared with fixed-point accelerator baselines, and only inference efficiency is compared because the baselines do not support training. For 16b applications, the improvement over the most recent Laconic is 3.67×, 7.69×, 6.97× and 6.05×. Even for *bitlet* (float 32), it behaves better than all the baselines. This confirms that bit interleaving is more powerful than the bit-serial/parallel computing philosophies. If end users are reluctant in quantizing their DNNs, directly deploying the floating



(a) Full system energy breakdown.



(b) PE-only energy breakdown.

Figure 9: Energy breakdown.

Table 6: Inference efficiency of the 16b applications.

Accelerators Baselines	Inference efficiency is in GOPs/W			
	lapSRN	DCPDNet	DenseNet -161	FCOS
Eyeriss (16b)	9.92	9.42	9.79	9.71
SCNN (16b)	24.29	20.96	19.49	23.77
Stripes (16b)	25.06	21.34	15.03	19.63
Laconic (16b)	46.04	39.32	31.27	36.64
***Bitlet* (16b)**	168.75	302.71	217.87	221.87
***Bitlet* (float 32)**	44.64	55.82	69.03	50.33

Table 7: Inference efficiency of the 8b applications.

Accelerators Baselines	Inference efficiency is in GOPs/W			
	ResNet-50	Mobile -NetV2	YoloV3	Multi -Pose
Eyeriss (8b)	9.79	9.80	9.76	9.85
SCNN (8b)	15.97	15.98	23.88	27.87
Stripes (8b)	14.32	13.03	18.91	20.57
Laconic (8b)	29.60	29.00	35.29	36.58
***Bitlet* (8b)**	236.82	224.13	261.50	202.07
***Bitlet* (float 32)**	62.83	53.92	48.46	52.24

point model on *Bitlet* will also bring satiable acceleration. Similar results are also exhibited for the 8b mode.

5.4 Hardware Ablation Study

We carry out an ablation study in this subsection to explore the impact of each hardware module on the fps. This is to pinpoint the source of the performance improvement at the hardware level. In the ablation study, the target modules include the preprocessing module that performs dynamic exponent matching (termed as "**M**") and the RR-reg with check window that performs bit distillation (termed as "**D**"). We have 4 scenarios in total as shown in Table 8:

(1) If we remove "M" and "D" in tandem (w/o M, w/o D) in *Bitlet*, it is a bare-metal design which is the same as setting N to 1. This scenario could be configured in both floating- and fixed-point *Bitlet*.
(2) If we reserve "M" and remove "D" (w/ M, w/o D), it deteriorates from the sparsity-aware to the sparsity-agnostic design, because the distillation phase is disabled and the ineffectual zero bits are still involved in computation. This only happens in the floating-point *Bitlet*.

Table 8: Hardware ablation study. "bare-m" refers to "bare-metal", "M" refers to dynamic exponent "Matching", and "D" refers to essential bit "Distillation". The results are in "fps", and higher is better.

Instance	bare-m	***bitlet-fp32***			
Parameter	$N = 1$	$N = 32$		$N = 64$	
Ablation	w/o M w/o D	w/ M w/o D	w/ M w/ D	w/ M w/o D	w/ M w/ D
CartoonGAN	0.012	0.209	0.269	0.244	0.352
Transformer	0.126	2.152	2.879	2.509	3.763
C3D	0.035	0.590	0.771	0.670	0.976
D3DNet	0.003	0.056	0.068	0.065	0.084
Instance	bare-m	***Bitlet-16b***			
Parameter	$N = 1$	$N = 32$		$N = 64$	
Ablation	w/o M w/o D	w/o M w/o D	w/o M w/ D	w/o M w/o D	w/o M w/ D
lapSRN	0.004	0.033	0.040	0.038	0.046
DCPDNet	0.011	0.096	0.184	0.109	0.239
DenseNet-161	0.187	1.567	2.260	1.779	2.812
FCOS	0.036	0.304	0.455	0.345	0.556
Instance	bare-m	***Bitlet-8b***			
Parameter	$N = 1$	$N = 32$		$N = 64$	
Ablation	w/o M w/o D	w/o M w/o D	w/o M w/ D	w/o M w/o D	w/o M w/ D
ResNet-50	0.354	2.970	4.598	3.371	5.793
MobileNetV2	4.730	39.644	60.001	45.001	73.189
YoloV3	0.114	0.959	1.640	1.089	2.066
Multi-Pose	0.030	0.250	0.346	0.284	0.416

(3) If we remove "M" and reserve "D" (w/o M, w/ D), *Bitlet* can only work at the fixed-point mode, because only this mode does not need exponent matching (no exponent in fixed-point values).
(4) If we reserve "M" and "D" in tandem (w/ M, w/ D), it is the standard prototype of the floating-point *Bitlet*.

We highlight two observations in the ablation results. First, generally speaking, all *Bitlet* instances perform better than bare-metal. For example, the speedup of *Bitlet* (float 32, w/ M w/ D) is 22.41× ($N = 32$) and 29.33× ($N = 64$) for CartoonGAN. Similar to *Bitlet* (8b, w/ M w/ D), it shows 12.69× ($N = 32$) and 15.47× ($N = 64$) speedup. Secondly, within *Bitlet* instances, larger N always leads to better fps results, no matter if "D" is set or not. Taking DCPDNet as the example, the speedup of $N = 64$ over $N = 32$ is 1.14× and 1.30× for w/o D and w/ D respectively. While in the same N configuration, the w/D over w/o D is 1.92× and 2.19× for $N = 32$ and $N = 64$ respectively. Upon the two observations, it concludes here that larger N and setting up "D" will both bolster the inference speed, but bit distillation ("D") is the major drive across all the precision and applications studied.

5.5 Sensitivity of Key Design Parameters

As concluded in the ablation study, larger N leads to better fps result. This experiment aims at the N design space exploration, and tries to find the best N configuration in terms of the inference speed. N controls the stride that weights could be interleaved in Figure 4(c). Intuitively, if we set N as large as possible, the number of weights that are simultaneously absorbed by *BCE* is also increased. It also provides a larger possibility for distilling the essential bits. As shown in Figure 10, we record the task completion time at each N and scale N from 2 to 64 (power of 2 at each step), the performance increases nearly exponentially for some of the applications, *i.e.,* YoloV3, DCPDNet and Transformer, and linearly for ResNet-50, Multi-Pose and lapSRN.

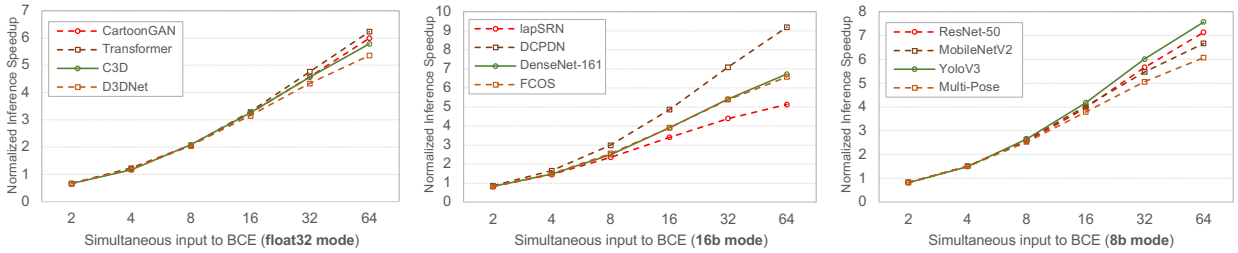


Figure 10: Sensitivity study of the key design parameter N.

Besides, we need to mention that larger N will not burden power consumption. N only decides how many MACs could be simultaneously executed by each PE, so increasing N does not mean the on-chip local buffer also needs to be expanded. If the memory access throughput could perfectly match the PE computing throughput, N is the larger the better. That is why we have selected $N = 64$ as the default configuration in previous experiments.

5.6 Scalability

Figure 11 shows the PE scaling from 8,16 to 32 with respect to the accelerator performance, normalized to PE=8. $Transformer$ is more memory intensive so the performance scales 2.41× under float 32 mode. Other benchmark DNNs are more computation intensive so more PEs are beneficial to the performance enhancement; for example, ResNet50 attains 3.85× speedup for 32 PEs under 8b precision. Minimized data precision, generally speaking, is helpful to the minimized memory accesses, so fixed-precision DNNs more possibly exhibit higher performance when PE scales larger.

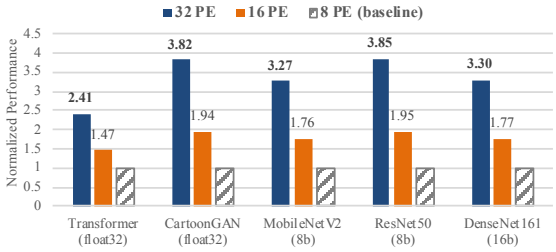


Figure 11: Sensitivity study to the PE array scale.

5.7 Area and Power Breakdown

Table 9: Area and power breakdown. (@TSMC 28nm)

Item	*Bitlet*(float 32) Area (mm^2)	*Bitlet*(float 32) Power (mW)	*Bitlet*(16b) Power (mW)	*Bitlet*(8b) Power (mW)
Preprocessing Module	0.553 (35.8%)	356.8 (62.6%)	296.598 (68.6%)	296.598 (81.2%)
Wire Orch. & Decoder	0.570 (35.9%)	63.857 (11.2%)	40.28 (9.73%)	20.651 (5.54%)
RR-Reg & Check Win.	0.131 (9.6%)	27.37 (4.8%)	20.28 (4.56%)	10.128 (2.88%)
Adder Tree	0.244 (15.8%)	107.424 (18.8%)	71.616 (16.6%)	35.808 (9.80%)
PostProcessing Module	0.044 (2.9%)	14.88 (2.6%)	2.304 (0.53%)	2.304 (0.63%)
Total	**1.542**	**570.15**	**432.08**	**365.49**

Table 10: Area and power breakdown. (@TSMC 65nm)

Item	*Bitlet*(float 32) Area (mm^2)	*Bitlet*(float 32) Power (mW)	*Bitlet*(16b) Power (mW)	*Bitlet*(8b) Power (mW)
Preprocessing Module	1.916 (33%)	1208.5 (66.1%)	1000.8 (71.9%)	1000.8 (83.5%)
Wire Orch. & Decoder	2.327 (40.1%)	164.1 (8.1%)	111.4 (8.0%)	55.7 (4.6%)
RR-Reg & Check Win.	0.7 (12.0%)	112.1 (7.2%)	74.8 (5.3%)	37.4 (2.9%)
Adder Tree	0.7 (12.0%)	293.3 (16.0%)	195.5 (14.1%)	97.8 (9.8%)
PostProcessing Module	0.2 (2.9%)	48.5 (2.7%)	7.4 (0.5%)	7.4 (0.6%)
Total	**5.8**	**1829.6**	**1390.0**	**1199.1**

Under 28nm TSMC technology node, *Bitlet* in float 32 mode exhibits 1.542mm^2 for the 32-PE prototype (5.802mm^2 with the 65nm technology node). Table 3 compares the area of the state-of-the-art accelerators, and *Bitlet* costs the smallest circuit area. Within *Bitlet*, Table 10 illustrates the largest floor space is occupied by the "*Wire Orchestrator & Decoder*" in *BCE* (40.1%), because the decoder and some of the wires reorganized are inevitably prolonged to avoid intersection. However, it is not the largest power consumer (only 11.2%), because no complex computation circuit are involved in this module. The "preprocessing module" consumes the largest power quota (62.6%) followed by "adder tree" (18.8%). By comparing the preprocessing module power, processing floating point data costs less power than the fixed point data, but the portion over total power increases from 62.6% to 81.2%. For the *Wire Orchestrator*, lower precision consumes less power as well. The point we want to emphasize is, from float-32 to 16b and 8b, *Bitlet*'s power consumption continues to scale down, which means the design is highly power scalable. However, looking back at the bit-serial accelerators, the power consumption remains unchanged for any fixed-point precision, because the bit-serial computing does not distinguish the precisions during serialization. *Bitlet*, again, is able to provide enough flexibility for the end users to select the best power/performance investment in practice.

6 DISCUSSION

The application scope of Bitlet. The Bitlet primarily accelerates the inference procedure. However, since the training procedure also includes the forward propagation, which is actually also the inference procedure. Therefore, the Bitlet could provide acceleration for both training and inference.

The importance of versatile precision support. The Bitlet focus on the general-purpose acceleration for various machine

learning applications. However, the different application scenarios requires various data types and precision support (see Table 2). Therefore, it is important to provide multi-precision support (i.e., $fp32/16$ and fixed-point $1b \sim 24b$) in one accelerator architecture.

7 CONCLUSION

In this paper, we propose a novel bit-level approach for general-purpose deep learning acceleration - "*bit interleaving*", and the corresponding accelerator design - "*Bitlet*". It leverages the sparsity parallelism in the parameters and implements "dynamic exponent matching" and "essential bit distillation" to circumvent the useless computations that would potentially drag down the inference speed. *Bitlet* is versatile by simultaneously supporting the floating-point (fp32/fp16) and fixed-point (1 ∼ 24b) precision. The users could test their models at any precision on a single platform to explore the best accuracy/speedup/power tradeoff, saving time and effort for the faster deployment. We believe that the techniques proposed in this paper can provide new opportunities for the researchers to explore novel applications in deep learning and even the algorithms beyond AI. We also hope it will probably inspire new ideas on the deep learning accelerator design, by applying the same concept in conjunction with some optimization techniques like pruning, and on other hardware platforms (*i.e.,* GPGPUs) in the future.

ACKNOWLEDGMENTS

This work is supported in part by National Natural Science Foundation of China grants No.62002339, No.62172387 and No.62104025, the Strategic Priority Research Program of the Chinese Academy of Sciences under grant No. XDB44030200, the NSAF under grant NO. U2030204 and the Key Research Program of State Key Laboratory of Computer Architecture under grant No. CARCH5301 and CARCH4506.

REFERENCES

[1] [n. d.]. COCO Dataset. https://cocodataset.org/#download
[2] [n. d.]. Flickr Image dataset. https://www.kaggle.com/hsankesara/flickr-image-dataset
[3] [n. d.]. ImageNet Large Scale Visual Recognition Challenge. http://www.image-net.org/challenges/LSVRC
[4] [n. d.]. Set 14. https://github.com/jbhuang0604/SelfExSR
[5] [n. d.]. UCF101 – Action Recognition Data Set. https://www.crcv.ucf.edu/research/data-sets/ucf101
[6] [n. d.]. WMT Dataset. http://data.statmt.org
[7] B. Ahn and T. Kim. 2020. Deeper Weight Pruning without Accuracy Loss in Deep Neural Networks. In *2020 Design, Automation Test in Europe Conference Exhibition (DATE)*. 73–78.
[8] Jorge Albericio, Alberto Delmás, Patrick Judd, Sayeh Sharify, Gerard O'Leary, Roman Genov, and Andreas Moshovos. 2017. Bit-pragmatic deep neural network computing. In *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*. 382–394.
[9] Y. Chen, T. Krishna, J. Emer, and V. Sze. 2016. 14.5 Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. In *2016 IEEE International Solid-State Circuits Conference (ISSCC)*. 262–263.
[10] Y. Chen, T. Krishna, J. S. Emer, and V. Sze. 2017. Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks. *IEEE Journal of Solid-State Circuits* 52, 1 (2017), 127–138.
[11] Y. Chen, Y. Lai, and Y. Liu. 2018. CartoonGAN: Generative Adversarial Networks for Photo Cartoonization. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 9465–9474.
[12] Y. Chen, T. Luo, S. Liu, S. Zhang, L. He, J. Wang, L. Li, T. Chen, Z. Xu, N. Sun, and O. Temam. 2014. DaDianNao: A Machine-Learning Supercomputer. In *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture*. 609–622.
[13] Y. Chen, T. Yang, J. Emer, and V. Sze. 2019. Eyeriss v2: A Flexible Accelerator for Emerging Deep Neural Networks on Mobile Devices. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* 9, 2 (2019), 292–308.
[14] Yu-Hsin Chen, Joel Emer, and Vivienne Sze. 2016. Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks. *ACM SIGARCH Computer Architecture News* 44, 3 (2016), 367–379.
[15] Microprocessor Standards Committee et al. 2019. 754-2019-IEEE Standard for Floating-Point Arithmetic.
[16] Ashish Gondimalla, Noah Chesnut, Mithuna Thottethodi, and T. N. Vijaykumar. 2019. SparTen: A Sparse Tensor Accelerator for Convolutional Neural Networks. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*. 151–165.
[17] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. A. Horowitz, and W. J. Dally. 2016. EIE: Efficient Inference Engine on Compressed Deep Neural Network. In *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*. 243–254.
[18] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep Residual Learning for Image Recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 770–778.
[19] Chao-Tsung Huang, Yu-Chun Ding, Huan-Ching Wang, Chi-Wen Weng, Kai-Ping Lin, Li-Wei Wang, and Li-De Chen. 2019. ECNN: A Block-Based and Highly-Parallel CNN Accelerator for Edge Inference. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*. 182–195.
[20] Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q. Weinberger. 2017. Densely Connected Convolutional Networks. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2261–2269.
[21] S. Hurkat and J. F. Martínez. 2019. VIP: A Versatile Inference Processor. In *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. 345–358.
[22] Patrick Judd, Jorge Albericio, Tayler Hetherington, Tor M. Aamodt, and Andreas Moshovos. 2016. Stripes: Bit-Serial Deep Neural Network Computing. In *IEEE/ACM International Symposium on Microarchitecture*.
[23] Matthias Jung, Christian Weis, and Norbert Wehn. 2015. Dramsys: A flexible dram subsystem design space exploration framework. *IPSJ Transactions on System LSI Design Methodology* 8 (2015), 63–74.
[24] Muhammed Kocabas, Salih Karagoz, and Emre Akbas. 2018. MultiPoseNet: Fast Multi-Person Pose Estimation using Pose Residual Network. In *Proceedings of the European Conference on Computer Vision (ECCV)*. 437–453.
[25] W. Lai, J. Huang, N. Ahuja, and M. Yang. 2017. Deep Laplacian Pyramid Networks for Fast and Accurate Super-Resolution. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 5835–5843. https://doi.org/10.1109/CVPR.2017.618
[26] Alberto Delmas Lascorz, Patrick Judd, Dylan Malone Stuart, Zissis Poulos, Mostafa Mahmoud, Sayeh Sharify, Milos Nikolic, Kevin Siu, and Andreas Moshovos. 2019. Bit-Tactical: A Software/Hardware Approach to Exploiting Value and Bit Sparsity in Neural Networks. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*. 749–763.
[27] J. Lee, C. Kim, S. Kang, D. Shin, S. Kim, and H. Yoo. 2019. UNPU: An Energy-Efficient Deep Neural Network Accelerator With Fully Variable Weight Bit Precision. *IEEE Journal of Solid-State Circuits* 54, 1 (2019), 173–185.
[28] J. Lee, J. Lee, D. Han, J. Lee, G. Park, and H. Yoo. 2019. 7.7 LNPU: A 25.3TFLOPS/W Sparse Deep-Neural-Network Learning Processor with Fine-Grained Mixed Precision of FP8-FP16. In *2019 IEEE International Solid- State Circuits Conference - (ISSCC)*. 142–144.
[29] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. arXiv/1608.08710, 2016. Pruning Filters for Efficient ConvNets. (arXiv/1608.08710, 2016).
[30] Z. Liu, J. Li, Z. Shen, G. Huang, S. Yan, and C. Zhang. 2017. Learning Efficient Convolutional Networks through Network Slimming. In *2017 IEEE International Conference on Computer Vision (ICCV)*. 2755–2763.
[31] Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, and Jan Kautz. 2016. Pruning convolutional neural networks for resource efficient inference. *arXiv preprint arXiv:1611.06440* (2016).
[32] A. Parashar, M. Rhu, A. Mukkara, A. Puglielli, R. Venkatesan, B. Khailany, J. Emer, S. W. Keckler, and W. J. Dally. 2017. SCNN: An accelerator for compressed-sparse convolutional neural networks. In *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*. 27–40.
[33] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. Pytorch: An imperative style, high-performance deep learning library. *arXiv preprint arXiv:1912.01703* (2019).
[34] Joseph Redmon and Ali Farhadi. 2018. YOLOv3: An Incremental Improvement. *arXiv: Computer Vision and Pattern Recognition* (2018).
[35] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L. Chen. 2018. MobileNetV2: Inverted Residuals and Linear Bottlenecks. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 4510–4520.
[36] S. Sharify, A. D. Lascorz, M. Mahmoud, M. Nikolic, K. Siu, D. M. Stuart, Z. Poulos, and A. Moshovos. 2019. Laconic Deep Learning Inference Acceleration. In *2019 ACM/IEEE 46th Annual International Symposium on Computer Architecture (ISCA)*.

304–317.

[37] H. Sharma, J. Park, N. Suda, L. Lai, B. Chau, V. Chandra, and H. Esmaeilzadeh. 2018. Bit Fusion: Bit-Level Dynamically Composable Architecture for Accelerating Deep Neural Network. In *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*. 764–775.

[38] Nathan Silberman, Derek Hoiem, Pushmeet Kohli, and Rob Fergus. 2012. Indoor segmentation and support inference from RGBD images. In *ECCV'12 Proceedings of the 12th European conference on Computer Vision - Volume Part V*. 746–760.

[39] Z. Tian, C. Shen, H. Chen, and T. He. 2019. FCOS: Fully Convolutional One-Stage Object Detection. In *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*. 9626–9635. https://doi.org/10.1109/ICCV.2019.00972

[40] D. Tran, L. Bourdev, R. Fergus, L. Torresani, and M. Paluri. 2015. Learning Spatiotemporal Features with 3D Convolutional Networks. In *2015 IEEE International Conference on Computer Vision (ICCV)*. 4489–4497.

[41] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He. 2017. Aggregated Residual Transformations for Deep Neural Networks. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 5987–5995.

[42] Tianfan Xue, Baian Chen, Jiajun Wu, Donglai Wei, and William T. Freeman. 2019. Video Enhancement with Task-Oriented Flow. *International Journal of Computer Vision* 127, 8 (2019), 1106–1125.

[43] Li Yang, Zhezhi He, and Deliang Fan. 2020. Harmonious coexistence of structured weight pruning and ternarization for deep neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 34. 6623–6630.

[44] X. Ying, L. Wang, Y. Wang, W. Sheng, W. An, and Y. Guo. 2020. Deformable 3D Convolution for Video Super-Resolution. *IEEE Signal Processing Letters* 27 (2020), 1500–1504.

[45] He Zhang and Vishal M. Patel. 2018. Densely Connected Pyramid Dehazing Network. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 3194–3203.

[46] S. Zhang, Z. Du, L. Zhang, H. Lan, S. Liu, L. Li, Q. Guo, T. Chen, and Y. Chen. 2016. Cambricon-X: An accelerator for sparse neural networks. In *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 1–12.

[47] X. Zhou, Z. Du, Q. Guo, S. Liu, C. Liu, C. Wang, X. Zhou, L. Li, T. Chen, and Y. Chen. 2018. Cambricon-S: Addressing Irregularity in Sparse Neural Networks through A Cooperative Software/Hardware Approach. In *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 15–28.

[48] Michael Zhu and Suyog Gupta. 2017. To prune, or not to prune: exploring the efficacy of pruning for model compression. *arXiv preprint arXiv:1710.01878* (2017).

[49] Michael H. Zhu and Suyog Gupta. 2017. To prune, or not to prune: exploring the efficacy of pruning for model compression. In *ICLR (Workshop)*.