FINEA: An Efficient Neural Network Accelerator Exploiting Factorized Input Features

Yujin Kim†, Chanhun Jeong†, Yunho Oh§, Myung Kuk Yoon‡, and Gunjae Koo†
†*Department of Computer Science and Engineering, Korea University, Seoul, South Korea*
§*School of Electrical Engineering, Korea University, Seoul, South Korea*
‡*Department of Computer Science and Engineering, Ewha Womans University, Seoul, South Korea*
E-mail: †{yujin_kim, qiqi26, gunjaekoo}@korea.ac.kr, §yunho_oh@korea.ac.kr, ‡myungkuk.yoon@ewha.ac.kr

***Abstract*—Modern deep neural network (DNN) models increasingly adopt quantized data formats to alleviate the computational burdens of convolution operations. Since quantized models use fewer bits to represent weight parameters and input features, duplicated values often appear within convolution filters. For convolution dot-product operations, we can reduce the number of arithmetic operations if we factorize input feature data with duplicated weight parameters. In this paper, we propose an efficient neural network accelerator architecture, called FINEA, which leverages factorized dot-product operations by exploiting weight redundancy in modern quantized DNN models. By factorizing convolution operations using these duplicated weights, FINEA can significantly reduce the number of required multiplications. To support this factorized approach, FINEA employs a processing engine capable of executing both factorized and unfactorized dot-product operations. FINEA collects input features using filter indexes derived from preprocessed weight tables. We implement a cycle-accurate simulator to evaluate the performance and hardware cost of the proposed architecture. Our evaluation results show that FINEA outperforms conventional systolic array architectures. FINEA also exhibits higher performance for large models compared to the state-of-the-art flexible architecture. Our cost analysis demonstrates that FINEA is a highly energy-efficient architecture for quantized DNN workloads.**

***Index Terms*—DNNs, Accelerator, Quantization**

I. Introduction

Modern neural network applications rely heavily on general matrix multiplication (GeMM) operations since the primary compute kernels in these applications can be mapped to matrix multiplications. Consequently, researchers have extensively investigated efficient neural engine architectures that can accelerate GeMM operations. Dataflow architectures such as systolic arrays are widely adopted in neural network processors because those architectures enable operand transfers between neighboring processing elements (PEs) to reduce data movement costs [1], [2]. However, the conventional systolic array architectures suffer from low utilization issues in PEs since input features and weight parameters include many zero elements that waste PE resources and memory space. As neural network models continue to grow in size and complexity, this performance bottleneck becomes more critical.

Modern neural network models employ quantization techniques to alleviate the heavy computation burdens and storage space of massive convolution operations demanded by large models. Notably, researchers have revealed that deep neural network (DNN) models can achieve reasonable accuracy even when using a fixed-point data format, resulting in significant reductions in computations and memory footprint [3], [4]. As artificial intelligence (AI) applications become pervasive, quantization has emerged as an essential approach for building lightweight and energy-efficient neural network processors. Instead of using floating-point numeric formats that require many bits, we can reduce the size of feature data and/or weight parameters using low-precision formats. Since the quantized formats use a smaller number of bits per data, feature data and weight parameters can exhibit specific characteristics that can be exploited for building more efficient processing engine architectures. However, conventional neural network accelerators based on systolic arrays cannot utilize such characteristics embodied in the quantized data since the systolic array incorporates PEs regularly organized in a two-dimensional array form.

In this paper, we propose an efficient neural network accelerator architecture for quantized DNN models, called FINEA. We observe the quantized DNN models include many duplicated weight parameters in a single convolution filter. Note that computations can be substantially reduced if input features are factorized using duplicated parameters. FINEA includes pairs of *factored* (i.e. multiple features mapped to a single parameter) and *unfactored* (i.e. one feature mapped to a parameter) PEs to perform factorized convolutions efficiently. FINEA allocates input features to factored and unfactored PEs based on filter indexes associated with weight parameters. We implement a cycle-accurate simulator of the proposed accelerator architecture to evaluate FINEA's performance and cost. Our evaluation exhibits FINEA can efficiently improve the performance of quantized DNN models even with fewer computation units.

II. Background

A. Underutilization in Systolic Arrays

A systolic array architecture, commonly used for GeMM operations, leverages data transfers between regularly organized PEs [1], [2], [5]. Systolic array architectures are designed to reuse operands within each PE. For instance, weight-stationary dataflow architecture reuses weight parameters assigned to PEs and streams input features horizontally through the PEs. Hence, systolic array architectures can reduce data movement

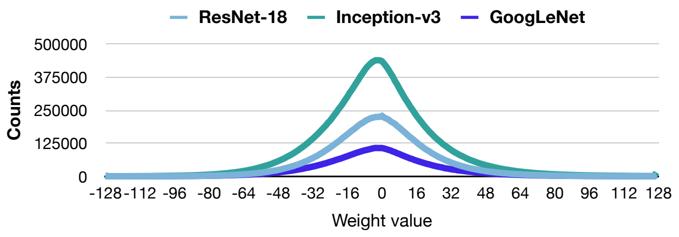


Fig. 1: Weight value counts in quantized DNN models

costs by allowing data transfers between neighbor PEs. However, despite such benefits, systolic array architectures suffer from underutilization issues [6]. The native timing control scheme of the systolic array architecture leaves some PEs idle during a GeMM operation. Additionally, PEs often perform unnecessary multiplications when one of the operands is zero. Researchers revealed zero values are frequently observed in input features and weight parameters [7], [8]. For a 128×128 systolic array, PE utilization is 21.7% on average for popular DNN models and lower than 17.2% for MobileNet and ShuffleNet (see Figure 10 in Section V).

B. Quantization

Quantization is one of the algorithmic approaches that can reduce the size of DNN models. Prior studies present DNN models can achieve reasonable accuracy levels even with reduced data precision [9]–[12]. Several researchers presented that DNN models can be implemented using an 8-bit integer data format without sacrificing the accuracy of the models [2], [3], [13]. By exploiting the quantization techniques, we can reduce the number of bits allocated to a single numeric data to decrease the storage space required for parameters and features [14], [15]. Moreover, each PE includes simpler hardware logic that handles reduced bits, thus hardware costs can be decreased [16]. Consequently, quantized models exhibit significant advantages in performance (i.e. lower latency) and cost (i.e. reduced energy consumption and chip area).

III. Motivation

A. Weight Duplication

In this section, we motivate the key idea of this paper by examining the characteristics of low-precision weight parameters in quantized DNN models. As mentioned in the previous section, the weight parameters in widely used DNN models can be represented using an 8-bit low-precision format. In such cases, only 256 distinct values can be represented. Moreover, a narrow range of quantized weight parameters is deployed more frequently since the weight parameter values in a DNN inference model exhibit a normal distribution [17]. We analyze the weight distributions of three DNN models quantized using an 8-bit integer format, as illustrated in Figure 1. Our analysis reveals that the majority of weight values fall within the range of -64 to 64 even though the full 8-bit format can support a much wider numeric range. This observation implies a high degree of redundancy in convolution filters as many weight parameters can be duplicated within a convolution filter set.

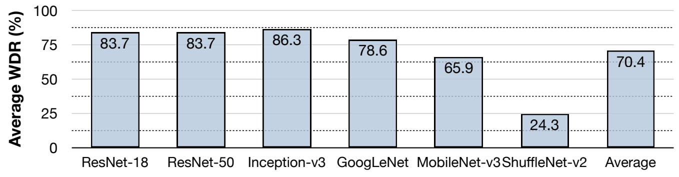


Fig. 2: Weight duplication rate in quantized DNN models

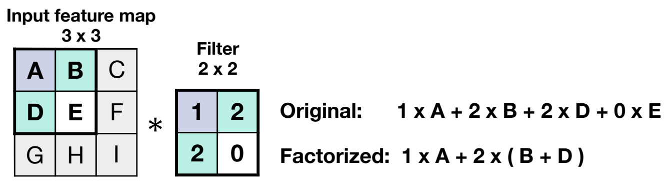

Input feature map
3 x 3
A
B
C
D
E
F
G
H
I
*
Filter
2 x 2
1
2
2
0
Original: 1 x A + 2 x B + 2 x D + 0 x E
Factorized: 1 x A + 2 x (B + D)

Fig. 3: Factorized dot-product in a convolution filter

As shown in Figure 2, we investigate the fraction of duplicated parameters within a convolution filter for the quantized DNN models listed in Table II. The studied DNN models employ 8-bit quantized weight parameters computed with either post-training quantization or quantization-aware model training. To quantify the fraction of duplicated weights in the DNN models, we compute the weight duplication rate (WDR) using Equation (1). In the equation, n and W are the number of parameters and a set of parameter values in a convolution filter, respectively. For each weight value (i.e. w), $c(w)$ is a count of the weight value, thus $(c(w) - 1)$ denotes a count of the duplicated weight value. Namely, the count is accumulated only if the weight value count is larger than one in the filter.

$$\mathrm{WDR} = \frac{\sum_{w \in W}(c(w) - 1)}{n} \tag{1}$$

We compute the average WDRs across all convolution filters in the quantized DNN models. Our analysis reveals that on average 70% of weight values are duplicated within a single filter. For the first five DNN models, the average WDR reaches approximately 80%. These results support the intuition that a significant number of weights are repeated within individual filters in quantized DNN models. This means we can streamline convolution operations by factorizing input features according to duplicated weight values. Note that a convolution operation is equivalent to a dot product between input features and weight parameters. Hence, we can reduce the number of multiplications by aggregating input features that correspond to the same weight values.

B. Factorization in a Convolution Filter

In this section, we describe how a factorization approach can reduce multiplication counts in a convolution filter. We explain the factorization approach using a simple convolution example as shown in Figure 3. We assume the input feature map size is 3×3 and the parameter filter size is 2×2. The convolution operations are performed by sliding the filter matrix on the input feature map matrix. First the upper-left 2×2 input features are convoluted with the filter matrix. For

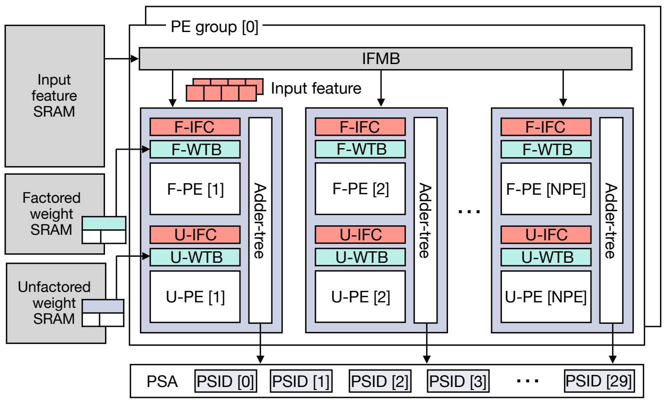


Fig. 4: Overall FINEA hardware architecture

the normal *unfactored* dot-product operation, input feature elements are multiplied with the corresponding filter elements, thus four multiplications are required. On the other hand, the dot-product equation can be factorized with the duplicated weight parameter (i.e. *2* in the example). In addition, the partial product multiplied by zero (i.e. $0 \times E$) is removed. Hence, the factorized convolution includes only two multiplications. As mentioned in Section III-A, the quantized weight parameters in an 8-bit data format exhibit an extremely high weight duplication rate in a convolution filter, thus we can dramatically reduce the number of multiplications if input features are ideally factorized with the duplicated weight parameters.

We evaluate how many multiplications can be removed if the factorized dot-product is fully deployed for convolution filters in the quantized DNN models. To quantify this, we normalize the number of multiplications under the ideal factorization approach to the original unfactored dot-product computations typically executed on systolic array architectures. As shown in Figure 11 in Section V, the multiplication counts decrease dramatically when input features are fully factorized. For quantized ResNet-18 and Inception-v3, we can reduce approximately 56% of multiplications if the factorized dot-product is fully employed in convolutions. The reduction rate of multiplications is proportional to the weight duplication rate (see Figure 2). To summarize, we can dramatically reduce heavy multiply operations demanded by convolutions using input feature factorization with duplicated weight parameters, thus the factorization approach can be an effective solution that can tackle the hardware underutilization issues observed in the existing neural network accelerators.

IV. Proposed Architecture

Given that quantized DNN models exhibit high weight duplication rates, factorization using duplicated weights presents an effective strategy for reducing the computation burdens of convolution operations. In order to implement an efficient hardware architecture that can exploit duplicated weights in quantized DNN models, we propose a *F̲actorized I̲nput-based N̲eural E̲ngine A̲rchitecture*, called *FINEA*. FINEA facilitates the factorization of input features using weight parameters by assigning input features to separate PE lanes for factored dot-product (by duplicated weights) and unfactored dot-product (by non-duplicated weights). By leveraging factorized input-based PE structures, FINEA can utilize PE resources efficiently to improve the performance of quantized DNN models.

Figure 4 depicts the overall architecture of FINEA. Like the conventional DNN accelerators, FINEA incorporates internal SRAM buffers for input features (i.e. input feature SRAM) and weight parameters (i.e. factored weight SRAM and unfactored weight SRAM). To avoid data access conflicts, the internal SRAM buffers employ a double-buffering structure, thus one partition fetches data from external device memory and another partition provides demanded data to PE groups. Each PE group includes an input feature map buffer (IFMB) and multiple factorization-based processing engines (PEs). Note that N_{PE} in the figure represents the number of PEs per PE group. Each PE collects input feature data from IFMB using the index information in a filter. Note that FINEA employs a weight-stationary architecture thus each PE holds the indexes of weight parameters for long cycles once weight data are assigned to PEs. Since a filter matrix slides on an input feature matrix for convolutions, IFMB employs a first-in-first-out (FIFO) structure like the input feature map sliding buffer of the weight-stationary dataflows. To implement the factorized dot-product described in Section III-B, each PE includes separated multiplier lanes for factored (F-PE) and unfactored (U-PE) operations. For the factored dot-product, input features are grouped by a duplicated weight, thus each multiplier lane of F-PE includes multiple slots for the input feature set. The input features collected in the slots of an F-PE multiplier lane are first accumulated, and then the accumulated input features are multiplied by a weight parameter. For the unfactored dot-product, only one feature is associated with a single weight parameter. The multiple products computed by the multiplier lanes of F-PE and U-PE are merged to generate a partial sum. The partial sum associated with each PE is encapsulated with a partial sum id (PSID) assigned to the PE, then the partial sums are delivered to the partial sum accumulator (PSA). FINEA generates output features by accumulating the partial sums bucketized by PSID in PSA.

A. Weight Tables

FINEA factorizes input features using weight parameters, thus input features in IFMBs are allocated to PE's slots associated with weights. For this purpose, FINEA employs weight tables that include mapping information between input features and weights. Figure 5 showcases the factored and unfactored weight tables based on an example filter matrix. The filter matrix includes duplicated elements (i.e. four *2*s and two *3*s), a non-duplicated element (i.e. *1*), and zeros. In the factored weight table, each entry includes a weight value and the indexes in the filter matrix. A single entry of the factored weight table includes four indexes thus a single weight parameter can be associated with up to four input features. An entry of the unfactored weight table contains a pair of a weight value and an index of the weight, but only

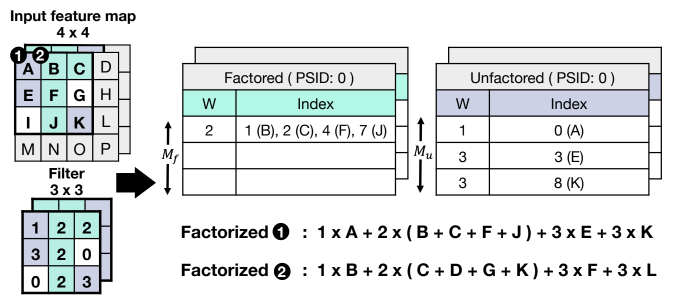


Fig. 5: Factored and unfactored weight tables

one index is mapped to a single weight value. Additionally, PSID identifies the partial sums generated by PEs, ensuring the partial sums are correctly merged in PSA.

In this example, the weight parameter `2` appears four times in the filter matrix, thus the weight `2` is registered in a factored weight table entry, and four indexes (i.e. `1`, `2`, `4`, and `7`) are assigned in the index field. Note that the letters (i.e. `B`, `C`, `F`, and `J`) just indicate the input features associated with the indexes. The indexes in the index field are used for fetching input features from an IFMB. Note that the weight parameter `3` is registered in the unfactored weight table even though the weight value appears twice in the filter matrix. That is because the number of the weight `3` is lower than the factorization threshold (Th_{fact}). Th_{fact} is determined by the ratio of multiplier counts between factored and unfactored PEs. If a weight count is over the threshold, weight parameters duplicated more are prioritized and assigned to the factored weight table first. Note that the weight *zero* is not registered in any weight tables thus FINEA can remove unnecessary multiplications. We configure that a single index in the weight tables includes 8-bit index data, thus FINEA can support a large filter matrix that includes up to 256 elements.

FINEA employs weight-stationary dataflows, thus once the weight tables are loaded in a PE, the same weight parameters can be reused for generating output features associated with input features and the same filter matrix. As shown in Figure 5, the filter matrix slides on the larger input feature matrix to generate the elements of output features. In order to support the sliding of a filter matrix, FINEA adjusts the location of input features in IFMBs. Namely, the input features propagate to the next registers in an IFMB when the filter matrix slides from the first position (❶) to the next position (❷) in the example. Hence, FINEA can deploy the same weight tables until all elements of the output feature matrix are computed.

Although FINEA deploys the weights in specific formats for building weight tables, the preprocessing overhead is negligible. Note that DNN inferences use fixed parameters computed during training. Hence, FINEA can repeatedly deploy the same preprocessed weight tables for multiple inferences once the parameters are formatted for FINEA's weight table structures.

B. Factorization-Based Processing Engine

Figure 6 illustrates the detailed architecture of a single PE of FINEA. In order to support the factorized dot-products, a

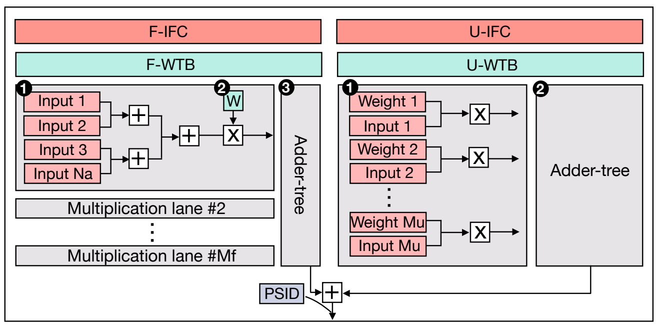
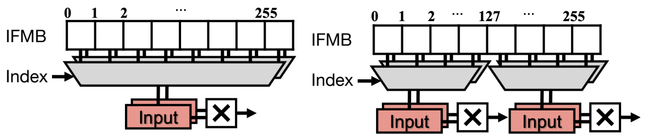


Fig. 6: F-PE and U-PE architectures in a single PE

single PE includes a factored PE (F-PE) and an unfactored PE (U-PE). A PE also accommodates the two weight table buffers, F-WTB and U-WTB for factored and unfactored weight tables respectively, to hold weight parameters. The input feature collectors (IFCs) fetch input features from IFMB in the corresponding PE group to fill the input feature slots in the multiply lanes in F-PE and U-PE. F-PE includes multiple multiply lanes and the adder tree that accumulates the products of the multiply lanes. Each multiply lane includes multiple input feature slots to support the factored dot-product. In the figure, N_a represents the number of input feature slots per multiply lane and M_f is the number of the multiply lanes in F-PE. Similarly, U-PE also has multiple multiply lanes, and each lane includes a single input feature slot. M_u is the number of multiply lanes in U-PE. The outputs of F-PE and U-PE are added to generate the partial sum result of the PE.



(a) FA IFC (b) PA IFC

Fig. 7: IFC associativity

Input feature collectors (IFCs): F-PE and U-PE include IFCs (i.e. F-IFC and U-IFC respectively) that collect input features from IFMB in the PE group. IFCs read the required input features using the indexes in the index field of individual weight table entries. To achieve high data access throughput from the IFMB, IFCs fetch multiple demanded elements concurrently. Namely, once the indexes from the weight tables are allocated to individual input feature slots in IFC, the demand data is loaded in parallel to each slot associated with the corresponding multiply lane. To support the required operations, IFC includes multiplexers that select input features from all IFMB entries using the allocated indexes (see Figure 7a). We call this type of IFC a fully-associative (FA) IFC.

Although FINEA's IFC design guarantees high throughput for loading input features to multiply lanes, our evaluation reveals the power consumption burden by IFCs is rather heavy since an individual IFC includes large-sized multiplexers. We synthesize the IFC modules using the ASAP7 standard-cell library [18] to estimate the power consumption by IFCs [19]. Our evaluation exhibits that, with 256 IFMB entries per PE group, the FA IFC accounts for 30.8% of FINEA's total power consumption (see Table III in Section V).

To reduce the power consumption burden of the original FA IFC design, FINEA restricts the number of IFMB entries associated with each IFC slot as shown in Figure 7b. This type of IFC is called a partially-associative (PA) IFC. PA IFCs can marshal smaller multiplexers compared to FA IFCs thus we can reduce the power consumption and wiring overhead of IFCs. If PA IFCs are employed in FINEA, the multiply lanes in F-PE and U-PE have fixed index bases. For instance, if the multiply lane $[n]$, where n is a lane number, is associated with the IMFB entries from 0 to 31, the index base of the multiply lane $[n]$ is 00000. Then, the index field of the corresponding weight table entry can include the remaining index bits (e.g. 3 bits if IFMB size is 256). As such, the weight tables need to be organized considering the PA IFC design approach since each entry of a weight table is associated with the slots of the corresponding multiply lane. If PA IFCs are employed, the performance of FINEA can be downgraded since the utilization of multiply lanes may go down due to slot conflict issues. We investigate such aspects intensively in Section V.

Factored dot-product: FINEA combines multiple input features with a single weight to perform factored dot-product in F-PE. FINEA allocates the valid indexes to the input feature slots of each multiply lane by reading the index field in the corresponding entry of the factored weight table. Then, F-IFC fetches input features concurrently from IFMB using the allocated indexes. Once the fetched data occupy the input feature slots of the multiply lane, the input features are summed by the slot adder-tree. Then, the accumulated feature value is multiplied by the weight parameter assigned to the multiply lane. F-PE sums the partial products from M_f multiply lanes to generate the result of the factored dot-product.

Unfactored dot-product: Like F-PE, multiply lanes of U-PE are also associated with the corresponding entries of the unfactored weight table in U-WTB. Namely, the single input feature is fetched by U-IFC using the index in the mapped weight table entry, then the fetched input feature is multiplied by the weight parameter assigned to the multiply lane. U-PE adds the partial products computed by M_u multiply lanes to make the unfactored dot-product result.

Finally a PE adds the results from F-PE and U-PE to generate a partial sum labeled with PSID. If a filter size is larger than the IFMB entry counts, multiple PEs are required to compute a single output feature. In this case, multiple partial sums with the same PSID are accumulated in PSA

C. Processing Flows

In FINEA, all operations pass through the common datapath elements except for the factored and unfactored dot-product operations within a PE. As shown in Figure 6, F-PE performs a factored dot-product operation composed of ① sum of input features, ② product of a weight and the summed inputs, and ③ accumulation from multiply lanes. The computation of U-PE consists of ① product of a weight parameter and an input feature, and ② sum of the products. Note that F-PE includes the additional *sum of input features* operation compared to U-PE. To balance the processing latency between F-PE and U-PE, we adjust the numbers of multiply lanes in F-PE and U-PE (i.e. M_f and M_u respectively). We estimate the processing latency of F-PE and U-PE by synthesizing with the ASAP7 standard cell library [18]. We choose (M_f, M_u)=(8, 32) for the baseline configuration since the latency of F-PE and U-PE aligns within 1 GHz of clock frequency. Since F-PE and U-PE processing flows are aligned, all processing steps of FINEA (i.e. input feature collection, F-PE and U-PE operations, and partial sum accumulation) can be pipelined.

V. Evaluation

We implement a cycle-accurate simulator to evaluate FINEA. We also implement the hardware modules of FINEA in Verilog HDL to estimate power and energy consumption. We synthesize the designed modules targeting 1 GHz of clock frequency using the ASAP standard cell library [18]. To compare FINEA with the prior accelerator architectures, we study three TPU-style systolic array architectures configured with 128×128 (with 16,384 PEs, denoted as *SA128*), 64×64 (4,096 PEs, *SA64*), and 32×32 (1,024 PEs, *SA32*). We also compare FINEA with MAERI which implements flexible architectures with reconfigurable interconnects [20]. Table I summarizes the default configurations of FINEA. By adjusting the number of PE groups (i.e. N_{PE}), we configure FINEA in three types (*small*, *medium*, and *large*), which include similar multiplier counts as SA32, SA64, and SA128 respectively. Table II exhibits the characteristics of the quantized DNN models evaluated in this work. All the DNN workloads employ an 8-bit fixed-point data format.

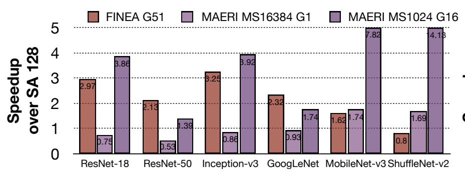
TABLE I: Default FINEA configuration

	Small	**Medium**	**Large**
Factored input slots per lane (N_a)	4	4	4
No. of multiplier lanes in F-PE (M_f)	8	8	8
No. of multiplier lanes in U-PE (M_u)	32	32	32
No. of PEs per PE group (N_{PE})	8	8	8
No. of PE groups (G)[†]	3	12	51
Input feature SRAM (KB)	5.5	5.5	5.5
Factored weight SRAM (KB)	27.5	27.5	27.5
Unfactored weight SRAM (KB)	11	11	11

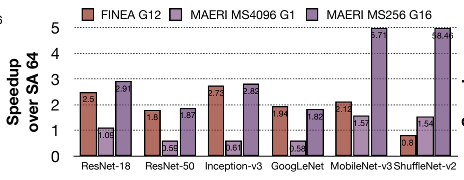
[†] G is chosen so that G×$(M_f + M_u)$×N_{PE} matches the total multiplier count of each systolic array.

TABLE II: DNN models

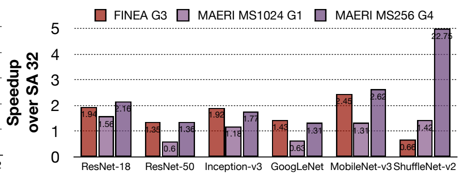
DNN model	**Input size**	**Conv.**	**Weights**	**MACs**
ResNet-18 [21]	224×224	20	11.69 M	1.81 G
ResNet-50 [21]	224×224	53	25.56 M	4.0 G
Inception-v3 [22]	224×224	94	27.16 M	5.71 G
GoogLeNet [23]	224×224	57	6.62 M	1.5 G
MobileNet-v3 [24]	224×224	62	5.48 M	0.22 G
ShuffleNet-v2 [25]	224×224	56	1.37 M	0.04 G



(a) Large (FINEA G51)



(b) Medium (FINEA G12)



(c) Small (FINEA G3)

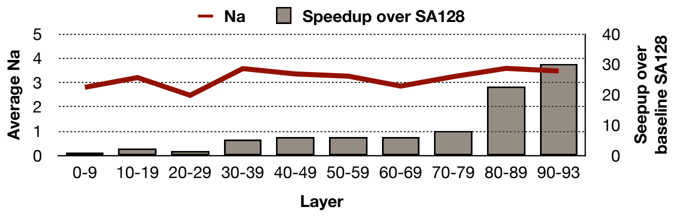
Fig. 8: Speedup of FINEA over systolic arrays

A. Performance

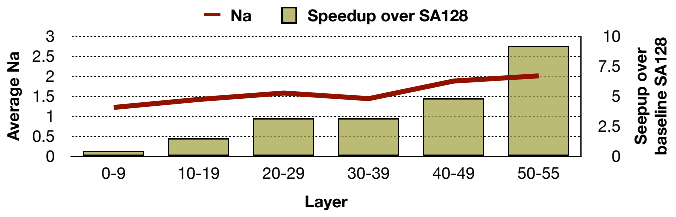
Figure 8 exhibits the performance of FINEA configured with *large*, *medium*, and *small* settings. The number next to G represents the number of PE groups in FINEA. We also compare the performance of FINEA with MAERI. The performance of FINEA and MAERI is normalized to systolic array architectures (i.e. SA128, SA64, and SA32 respectively). For the *large*, *medium*, and *small* configurations, FINEA achieves 2.18×, 1.98×, and 1.62× speedups over systolic arrays respectively. FINEA outperforms baseline systolic arrays since FINEA employs factorized dot-product operations to utilize processing units more efficiently. FINEA exhibits dramatic performance improvement for large DNN models (the first four models in Table II). On the other hand, we can observe minimal performance changes for the small models such as ShuffleNet-v2 since most of the convolution layers use 1×1 kernels and shallow channels, so they contain fewer weight elements than the 256 entries per IFMB, leaving insufficient weight redundancy to achieve substantial acceleration.

To compare the performance of FINEA with MAERI, we configure MAERI in two different settings. MAERI employs reconfigurable tree-structured multipliers and hierarchical router structures to support various DNN layers. We match the total number of multipliers in MAERI to that of the baseline systolic arrays. In this context, the number following *MS* indicates the number of leaf node multipliers in MAERI, while the number following *G* denotes the number of parallel multiplier-switch tree instances. For example, MAERI MS1024 G16 contains a total of 16,384 multipliers, equivalent to the PE count in SA128. MAERI with more parallel trees (higher *G*) delivers better performance on small DNN models (e.g., MobileNet-v3 and ShuffleNet-v2) since it is highly optimized for 1×1 point-wise convolutions. In contrast, FINEA outperforms MAERI with a single large reduction tree (i.e. *G1*) when running large DNN models.

We further analyze the performance of FINEA by layers of the DNN models as shown in Figure 9. Figure 9a exhibits the speedup of FINEA over the baseline SA128 for every 10 layers of Inception-v3, showing how speedup relates to the Na. Na indicates the number of inputs shared by each weight, reflecting the degree of redundancy. Note that FINEA exhibits high performance for Inception-v3 compared to the SA128. In our evaluation, layers 30–93, which feature many filters and deep channel dimensions, achieved the highest speedup by maximizing weight redundancy. By factorizing dot-product operations and reducing the total number of processing counts, FINEA effectively boosts the performance of convolution layers. On the other hand, FINEA exhibits marginal improvements in the first 30 layers, due to their shallow channel depths and small filter counts reflected by the low Na values in Figure 9a. Compared to layers 10-19, layers 60-69 have less weight duplication but a much smaller input feature, resulting in less IFMB reload and activation reuse. Figure 9b shows the performance of FINEA on ShuffleNet-v2 layers as the channel width scales from 24 to 116, 232, 464, and 1,024. In the early stages (0–9 layers), only 24 channels combined with a large feature map size results in low input reuse redundancy and low acceleration. However, as the network grows deeper, the channel depth increases and the activation dimension shrinks to eliminate redundant computation of weights while reusing activations. In fact, across widely deployed quantized CNNs such as ResNet, Inception, and GoogLeNet, the average duplication rate reaches 70.4%, where FINEA demonstrates its strongest benefits. Importantly, even in low redundancy cases, FINEA prevents severe degradation by employing a threshold-based allocation (Th_{fact}) that balances factored and unfactored paths, ensuring robustness while scaling performance with redundancy.



(a) Average speedup of FINEA by every 10 layers (Inception-v3)

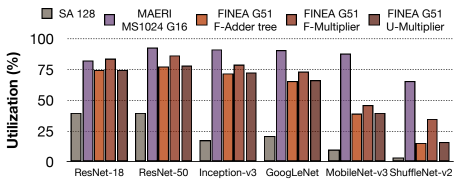


(b) Average speedup of FINEA by every 10 layers (Shufflenet-v2)

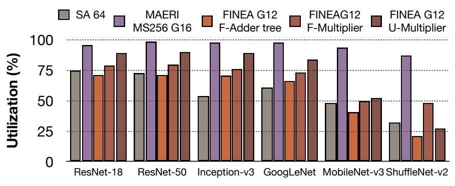
Fig. 9: Performance of FINEA by DNN layers

B. Utilization

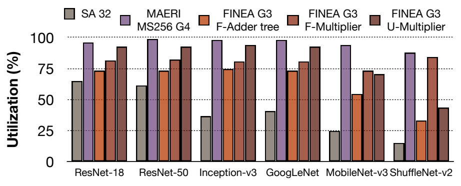
We analyze the utilization of compute units in FINEA, MAERI, and the systolic arrays (i.e. SA128, SA64, and SA32) as shown in Figure 10. For the systolic arrays, PE resources



(a) Large (FINEA G51)



(b) Medium (FINEA G12)



(c) Small (FINEA G3)

Fig. 10: Utilization of FINEA, systolic arrays, and MAERI

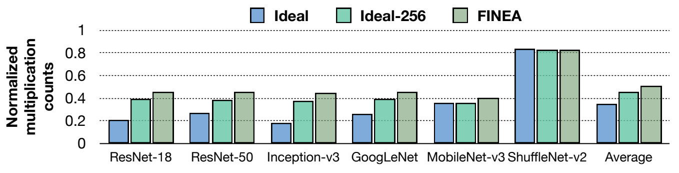
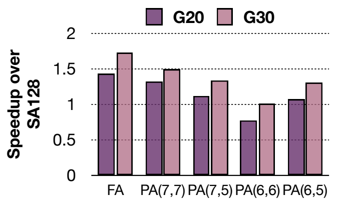
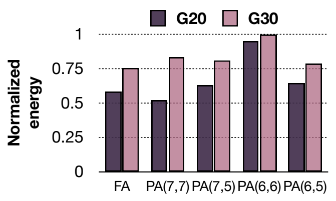


Fig. 11: Multiplication counts by factorization of FINEA



(a) Performance by IFC assoc.



(b) Energy by IFC assoc.

Fig. 12: Performance/energy of FINEA by IFC associativity

are wasted if no input feature is assigned to the PE or a zero is allocated for the weight parameter. As shown in the figure, the systolic array exhibits significantly low processing unit utilization, with an average of 39.7%. For small DNN models, such low utilization of PEs is the main performance bottleneck of the systolic array as mentioned in Section II.

Considering FINEA's unique hardware architecture, we break down the utilization of FINEA based on F-PEs and U-PEs. As described in Section IV-B, each multiply lane of F-PE includes multiple input feature slots for factored dot-products. *F-adder tree* represents the utilization of these input feature slots, and we assume the input slots are wasted if no input features are allocated. Our analysis exhibits that the utilization of *F-Adder tree* is very high (71.9% on average) for the four large DNN models. We also measure the utilization of the multiply lanes of F-PE (i.e. *F-Multiplier*) and U-PE (i.e. *U-Multiplier*). By efficiently distributing both factored and unfactored dot-products to F-PEs and U-PEs, FINEA achieves an average multiplier utilization of 80% across four large DNN smodels. On small models like MobileNet-v3 and ShuffleNet-v2, their tiny input feature maps and filters can't keep all provisioned PE groups busy, resulting in low utilization. However, scaling the systolic array down from large to medium and small (Figure 10) improves PE utilization across all architectures. MAERI similarly leverages a flexible reduction-tree to distribute weights and activations and dynamically accumulate partial sums in flight, achieving high utilization. However, residual redundant operations still limit its speedup compared to FINEA's factorization approach.

C. Computation Reduction

In Figure 11 we compare the multiplication counts by factorized dot-products for three cases: an ideal case with infinite IFMB size, an ideal case with limited IFMB size (256 entries per IFMB), and FINEA with limited input feature slots (i.e. N_a = 4). In the figure, the number of multiplications for each case is normalized with the original (i.e. fully-unfactored) dot-product operations. Note that this fully-unfactored baseline is exactly the same multiplication count incurred by both MAERI and the systolic arrays. For the ideal factorized dot-product scenario, we assume all duplicated weight parameters can be factored perfectly. If larger IFMBs are employed, FINEA can find more duplicated parameter weights since FINEA can exploit the weight tables extracted from larger convolution filters that may include more duplicated weight parameters. Our analysis results reveal that more than 60% of total multiplications can be removed if convolutions are perfectly factorized. FINEA can efficiently exploit the factorized dot-product operations even with limited hardware resources in IFMBs and F-PE, hence FINEA can reduce 50.5% of multiplications on average.

D. Power and Energy Consumption

We break down the power consumption of the hardware components of FINEA for a single PE group as listed in Table III. As described in Section IV, a single PE group includes an IFMB and multiple PEs. We use the SA 128 configuration listed in Table I. Our breakdown analysis exhibits IFCs consume 30.8% of the total power of a single PE group since IFCs employ large multiplexers to fetch input features using 8-bit weight indexes. In order to reduce the power consumption overhead by IFCs, FINEA can employ the partially-associative IFCs as described in Section IV-B.

TABLE III: Power consumption breakdown in a PE group

Component	**F-IFC**	**U-IFC**	**F-PE**	**U-PE**
Power (mW)	7.24	7.24	4.77	10.48
Component	**IFMB**	**F-WTB**	**U-WTB**	**Total**
Power (mW)	5.10	6.47	5.69	46.99

Figure 12 exhibits FINEA's average performance and energy consumption by IFC associativity. To isolate the effect of IFC associativity under equal baseline throughput, we fix FINEA to

G20 an iso-performance setting that yields consistent speedup across all designs. The power and energy consumption of FINEA is normalized to the SA 128. FA represents the fully-associative IFC design that uses 8-bit indexes for both F-PE and U-PE. PA represents the partial-associative IFC design and the following numbers denote the width of an index employed by F-IFC and U-IFC respectively. For instance, if the number is 6, IFC uses 6-bit indexes thus 2-bit is used as the index base (see the descriptions in Section IV-B). The power consumption by an IFC can be reduced as the width of indexes in the IFC is decreased. However, FINEA's performance can be also downgraded with the reduced IFC design since IFCs cannot allocate input feature data due to conflicts in PE slots.

Our evaluation results exhibit FINEA's performance is downgraded when the partially-associative IFCs are employed. However, the energy consumption can be optimized with a more power-efficient IFC design. FINEA with 20 PE groups (i.e. G20) exhibit higher performance when FA IFCs are employed, but PA(7, 7) IFCs exhibit better energy efficiency even though the performance is decreased by 12%. Note that FINEA G20 uses only 39% of the multipliers of SA128 while consuming just 52% of its energy. FINEA further increases throughput with G30 as shown in Figure 12a. Although this larger configuration reduces energy efficiency, it still sustains the same energy level as SA128.

VI. Related Work

Researchers have explored the enhanced neural network architectures that can tackle performance burdens by PE underutilization and redundant computations. Cnvlutin detects zero values in activations at runtime to skip the associated MAC operations [26]. CluSpa proposes a software-based approximation for convolution inference that clusters similar weight values using Gaussian mixture models [27]. By leveraging both weight and activation sparsity, CluSpa effectively eliminates redundant multiplications. CluSpa groups weights into a configurable number of clusters per filter and performs multiplication using only the centroid of each cluster, significantly reducing both MAC operations and memory bandwidth requirements.

Several researchers presented structural factorization and pattern reuse approaches to reduce computation burdens. An adaptive pooling–based convolution factorization technique transforms MAC-intensive convolutions into simpler pooling aggregations on a per-activation basis [28]. This transformation is guided by a lightweight CNN predictor (RedZAP), which determine under a given accuracy budget which activations can be factorized. This dynamic approach achieves an additional reduction in MAC operations compared to zero-activation prediction methods. ReRAM-based PIM accelerators have begun to exploit nonzero weight pattern repetitions (WPRs) [29]. PattPIM conducts a thorough analysis of WPRs across several quantized DNN models and introduces a WPR-to-operation unit (OU) mapping scheme, where an OU represents a small crossbar partition. OUs classified as WPSP (weight-pattern-same-position) compute each common weight pattern only once and broadcast the result to all OUs sharing that pattern, thereby maximizing both spatial and computational efficiency.

Prior research exploit redundant weight values to reduce computation burdens. UCNN employs a factorization approach [16]. UCNN identifies a set of unique weights within a layer and groups the associated activations to compute partial sums. These partial sums are then reused across filters that share the same weights. UCNN also employs an input-stationary dataflow to maximize the reuse of partial sums. However, UCNN employs a specific quantized format, thus UCNN cannot be applied to general quantized formats. PatterNet trains neurons to adopt a small set of fixed weight templates [30]. Once training is complete, the accelerator computes each template only once and broadcasts the result to all filters, assuming identical weight patterns across filters. On the other hand, FINEA's approach can be applied to general quantized formats that exhibit high weight redundancy rates. Once weight parameters are formatted into FINEA's weight tables, FINEA can perform factored and unfactored dot-products to achieve efficient inferences.

VII. Conclusion

In this paper, we propose FINEA, a high-performance and energy-efficient neural engine architecture that exploits factorized convolution operations. We reveal that modern quantized DNN models exhibit high weight duplication rates, namely many weight parameters are duplicated in a single convolution filter. Hence, the number of required multiplications can be significantly reduced by factorizing convolution operations using the duplicated weights. In order to exploit such optimizations, FINEA incorporates a processing element architecture capable of executing both factorized and unfactorized dot-product operations. FINEA collects input features based on the indexes of filter parameters from preprocessed weight tables. We also present optimization strategies considering the power and energy consumption of FINEA. By employing an architecture tailored for factorized convolutions, FINEA achieves a 2.18× performance improvement over systolic arrays when using the same number of processing units. Furthermore, FINEA demonstrates lower energy consumption compared to traditional systolic array designs.

Acknowledgment

This work was supported in part by the National Research Foundation of Korea (NRF) funded by Korea government (MSIT) (NRF-2021R1C1C1012172 and RS-2025-02214322), and in part by the Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (IITP-2025-RS-2020-II201819, ICT Creative Consilience Program / RS-2025-02304483, Chiplet-Based Hub SoC Development Optimized for On-Device Al). The EDA tool was supported by the IC Design Education Center (IDEC), Korea. Gunjae Koo is the corresponding author.

References

[1] H.-T. Kung, "Why systolic architectures?" *Computer*, vol. 15, no. 1, pp. 37–46, 1982.

[2] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers *et al.*, "In-datacenter performance analysis of a tensor processing unit," in *Proceedings of the 44th annual international symposium on computer architecture*, 2017, pp. 1–12.

[3] R. Krishnamoorthi, "Quantizing deep convolutional networks for efficient inference: A whitepaper," *arXiv preprint arXiv:1806.08342*, 2018.

[4] Y. Yang, L. Deng, S. Wu, T. Yan, Y. Xie, and G. Li, "Training high-performance and large-scale deep neural networks with full 8-bit integers," *Neural Networks*, vol. 125, pp. 70–82, 2020.

[5] F. Sijstermans, "The nvidia deep learning accelerator," in *Hot Chips*, vol. 30, 2018, pp. 19–21.

[6] E. Qin, A. Samajdar, H. Kwon, V. Nadella, S. Srinivasan, D. Das, B. Kaul, and T. Krishna, "Sigma: A sparse and irregular gemm accelerator with flexible interconnects for dnn training," in *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2020, pp. 58–70.

[7] A. Mishra, J. A. Latorre, J. Pool, D. Stosic, D. Stosic, G. Venkatesh, C. Yu, and P. Micikevicius, "Accelerating sparse deep neural networks," *arXiv preprint arXiv:2104.08378*, 2021.

[8] T. Gale, E. Elsen, and S. Hooker, "The state of sparsity in deep neural networks," *arXiv preprint arXiv:1902.09574*, 2019.

[9] S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan, "Deep learning with limited numerical precision," in *International conference on machine learning*. PMLR, 2015, pp. 1737–1746.

[10] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 2704–2713.

[11] M. Nagel, M. Fournarakis, R. A. Amjad, Y. Bondarenko, M. Van Baalen, and T. Blankevoort, "A white paper on neural network quantization," *arXiv preprint arXiv:2106.08295*, 2021.

[12] R. Banner, Y. Nahshan, and D. Soudry, "Post training 4-bit quantization of convolutional networks for rapid-deployment," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[13] F. Zhu, R. Gong, F. Yu, X. Liu, Y. Wang, Z. Li, X. Yang, and J. Yan, "Towards unified int8 training for convolutional neural network," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 1969–1979.

[14] A. Gholami, S. Kim, Z. Dong, Z. Yao, M. W. Mahoney, and K. Keutzer, "A survey of quantization methods for efficient neural network inference," in *Low-Power Computer Vision*. Chapman and Hall/CRC, 2022, pp. 291–326.

[15] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. A. Horowitz, and W. J. Dally, "Eie: Efficient inference engine on compressed deep neural network," *ACM SIGARCH Computer Architecture News*, vol. 44, no. 3, pp. 243–254, 2016.

[16] K. Hegde, J. Yu, R. Agrawal, M. Yan, M. Pellauer, and C. Fletcher, "Ucnn: Exploiting computational reuse in deep neural networks via weight repetition," in *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2018, pp. 674–687.

[17] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *arXiv preprint arXiv:1510.00149*, 2015.

[18] L. T. Clark, V. Vashishtha, L. Shifren, A. Gujja, S. Sinha, B. Cline, C. Ramamurthy, and G. Yeric, "Asap7: A 7-nm finfet predictive process design kit," *Microelectronics Journal*, vol. 53, pp. 105–115, 2016.

[19] J. E. Stine, I. Castellanos, M. Wood, J. Henson, F. Love, W. R. Davis, P. D. Franzon, M. Bucher, S. Basavarajaiah, J. Oh *et al.*, "Freepdk: An open-source variation-aware design kit," in *2007 IEEE international conference on Microelectronic Systems Education (MSE'07)*. IEEE, 2007, pp. 173–174.

[20] H. Kwon, A. Samajdar, and T. Krishna, "Maeri: Enabling flexible dataflow mapping over dnn accelerators via reconfigurable interconnects," *ACM Sigplan Notices*, vol. 53, no. 2, pp. 461–475, 2018.

[21] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[22] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 2818–2826.

[23] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1–9.

[24] A. Howard, M. Sandler, G. Chu, L.-C. Chen, B. Chen, M. Tan, W. Wang, Y. Zhu, R. Pang, V. Vasudevan *et al.*, "Searching for mobilenetv3," in *Proceedings of the IEEE/CVF international conference on computer vision*, 2019, pp. 1314–1324.

[25] N. Ma, X. Zhang, H.-T. Zheng, and J. Sun, "Shufflenet v2: Practical guidelines for efficient cnn architecture design," in *Proceedings of the European conference on computer vision (ECCV)*, 2018, pp. 116–131.

[26] P. Judd, A. Delmas, S. Sharify, and A. Moshovos, "Cnvlutin2: Ineffectual-activation-and-weight-free deep neural network computing," *arXiv preprint arXiv:1705.00125*, 2017.

[27] I. Longchar, A. Varhade, C. Ingle, S. Baranwal, and H. K. Kapoor, "Cluspa: Computation reduction in cnn inference by exploiting clustering and sparsity," in *Proceedings of the Second International Conference on AI-ML Systems*, 2022, pp. 1–8.

[28] A. Yoosefi and M. Kargahi, "Adaptive pooling-based convolution factorization for deploying cnns on energy-constrained iot edge devices," *Microprocessors and Microsystems*, vol. 98, p. 104776, 2023.

[29] Y. Zhang, Z. Jia, H. Du, R. Xue, Z. Shen, and Z. Shao, "A practical highly paralleled reram-based dnn accelerator by reusing weight pattern repetitions," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 41, no. 4, pp. 922–935, 2021.

[30] B. Khaleghi, U. Mallappa, D. Yaldiz, H. Yang, M. Shah, J. Kang, and T. Rosing, "Patternet: explore and exploit filter patterns for efficient deep neural networks," in *Proceedings of the 59th ACM/IEEE Design Automation Conference*, 2022, pp. 223–228.