

Article

Fast Design Space Exploration for Always-On Neural Networks

Jeonghun Kim  and Sunggu Lee *

Department of Electrical Engineering, Pohang University of Science and Technology,
Pohang 37673, Republic of Korea; salmon@postech.ac.kr

* Correspondence: slee@postech.ac.kr

Abstract: An analytical model can quickly predict performance and energy efficiency based on information about the neural network model and neural accelerator architecture, making it ideal for rapid pre-synthesis design space exploration. This paper proposes a new analytical model specifically targeted for convolutional neural networks used in always-on applications. To validate the proposed model, the performance and energy efficiency estimated by the model were compared with actual hardware and post-synthesis gate-level simulations of hardware synthesized with a state-of-the-art electronic design automation (EDA) synthesis tool. Comparisons with hardware created for the Eyeriss neural accelerator showed average execution time and energy consumption error rates of 3.33% and 13.54%, respectively. Comparisons with hardware synthesis results showed an error of 3.18% to 9.44% for two example neural accelerator configurations used to execute MobileNet, EfficientNet, and DarkNet neural network models. Finally, the utility of the proposed model was demonstrated by using it to evaluate the effects of different channel sizes, pruning rates, and batch sizes in several neural network designs for always-on vision, text, and audio processing.

Keywords: design space exploration; neural network compression; analytical model; performance analysis; neural accelerator



Citation: Kim, J.; Lee, S. Fast Design Space Exploration for Always-On Neural Networks. *Electronics* **2024**, *13*, 4971. <https://doi.org/10.3390/electronics13244971>

Academic Editor: Domenico Rosaci

Received: 6 November 2024

Revised: 10 December 2024

Accepted: 16 December 2024

Published: 17 December 2024



Copyright: © 2024 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

1. Introduction

The design of efficient neural networks and low power neural accelerators are both major areas of interest for challenging neural network applications such as real-time vision or audio processing using mobile or edge devices. Although training of neural network parameters is typically conducted using GPU servers due to the extremely large numbers of training samples used (e.g., 1.28 million for ImageNet) and the resulting massive computations required (e.g., computations requiring several weeks even on high-performance multi-node Nvidia GPU server clusters), inference for new real-time test data input can be conducted on extremely low power mobile devices, such as smart ID cards or smart earbuds, since such devices can use pretrained neural networks that only have to process a single image or audio file once every few tens of milliseconds.

For this type of application, a neural accelerator could be used for highly efficient (capable of high performance computation with low energy usage) computation of the matrix operations used in neural network models. Unlike a general-purpose CPU or GPU, a neural accelerator is computing hardware that is specifically designed to efficiently execute neural network models. Since the main computations required in neural network models are matrix multiplication operations, such neural accelerators are typically designed using highly efficient digital or analog multiplier designs implemented in Application Specific Integrated Circuit (ASIC) chips.

When designing a neural network, various parameters of the neural network, such as the channel sizes and pruning rates used, can affect its performance. It is not easy to predict the impact of each design parameter on performance because the impact of such parameters are interdependent. Although simulation methods, such as those proposed in previous studies [1–3], can be used, such methods are excessively slow when implemented at the

transistor or gate-level with clock-cycle accuracy or when applied to large neural network models. Also, previous analytical models rely on non-closed-form equations or are in other aspects unsuitable for accurate analyses of large numbers of candidate neural network models in both nonstreaming and streaming (always-on) applications. Thus, this paper proposes a new analytical model for quickly evaluating possible neural network designs, with various combinations of channel size, pruning rate, batch size, etc., for always-on applications. An always-on application is defined as a system that continuously senses for potential input, with the input to the neural network streamed at regular intervals. Specific examples include object detection for self-driving cars, face recognition for secure building entry, and wakeup keyword recognition for an artificial intelligence assistant.

The analytical model proposed in this paper is different from previous models and methods in several ways. First, it is extremely fast because it is a purely analytical model that does not require simulation-based analysis (several of the previous methods are simulation methods or use simulation-based results in their analysis). Second, it considers neural network batch sizes, pruning rates, and various other parameters that can affect the performance of neural networks. Finally, the proposed model can accurately model neural networks for always-on AI applications with streaming data input.

Figure 1 shows an overview of how the proposed analytical model can be used. A set of candidate neural network models, with various design parameter settings, is evaluated with the aid of a neural accelerator hardware model. The output, for each neural network model candidate with a specific set of design parameter settings, is the expected latency, throughput, and energy usage. If the user's application has specific latency, throughput and energy usage requirements (shown by the blue dashed lines and gray region), then only a few designs (shown with red X marks) may meet the user's requirements.

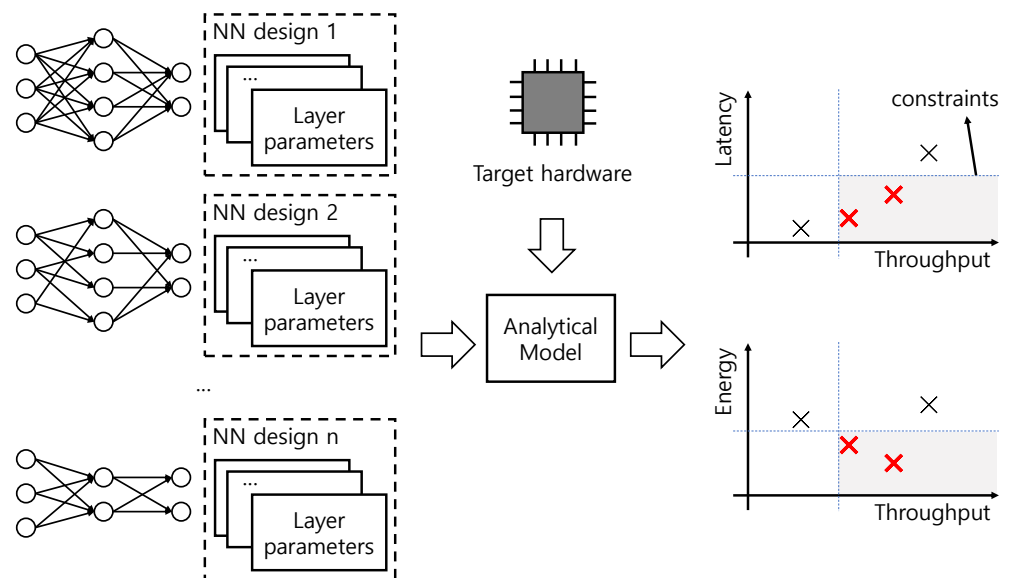


Figure 1. An overview of the proposed analytical model.

The main contributions of this paper are as follows.

1. This paper proposes an analytical model, using prototypical neural accelerator hardware models, which can be used for fast design space exploration of candidate neural network models for a given AI application.
2. The proposed model provides insight into the impact of various neural network model on actual performance.
3. The proposed model can be used for always-on neural network applications.

2. Preliminaries

2.1. Related Work

This section summarizes existing studies that have proposed analytical models according to the following three criteria: (1) whether the model is purely analytical, (2) whether sparsity is considered, and (3) what the hardware target is. In this paper, the term purely analytical means that the model only consists of equations. In many studies, the analytical models used were purely analytical, but some studies used a mixed approach. For example, one method performs simulations or actual measurements on several possible designs and uses those results in an analytical model [4,5]. Additionally, algorithms that cannot be expressed using closed form equations have been proposed in previous analytical models [6]. Although these methods can increase performance estimation accuracy, they have the disadvantage of requiring additional experiments or slowing down performance estimation.

Network sparsity due to pruning can have a significant impact on performance, so modeling the impact of network sparsity on performance is very important. Several studies have proposed models that take sparsity into account [4,7–9], but the hardware targets are often too limited or the models are reliant on compression methods for sparse data.

Neural networks can be executed on many different types of hardware, and it is impossible to propose a single, unified analysis model applicable to all hardware architectures. All analytical models can be classified according to their target hardware architecture, which include CPUs [10], GPUs [5,11], analog in-memory computing accelerators [12], and PE-array based neural accelerators [4,6,13–16]. Applying an analytical model to target hardware not considered in the model may result in inaccurate estimates. Thus, analytical models such as the one proposed in this paper should only be used for fast design space exploration of possible neural network models (before detailed design) for a user's specific target application.

2.2. Model Assumptions

Because there are many different types of neural networks and accelerators, it is difficult to create an accurate model for all cases. Thus, several assumptions were made for modeling. First, it was assumed that PE-array based hardware was used as the core of the neural accelerator architecture. This type of neural accelerator, which consists of a global buffer and multiple PEs, is also sometimes called Eyeriss-like and has been adopted in many neural accelerator studies [17–20]. Although the method proposed in this paper cannot be applied to all existing neural network accelerators, such as analog-based in-memory computing accelerators [21–24], the analytical approach presented here can provide insights into the design of neural networks operating on various hardware platforms. The 2D convolution is one of the main representative neural network operations used in a Convolutional Neural Network (CNN). Matrix–vector multiplication, matrix–matrix multiplication, and element-wise multiplication can all be expressed as special cases of 2D convolution. Therefore, the proposed analytical model can also be applied to LSTM, GRU, and transformer neural network models, which mainly use these types of operations.

2.3. Always-On Neural Networks

In real situations, the data to be processed are not always ready for processing. Many always-on applications are *streaming* multimedia applications, where throughput and latency are the two most important metrics. In these situations, the batch size, which refers to the number of data items (images or frames) processed at once, is usually assumed to be 1. This means that data are processed immediately whenever it is provided. However, a batch size of 1 reduces the opportunities for data reuse. Therefore, energy efficiency may decrease and the maximum number of data items that can be processed per unit time, that is, throughput may decrease.

Since a streaming multimedia application only requires that data be processed at a certain fixed rate, a looser latency requirement can often be accommodated. In such situations, it may be advantageous to wait until data accumulates and then calculate multiple data items at once. For example, in an image detection application, the required throughput could be 24 frames per second, which is the typical video frame rate for a YouTube video, while the required latency could be 0.5 s, which is a time delay that we would find acceptable. Then, the maximum batch size usable would be 12 frames in this example.

Figure 2 is an illustration showing the concepts defined above. With a tight latency requirement, as in part (a), 1 image must be completely processed before the next image can be input. However, with a looser latency requirement, which is frequently the case in multimedia applications, two or more images can be processed at once. In the latter situation, energy can be conserved by processing two or more images at the same time. This is the situation shown in part (b) of the image, which shows the situation with a batch size of two. This suggests that the choice of batch size has a very important impact on latency and throughput when designing an always-on neural network application and that a model to systematically search for and find the best value is needed.

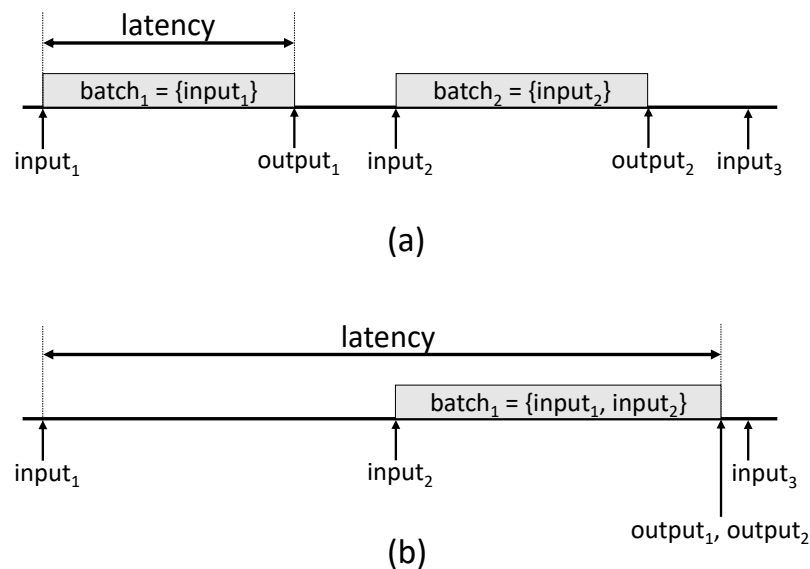


Figure 2. Two example scenarios are shown. Part (a), which uses batch size = 1, outlines an instance where the hardware completely processes an input (image) before the next input can be accepted. However, in part (b), which uses batch size = 2, the hardware waits until two inputs come in and processes two inputs at once.

3. Proposed Analytical Model

In this paper, an analytical model of a PE array-based neural accelerator for always-on neural network designs is proposed. The proposed model can be used to compute the expected latency, throughput, and energy usage for a candidate neural network model, with specific design parameters, by using the procedure shown in Figure 3. Then, the user can repeat these calculations for each neural network model, with various parameter values (such as the batch size), that he/she is interested in.

- Procedure: Compute latency, throughput, and average energy usage
- Input: The layer and hardware parameters in Table 1
1. Compute the number of MAC operation and the number of data elements for each data type (n_{MAC}, n_i, n_o, n_w) given inputs ($N, C_{in}, H_{in}, W_{in}, C_{out}, H_{out}, W_{out}, K_h, K_w$) using Eq. (2) and Eq. (11)
 2. Constraint tiling parameters ($tN, tC_{in}, tH_{in}, tW_{in}, tC_{out}, tH_{out}, tW_{out}$) given inputs ($n_i, n_o, n_w, b_f, b_w, M_i, M_o, M_w, R_i, R_o, R_w$) using Eq. (3)
 3. Compute data reuse parameters of each data type and memory level ($\alpha_i^{(i)}, \alpha_w^{(i)}, \alpha_o^{(i)}$) given inputs ($N, C_{in}, H_{in}, W_{in}, C_{out}, H_{out}, W_{out}, K_h, K_w, tN, tC_{in}, tH_{in}, tW_{in}, tC_{out}, tH_{out}, tW_{out}$) using Eq. (9)
 4. Compute memory access for each data type and memory level ($MA_t^{(i)}$) given inputs ($\alpha_i^{(i)}, \alpha_w^{(i)}, \alpha_o^{(i)}, n_i, n_o, n_w, b_f, b_w, r_f, r_w$) using Eq. (1) and Eq. (14)
 5. Compute the time and energy usage for data transfer for each data type and memory level ($T_{mem,t}^{(i)}, E_{mem,t}^{(i)}$) given inputs ($MA_t^{(i)}, B_{buf}^i, B_{buf}^o, B_{buf}^w, B_{offchip}$) using Eq. (10)
 6. Compute the time and energy usage for arithmetic unit (T_{MAC}, E_{MAC}) given inputs ($n_{MAC}, A_h, A_w, r_f, r_w$) using Eq. (15) and Eq. (16)
 7. Compute the time and energy usage per batch (T_{Total}, E_{Total}) given inputs ($T_{mem,t}^{(i)}, E_{mem,t}^{(i)}, T_{MAC}, E_{MAC}$) using Eq. (17)
 8. Compute the latency, throughput, and average energy usage given inputs ($N, T_{input_interval}, T_{Total}, E_{Total}$) using Eq. (17)–(19)

Figure 3. A step-by-step procedure for calculating latency, throughput, and energy usage using the proposed analytical model.

The proposed analytical model differs from previous studies by including various factors that should be considered when designing a neural network in an always-on application. The proposed model takes a set of design parameters as input. This parameter set consists of layer design parameters and hardware design parameters and is shown in Table 1. The final outputs of the proposed model are the latency, throughput, and energy usage of a given neural network on the given neural accelerator. However, intermediate results such as data reuse and memory access counts are also useful. Using this information, the model can evaluate the effectiveness of a particular neural network on a given neural accelerator.

Table 1. Input parameters of the hardware model.

Type	Description	Symbol
Layer Parameters	Batch size	N
	Input feature map channels	C_{in}
	Input feature map height	H_{in}
	Input feature map width	W_{in}
	output feature map channels	C_{out}
	output feature map height	H_{out}
	output feature map width	W_{out}
	Filter height	K_h
	Filter width	K_w
	Stride	S
	pruning rate of the feature map	r_f
	pruning rate of the weight	r_w
	Feature map bit-width	b_f
	Weight bit-width	b_w
	Input arrival interval	$T_{input_interval}$

Table 1. Cont.

Type	Description	Symbol
Hardware Parameters	Frequency of core	f_{core}
	PE array height	A_h
	PE array width	A_w
	On-chip input buffer size	M_i
	On-chip output buffer size	M_o
	On-chip weight buffer size	M_w
	PE register file size for input	R_i
	PE register file size for output	R_o
	PE register file size for weight	R_w
	On-chip input buffer bandwidth	B_{buffer}^i
	On-chip output buffer bandwidth	B_{buffer}^o
	On-chip weight buffer bandwidth	B_{buffer}^w
	Bandwidth of off-chip memory	$B_{offChip}$

3.1. Modeling Data Movement Between Hierarchical Memories

The memory system is one of the major power-hungry components, sometimes creating a bottleneck and slowing down system performance. Therefore, accurately analyzing data movement between memories is very important in modeling. Most systems using neural accelerators have a hierarchical memory structure. For example, level 3 memory uses DRAM outside the accelerator chip, level 2 memory uses on-chip SRAM, and level 1 memory uses relatively small SRAM or register files inside the PE. If all memory levels have enough memory space, data only need to be moved once for each level. However, in practice, the memory space becomes smaller for lower level memory devices. $MA_t^{(i)}$, which refers to the number of memory accesses of data type $t \in \{i, o, w\}$ at the i -th memory level, can be written using the equation shown below.

$$MA_t^{(i)} = n_t^{(i)} \alpha_t^{(i)} \frac{b_t}{8} \quad (1)$$

$$\begin{aligned} n_i &= N \cdot C_{in} \cdot H_{in} \cdot W_{in} \\ n_o &= N \cdot C_{out} \cdot H_{out} \cdot W_{out} \\ n_w &= C_{out} \cdot C_{in} \cdot K_h \cdot K_w \end{aligned} \quad (2)$$

where $\alpha_t^{(i)}$ is a parameter that indicates how many times data are accessed repeatedly for each data type t at memory level i , and n_t is a parameter that represents the volume of data for each data type t . As described above, α is 1 if there is enough memory space to store the data for all operations. When there is not enough memory space, the operation is divided into several sub-operations, which are called tiling. Because of tiling, weights, input features, or partial sum are read or written from/to memory multiple times, and α increases.

The calculation of α at each memory level is mainly determined by the memory size and tiling method at that level. To systematically formalize the impact of tiling on data reuse, the tiling parameters tN , tC_{in} , tH_{in} , tW_{in} , tC_{out} , tH_{out} , and tW_{out} , representing the sizes of each tile in tiled 2D convolution, are used. Figure 4 and its caption illustrates an example of tiled 2D convolution and tiling parameters. All tiling parameters are integers that do not exceed their corresponding original layer parameters and are constrained by the buffer size. These constraints ensure that the tile size remains within the limits of the buffer capacity. Equation (3) defines the tile size constraints at memory level 1, where the size of the on-chip buffers limits the tile size. At memory level 2, the size of the register

files determines the tile size constraints, causing the left-hand side of the inequalities to change to R_i , R_o , and R_w .

$$\begin{aligned} tN \cdot tH_{in} \cdot tW_{in} \cdot tC_{in} \cdot \frac{b_f}{8} &\leq M_i \\ tN \cdot tH_{out} \cdot tW_{out} \cdot tC_{out} \cdot \frac{b_f}{8} &\leq M_o \\ K_h \cdot K_w \cdot tC_{in} \cdot tC_{out} \cdot \frac{b_w}{8} &\leq M_w \end{aligned} \quad (3)$$

α_i , α_o , and α_w are calculated using the tiling parameters. Considering α_w first, the weight data are divided into a total of $\lceil C_{in}/tC_{in} \rceil \lceil C_{out}/tC_{out} \rceil$ parts. Using this, we can calculate how many times each part is used repeatedly on average.

$$n_{tiles} = \left\lceil \frac{N}{tN} \right\rceil \left\lceil \frac{C_{in}}{tC_{in}} \right\rceil \left\lceil \frac{C_{out}}{tC_{out}} \right\rceil \left\lceil \frac{H_{out}}{tH_{out}} \right\rceil \left\lceil \frac{W_{out}}{tW_{out}} \right\rceil \quad (4)$$

$$\alpha_w = \frac{n_{tiles}}{\left\lceil \frac{C_{in}}{tC_{in}} \right\rceil \left\lceil \frac{C_{out}}{tC_{out}} \right\rceil} = \left\lceil \frac{N}{tN} \right\rceil \left\lceil \frac{H_{out}}{tH_{out}} \right\rceil \left\lceil \frac{W_{out}}{tW_{out}} \right\rceil \quad (5)$$

Let us see the example in Figure 4. In this example, the n_{tile} is $\frac{1}{1} \times \frac{3}{1} \times \frac{3}{1} \times \frac{6}{2} \times \frac{6}{2} = 81$ and the total weight data are divided into $\frac{6}{2} \times \frac{6}{2} = 9$ parts, so α_w is $\frac{81}{9} = 9$.

α_i can be calculated using a similar method. However, it must be considered that each part of the input feature map is overlapped. Some pixels adjacent to each part of the input feature map are included in both parts at the same time. Thus, these pixels are read more frequently than other pixels, which increases α_i . In the example in Figure 4, the original image width W_{in} is 8, but the effective image width $W_{in,eff}$ is $8 + 2 \times 2$ because adjacent tiles overlap by 2 pixels and there are 2 overlapping boundaries. The effective image height $H_{in,eff}$ is calculated in the same manner as $W_{in,eff}$. $H_{in,eff}$, $W_{in,eff}$, and α_i can be written as follows:

$$\begin{aligned} H_{in,eff} &= H_{in} + \left(\left\lceil \frac{H_{in}}{tH_{in}} \right\rceil - 1 \right) \cdot (K_h - S) \\ W_{in,eff} &= W_{in} + \left(\left\lceil \frac{W_{in}}{tW_{in}} \right\rceil - 1 \right) \cdot (K_w - S) \end{aligned} \quad (6)$$

$$\begin{aligned} \alpha_i &= \frac{H_{in,eff} W_{in,eff}}{H_{in} W_{in}} \frac{n_{tiles}}{\left\lceil \frac{N}{tN} \right\rceil \left\lceil \frac{C_{in}}{tC_{in}} \right\rceil \left\lceil \frac{H_{in}}{tH_{in}} \right\rceil \left\lceil \frac{W_{in}}{tW_{in}} \right\rceil} \\ &= \frac{H_{in,eff} W_{in,eff}}{H_{in} W_{in}} \left\lceil \frac{C_{out}}{tC_{out}} \right\rceil \end{aligned} \quad (7)$$

When calculating α_o , both reading data and writing data must be considered. Unlike α_w and α_i , α_o is multiplied by 2; however, the model subtracts 1 from α_o because the partial sum does not need to be read during the first accumulation.

$$\begin{aligned} \alpha_o &= 2 \frac{n_{tiles}}{\left\lceil \frac{N}{tN} \right\rceil \left\lceil \frac{C_{out}}{tC_{out}} \right\rceil \left\lceil \frac{H_{out}}{tH_{out}} \right\rceil \left\lceil \frac{W_{out}}{tW_{out}} \right\rceil} - 1 \\ &= 2 \left\lceil \frac{C_{in}}{tC_{in}} \right\rceil - 1 \end{aligned} \quad (8)$$

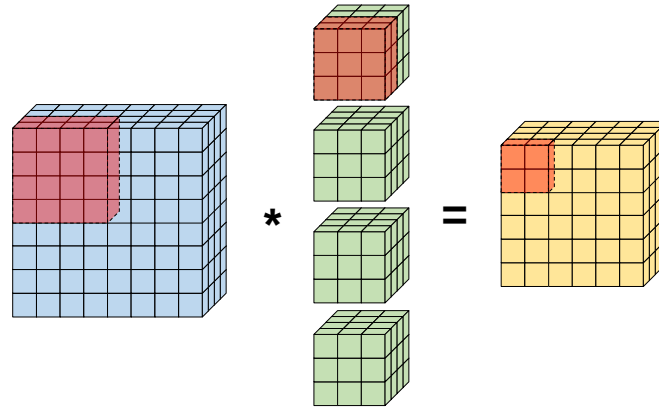


Figure 4. A tiled 2D convolution example. $N = 1$, $C_{in} = 3$, $H_{in} = W_{in} = 8$, $C_{out} = 3$, $H_{out} = W_{out} = 6$, $S = 1$, and $K = 2$. $tN = 1$, $tC_{in} = 1$, $tH_{in} = tW_{in} = 4$, $tC_{out} = 1$, $tH_{out} = tW_{out} = 2$.

Although α_i , α_w , and α_o can initially be calculated in this way, this method may overestimate MA compared to actual calculations. This overestimation occurs due to inter-tile data reuse, where certain data can be reused across consecutive tile calculations. Figure 5 shows an example of inter-tile data reuse. In the top illustration, the first, second, and third tiles use the same weights. When tiles are calculated in this order, the value of α_w becomes 1 because retrieving the weights from off-chip memory for each tile is unnecessary. In the middle illustration, α_i is influenced only by the overlapping regions since the same input feature map is reused. In the bottom illustration, α_o becomes 1 due to the reuse of the same output feature map. To incorporate this inter-tile data reuse into the model, new binary parameters x_1 , x_2 , and x_3 are introduced, and α is redefined as follows.

$$\alpha_w = \begin{cases} \left\lceil \frac{N}{tN} \right\rceil \left\lceil \frac{H_{out}}{tH_{out}} \right\rceil \left\lceil \frac{W_{out}}{tW_{out}} \right\rceil & \text{if } x_1 = 0 \\ 1 & \text{else} \end{cases}$$

$$\alpha_i = \begin{cases} \frac{H_{in,eff} W_{in,eff}}{H_{in} W_{in}} \left\lceil \frac{C_{out}}{tC_{out}} \right\rceil & \text{if } x_2 = 0 \\ \frac{H_{in,eff} W_{in,eff}}{H_{in} W_{in}} & \text{else} \end{cases} \quad (9)$$

$$\alpha_o = \begin{cases} 2 \left\lceil \frac{C_{in}}{tC_{in}} \right\rceil - 1 & \text{if } x_3 = 0 \\ 1 & \text{else} \end{cases}$$

$$x_1 + x_2 + x_3 = 1 \quad x_i \in \{0, 1\}$$

When $x_1 = 1$, the accelerator computes the tiles in the order shown in the top illustration of Figure 5. Similarly, x_2 and x_3 correspond to the middle and bottom illustrations, respectively, in Figure 5.

The time taken to transfer data in the i -th level memory is calculated as MA divided by the memory bandwidth, and the energy consumed at this time is calculated as the product of MA and the unit energy consumed for each access. Unit energy can be obtained using the specifications of the memory used or a memory simulation program such as CACTI [25]. The other main factor for energy usage is idle energy. Because there are many components in the accelerator and their workloads are different, several components have much longer idle time than others. Energy usage during idle time must be considered in order to more accurately estimate the energy usage of a neural network model executing on a neural accelerator.

$$T_{mem,t}^{(i)} = \frac{MA_t^{(i)}}{B_t^{(i)}} \quad (10)$$

$$E_{mem,t}^{(i)} = MA_t^{(i)} \cdot e_{mem,t}^{(i)} + T_{total} \cdot p_{mem,t}^{(i)}$$

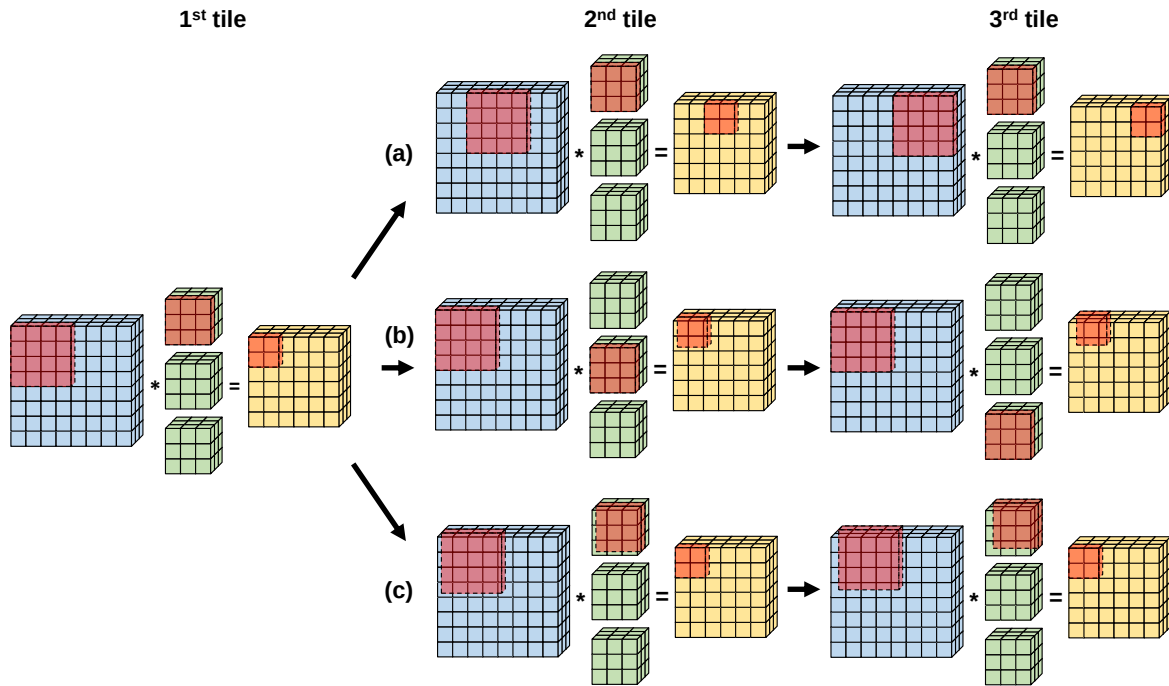


Figure 5. An example of different inter-tile data reuse patterns for a tiled 2D convolution is presented. The parameters are set as follows: $N = 1$, $C_{in} = 3$, $H_{in} = W_{in} = 8$, $C_{out} = 3$, $H_{out} = W_{out} = 6$, $S = 1$, and $K = 2$. For tiling, $tN = 1$, $tC_{in} = 1$, $tH_{in} = tW_{in} = 4$, and $tC_{out} = tH_{out} = tW_{out} = 2$. The three paths marked as (a–c) illustrate different inter-tile reuse patterns, showing the sequences of tiles being computed. A neural accelerator reads input and output image pixels and weight values from off-chip memory to compute the first tile, marked in red in the leftmost part of the figure. In path (a), the weight values are reused in the second and third tiles, so the accelerator only reads input and output image pixels from off-chip memory. Similarly, in path (b), input image pixels are reused; in path (c), output image pixels are reused. Depending on which data the accelerator reuses, its data reuse characteristics will vary, thereby significantly impacting its performance.

3.2. Modeling Execution Efficiency for the Arithmetic Unit

Hardware components other than memory are also important for accurate and practical execution efficiency analysis models. In particular, arithmetic operation circuits have a large impact on latency and throughput in always-on neural network programs. Neural acceleration hardware based on PE arrays is assumed to have one computational circuit for each PE. Therefore, the maximum computation throughput is $Ah \times Aw$ per cycle, so the time required to perform all operations can be calculated as

$$n_{MAC} = NC_{out}H_{out}W_{out}C_{in}K_hK_w \quad (11)$$

$$T_{MAC} = \frac{n_{MAC}}{A_h A_w U_{spatial}} \quad (12)$$

$$U_{temporal} = T_{MAC} / T_{total}$$

This calculation assumes that all PEs perform operations during all cycles. However, during actual neural network operation, not all PEs are used and may not perform operations in every cycle. Therefore, spatial utilization and temporal utilization should be considered. Spatial utilization is determined in the mapping phase and during tiling. Temporal utilization usually falls when data reuse is low and data input rate is slow.

In order to build a model considering memory speed and computation speed, we first calculate the execution time when the memory is sufficiently fast, that is, assuming that

temporal utilization is 100%. Then, the execution time of the entire system is calculated by considering the memory time and the operation time together.

$$E_{MAC} = n_{MAC} \cdot e_{MAC} + T_{total} \cdot p_{MAC} \quad (13)$$

3.3. Effect of Sparse Network

Most energy efficient neural network designs are sparse networks due to pruning, and low-power neural accelerators exploit the sparsity to improve energy efficiency. There are many different ways to compress a sparse network [26], but most compression methods effectively only store non-zero weight values. At this time, in order to maintain the original information, additional information other than the weight value must be stored, which is called overhead.

No specific compression method is assumed in the proposed model. $MA_{overhead}$ is a lower bound computed by information theory, assuming that the location of non-zero values is random. If the pruning is not random (for example, when using structural pruning), the actual overhead may be lower than the lower bound calculated here. However, this situation does not occur often, and even if it does occur, it is difficult to achieve a high pruning rate in structural pruning. Therefore, $MA_{overhead}$ is not dominant in the MA calculation.

$$\begin{aligned} MA_{compress} &= MA_{nonZero} + MA_{overhead} \\ MA_{nonZero} &= MA \cdot (1 - r) \\ MA_{overhead} &= \frac{MA}{b} \left(r \log_2 \frac{1}{r} + (1 - r) \log_2 \frac{1}{1 - r} \right) \end{aligned} \quad (14)$$

Pruning also affects the computation cycles and energy consumed in the PE and the MA, depending on how the hardware is implemented. For example, in the case of Eyeriss [17], when performing a MAC operation, if the value to be multiplied is 0, the MAC input data are blocked to reduce the energy used in the MAC operation. This case can be modeled by changing the energy equation as shown below.

As another example, ZeNa [27] not only blocks data, but also skips unnecessary operations to reduce the cycle time itself. In this case, T_{MAC} and E_{MAC} , which represent the time and energy required for MAC operation, can be modeled as follows.

$$E_{MAC} = n_{MAC}(1 - r_f)(1 - r_w) \cdot e_{MAC} + T_{total} \cdot p_{MAC} \quad (15)$$

$$T_{MAC} = \frac{n_{MAC}(1 - r_f)(1 - r_w)}{A_h A_w U_{spatial}} \quad (16)$$

After modeling the time and energy usage caused by memory transfers and arithmetic units while considering sparsity, the time and energy consumption per batch can finally be formulated as follows. E_{avg} represents the average energy usage per single input data, which can be obtained simply.

$$\begin{aligned} T_{total} &= \sum_t T_{mem,t}^{(1)} + \max(T_{MAC}, \max_{i \neq 1} \max_t T_{mem,t}^{(i)}) \\ E_{total} &= \sum_i \sum_t E_{mem,t}^{(i)} + E_{MAC} \\ E_{avg} &= E_{total} / N \end{aligned} \quad (17)$$

In this equation, a simple summation is used to calculate energy, while the calculation of time applies summation only to the time incurred at memory level 1, with the remaining levels utilizing the max operation. The use of the max operation is justified by the assumption that the hardware modeled here employs processing elements (PEs) that operate

independently via FIFO for data transfer, and on-chip buffers utilize distinct datapaths for input, weight, and output. Notably, if double buffering is assumed, the max operation would also be applied to the time incurred during access to off-chip memory.

3.4. Impact of Always-On Constraints on Energy Efficiency

Always-on neural network design applications differ from other neural network design applications in that the neural network needs to be continuously processing data *all* of the time. This can require extremely lightweight neural network models using extreme network compression techniques when used in energy-constrained environments such as in mobile IoT devices. For the neural network models used in these devices, a consideration is the relationship between batch size, latency, and throughput. Since a large batch size typically leads to a higher possibility of data reuse, the batch size is one of the most important factors to be considered when estimating hardware execution efficiency in always-on systems.

Since always-on neural network programs have to perform neural network operations on incoming data in real time, a batch size of one is often used. However, the smaller the batch size, the lower the possibility of data reuse, resulting in lower energy efficiency and increased execution time due to memory bottlenecks. In particular, if the FPS required by the service is high, there is a possibility that the requested FPS cannot be met when batch size is 1. In this case, one can use a method that does not process the incoming images immediately, but instead waits until a certain number of images are gathered, and then calculates the collected images as one batch. However, if the hardware waits for data to be gathered, it can become difficult to satisfy the latency constraint. So, a neural network application designer must carefully select the appropriate batch size. The relationship between batch size N , input data interval $T_{input_interval}$, latency, and throughput is as follows:

$$(\text{Throughput}) = \frac{N}{T_{total}} \quad (18)$$

$$(\text{Latency}) = T_{total} + (N - 1)T_{input_interval} \quad (19)$$

4. Model Validation

4.1. Comparison with Hardware Measurements

To validate the proposed model, we first compared the execution time and energy usage estimated by this model with values measured using real hardware, a chip created for the Eyeriss neural accelerator [17]. The hardware parameters used in Eyeriss are summarized in Table 2. We estimated the execution time using the proposed model and compared it with the measured values reported for Eyeriss [17]. Table 3 shows the layer parameters in the AlexNet [28] neural network model, for which specific measurement data were reported in [17]. In Eyeriss, pruning was applied to the basic AlexNet model using the r_f values shown in the last column of Table 3.

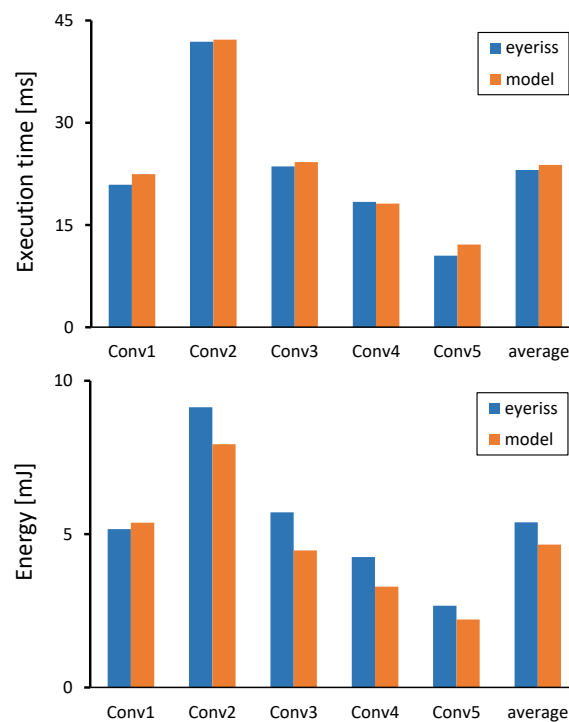
Table 2. Hardware parameters of Eyeriss.

Symbol	Value	Symbol	Value
f_{core}	200 MHz	M_o	50 KB
$A_h \times A_w$	12×14	M_w	8 KB
R_i	24 B	B_{buffer}^i	1.6 GB/s
R_o	48 B	B_{buffer}^o	1.6 GB/s
R_w	448 B	B_{buffer}^w	1.6 GB/s
M_i	50 KB	$B_{offChip}$	0.48 GB/s

Table 3. Layer parameters of AlexNet.

Layer	H_{in}	W_{in}	W_{out}	C_{in}	H_{out}	C_{out}	K_h	K_w	S	r_f
Conv1	227	227	3	55	55	96	11	11	4	0.0001
Conv2	31	31	48	27	27	256	5	5	1	0.387
Conv3	15	15	256	13	13	384	3	3	1	0.725
Conv4	15	15	192	13	13	384	3	3	1	0.793
Conv5	15	15	192	13	13	256	3	3	1	0.776

Figure 6 compares the execution time and energy usage values of AlexNet executed on the Eyeriss hardware, calculated using the proposed model, with the measurements obtained directly from the Eyeriss paper. The model accurately estimated the average execution time with an error of 3.33%. The maximum of the layerwise error was 15.61% and the mean was 5.58%. The correlation between the two was 0.998. For energy consumption, the model estimated the average energy consumption with an error of 13.54%. The maximum of the layerwise error was 22.60% and the mean was 15.66%. The correlation was 0.968. The reasons for the differences between the measured values of Eyeriss and the proposed model can be attributed to the differences in actual implementation details versus the general implementation methods assumed in the proposed model and Eyeriss.

**Figure 6.** The layerwise execution time and energy consumption of AlexNet.

4.2. Comparison with Gate-Level Hardware Simulation

To validate the analytical model proposed in the previous section, the execution efficiency estimated using the proposed model was compared with the execution efficiency estimated by post-synthesis gate-level simulation of the hardware synthesized using state-of-the-art synthesis and simulation tools. The hardware used for this validation comparison is a PE array-based neural accelerator that contains one MAC operator per PE. Synopsys Design Compiler was used for synthesis, and Synopsys PrimePower was used for energy usage evaluation. Since the synthesized design does not include off-chip memory, the energy usage value used for comparison was the energy usage estimate of the analytical model minus $E_{offChip}$. A Samsung 28 nm standard cell library was used during RTL synthesis and energy usage analysis.

Two different sizes of hardware were selected so that the model could be validated on different hardware scales. One is hardware with a PE array size of 12×14 . This is the same size as the PE array used by Eyeriss [17]. The other is hardware with a PE size of 16×16 . The size of the input and output feature map buffers on the first hardware platform is 32 KB each; on the second platform, its 64 KB each. Other hardware parameters include $f_{core} = 100$ MHz, $b_f = b_w = 16$ bits, $R_i = 32$ Byte, $R_o = 64$ Byte, $R_w = 512$ Byte, $B_{buffer}^i = B_{buffer}^w = B_{buffer}^o = 0.2$ GB/s, $B_{offChip} = 0.2$ GB/s and $M_w = 8$ KB.

MobileNet, EfficientNet and DarkNet were used for the target neural network structure. MobileNet [29] and EfficientNet [30] are representative CNN structure that use depthwise and pointwise convolutional layers to conserve the total number of weight parameters that need to be used. DarkNet is the backbone neural network used for feature extraction in YOLOv3, a real-time object detection system. It is assumed that neither the feature maps nor the weights are pruned during the calculation process. The results of comparing the execution time and energy usage for each network are shown in Figure 7. Using the 12×14 PE array, the percentage error averages for the three networks were 7.23% for execution time and 9.44% for energy usage, and 3.18% and 7.06%, respectively, for the 16×16 PE array.

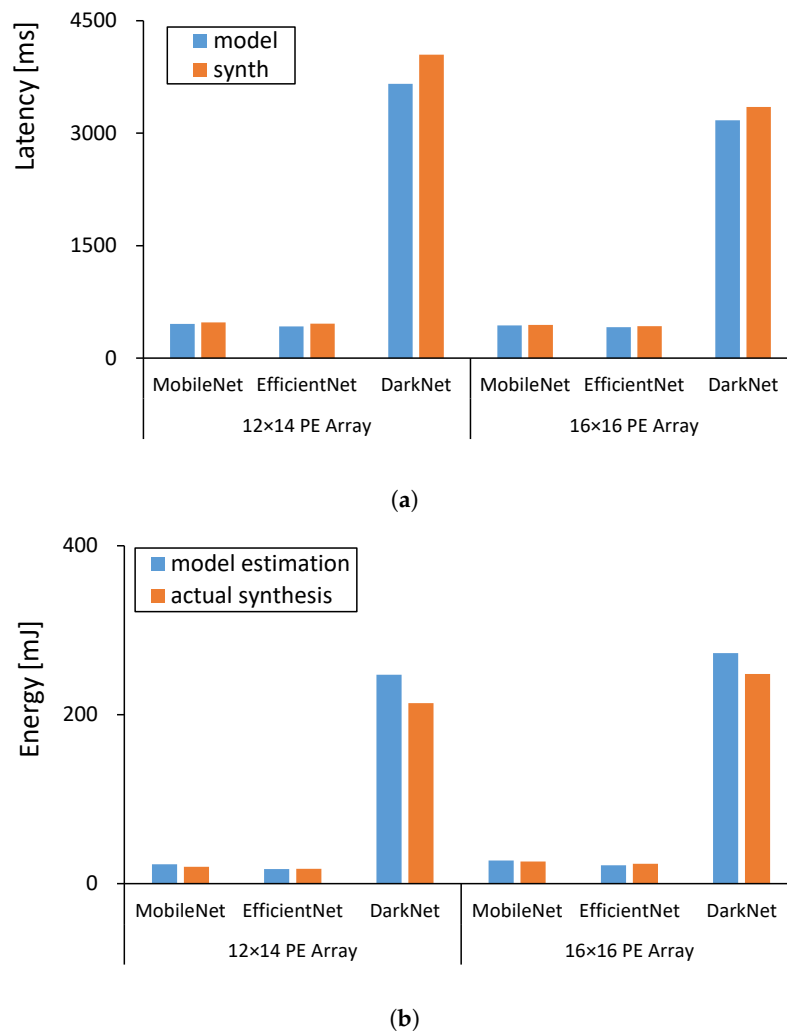


Figure 7. Comparison of performance estimates of proposed model and gate-level simulation for three different neural networks: (a) latency of different networks and PE array size; (b) energy usage of different networks and PE array size. To measure simulation estimates, hardware written in Verilog was synthesized with the Synopsys Design compiler and energy was measured using the Synopsys PrimePower tool.

Table 4 shows a direct comparison of the computation runtimes for a simulation based method and the proposed analytical model. The gate-level simulation method required drastically longer computation times than the analytical model-based method, which required less than one millisecond for all neural network models tested. The computer used for RTL synthesis and gate-level simulation was a Linux server computer running CentOS release 6.10 with a 2.2 GHz AMD Ryzen Threadripper 2990WX CPU with 256 GB memory and 32 cores.

Table 4. Runtime comparison of RTL synthesis and simulation and the analytical model.

HW Config.	Network	Synth. & Sim. [min]	Modeling [ms]	Speedup [Millions]
12 × 14 PE array	MobileNet	251.2	0.019	×793
	EfficientNet	389.7	0.026	×899
	DarkNet	725.8	0.010	×4355
16 × 16 PE array	MobileNet	549.7	0.012	×2749
	EfficientNet	876.0	0.020	×2628
	DarkNet	1616.3	0.011	×8816

The results clearly demonstrate that the proposed analytical model is not merely a replacement for time-consuming simulations but an essential tool in the design of hardware-efficient neural networks. Its practical utility is especially evident in high-dimensional design spaces.

For example, consider the task of configuring a single layer where the number of input channels, the number of output channels, and the pruning rate need to be determined. This involves selecting a design point within a three-dimensional design space. Even if each parameter has only five options, designing a single layer requires testing $5^3 = 125$ configurations. Given that a neural network typically consists of dozens of layers, it is practically impossible to test all combinations through simulations.

In contrast, the proposed analytical model enables testing at speeds hundreds to thousands of times faster than simulations. This makes it highly effective in efficiently exploring the complex relationships between neural network design parameters and hardware design parameters, significantly accelerating the process of identifying optimal designs.

4.3. Comparison with a State-of-the-Art Analytical Model

For validation, the proposed method also compared with a state-of-the-art analytical hardware model, Sparseloop [7], which is an analytical model for evaluation of neural accelerators. Sparseloop is an analytical model applicable to sparse neural networks, and the hardware it targets is similar to the hardware targeted by the proposed model. Other analytical models cannot be applied to sparse neural networks or their hardware targets are quite different from the hardware targeted in the proposed paper, making comparison difficult.

First, when the same hardware and neural network were used as inputs for the two models, the energy that changed depending on the pruning rate was compared. Eyeriss was used as the target hardware, and Alexnet was used as the target neural network. Figure 8a shows that the two methods similarly predict the energy reduction due to sparsity. The trends of the two methods are almost the same. The reason for the slight difference is that while Sparseloop assumes that the number of overhead bits that occur when compressing data with sparsity is fixed, the proposed model assumes that the number of overhead bits changes depending on the degree of sparsity.

Next, the PE array size of the hardware was changed while other configurations were fixed. The PE array size was varied from 8×8 to 20×20 . Figure 8b shows the results. Although there is a slight difference, it is confirmed that the tendency is not significantly different.

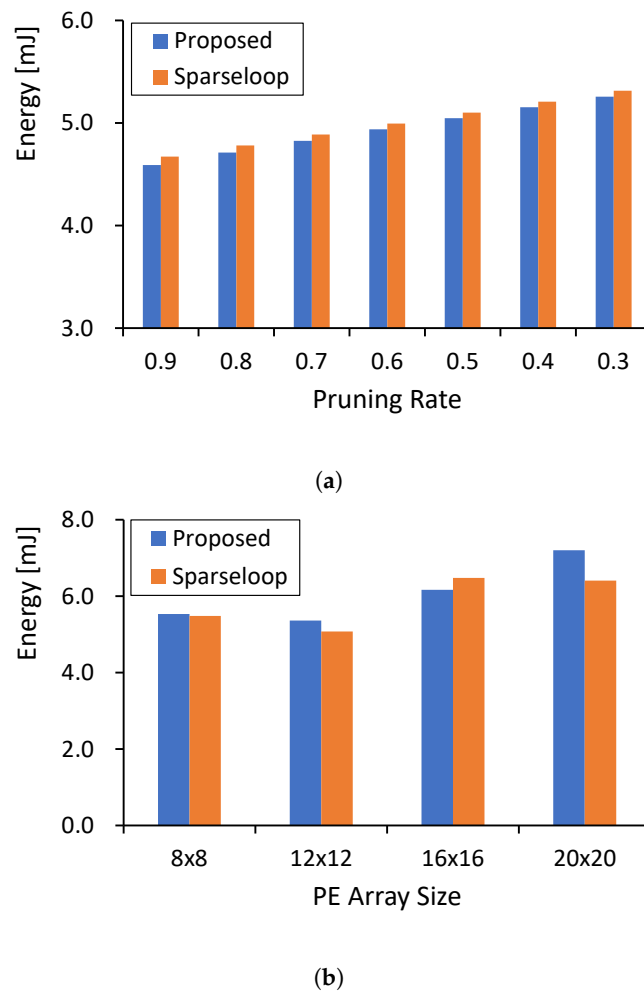


Figure 8. Comparison of the results of the proposed model and Sparseloop: (a) pruning rate vs. energy; (b) PE array size vs. energy.

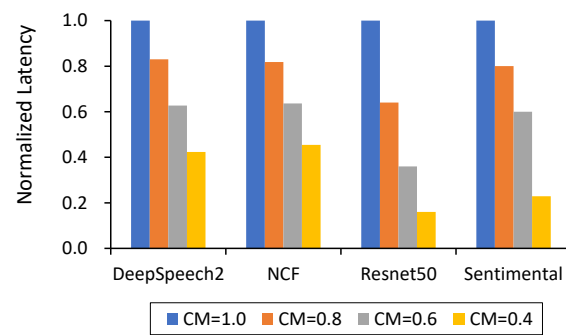
5. Case Studies

The utility of the proposed analytical model is demonstrated in this section by showing how it can be used (1) to analyze the execution efficiency of possible neural network topology candidates in general and (2) to select appropriate batch and channel sizes to use in an always-on neural network application.

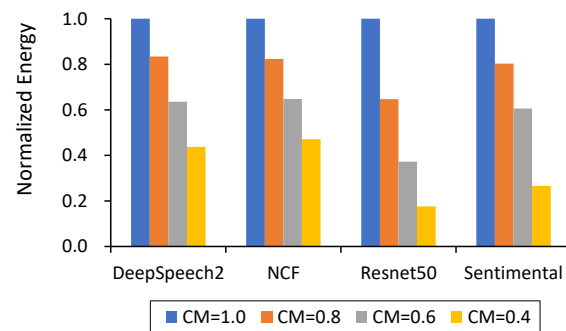
5.1. Use of Model to Analyze Various Network Topologies

First, the analytical model was applied to several representative neural network applications, including vision, text, and audio processing. ResNet50 [31] was used for a neural network for a vision task, a CNN model for sentimental classification [32] was used for a text analysis task, Deep speech 2 [33] was used for a speech recognition task, and NCF [34] for was used as a recommendation system that processes sparse data.

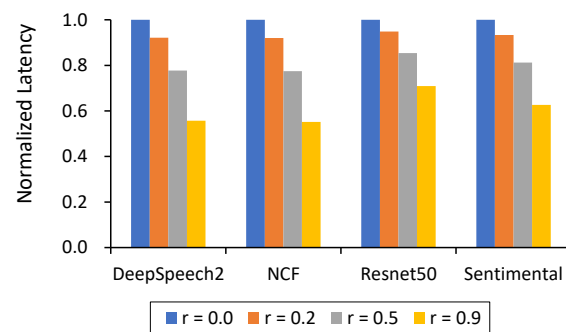
For a given neural network topology, in addition to the basic structure, variations of the basic structure created by changing the channel multiplier and pruning rate are analyzed to determine how each design parameter affects the execution efficiency. Figure 9 shows the change in normalized execution time and energy usage when various degrees of channel multiplier and pruning rate are applied to four different neural network topologies.



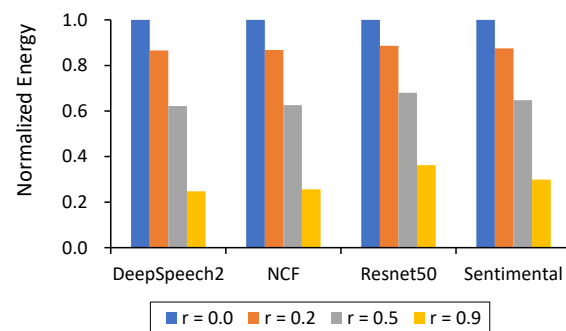
(a)



(b)



(c)



(d)

Figure 9. Effects of channel size and pruning rate on execution time and energy usage in various neural network architectures. CM represents the channel multiplier, and r represents the pruning rate. (a) The latency for various networks with respect to channel multipliers. (b) The energy usage for various networks with respect to channel multipliers. (c) The latency for various networks with respect to pruning rates. (d) The energy usage for various networks with respect to pruning rates.

In particular, looking at the effect of the channel multiplier, it can be seen that the effect on execution efficiency varies significantly depending on the structure of the neural network even when the same parameter is changed. In the case of ResNet50, it is a neural network structure in which a general CNN layer is mainly used, and it can be seen that the execution time is proportional to the square according to the channel multiplier. On the other hand, in the case of Deepspeech and NCF recommendation systems, which consist of layers with relatively small computational intensity such as LSTM and fully connected layers, it can be seen that the execution time is approximately linearly proportional to the channel multiplier. Lastly, the results of the Sentimental seqCNN structure changed dramatically when the channel multiplier changed from 0.4 to 0.6. Such rapid changes are often seen in hardware with a hierarchical memory structure, such as the CPU's cache. When the pruning rate changes, the graph shows that energy changes more than latency. This is because the number of accesses to memory, which has high energy usage, has been greatly reduced due to sparsity.

This experiment demonstrated that the proposed analytical model can capture the varying impact of the same parameter changes on performance depending on the neural network structure. This indicates that the proposed model accurately represents the complex and unpredictable correlations among various design parameters. Furthermore, in practical neural network design, these results can provide insights into which compression methods to prioritize or which layer structures to use.

5.2. Use of Model to Determine Design Parameters

Given a specific choice of a neural network topology, the proposed model can be used to determine the most appropriate design parameters for the selected neural network. As an example, let us consider the determination of the channel sizes and batch sizes to use for a specific always-on application.

Various goals can exist in always-on neural network design. For example, the goal may be to reduce latency with a specified minimum throughput constraint. Or conversely, only the upper boundary of latency is fixed and the design goal may be to increase the throughput as much as possible. Moreover, in a situation where both throughput and latency boundary are specified, the design goal may be to increase the accuracy of the neural network or to reduce the energy usage. The proposed model can be used in all such cases.

First, an experiment was performed to confirm the effect of batch size on throughput and latency. The difference in data reuse between different batch sizes is particularly significant when weight reuse is low. For example, in the case of a fully connected layer, reuse of weight data is considerably low because each weight value is used only once each time one batch is calculated. Figure 10 shows how throughput and latency values change with batch size for fully connected layers.

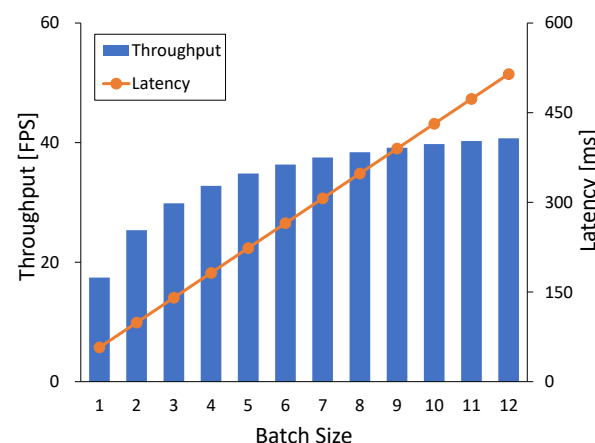


Figure 10. Throughput and latency vs. batch size.

As the batch size is increased, latency increases almost linearly because the time spent waiting for the data in a single batch is the main factor in latency. On the other hand, the throughput gradually increases at an increasingly decreased rate before becoming saturated. This implies that when batch size is small, the hardware resources are not utilized enough because of a memory bottleneck problem caused by low data reuse. As the batch increases, the data reuse increases and the hardware resources operate in parallel. If the batch grows to a certain extent and all resources are sufficiently utilized in parallel, the throughput does not increase as much and eventually becomes saturated.

Now, let us look at an example of a real neural network design problem. Let us assume that throughput must satisfy at least 50 FPS and latency must satisfy at most 150 ms. This will correspond to the green region shown in Figure 10.

Figure 11 shows throughput and latency when varying channel size and batch size. The proposed analytical model can be used to find acceptable ranges of channel sizes and batch sizes that satisfy the specific throughput and latency constraints of the target application. Additional experiments can then be conducted by the user (using, for instance, a PyTorch framework with their test data) in order to determine if the user's accuracy requirements can be met with one or more those channel and batch size combinations. As shown in the figure, when the channel multiplier is 1 to 0.9, the throughput/latency condition cannot be satisfied with any batch size. However, when the channel multiplier is 0.8, the condition can be satisfied by using any batch size between 2 and 4. Note that if batch size is fixed to one, the throughput/latency condition can be satisfied for the first time when the channel multiplier is 0.6. This is significant because a streaming always-on application will tend to achieve higher accuracy when multiple consecutive data items can be used to determine the output of the neural network (i.e., the determination of the meaning of the current data are aided by information about the immediate past data). Thus, in order to consider more than two data items at once, the proposed analytical model tells us that this neural network requires a channel multiplier larger than 0.6.

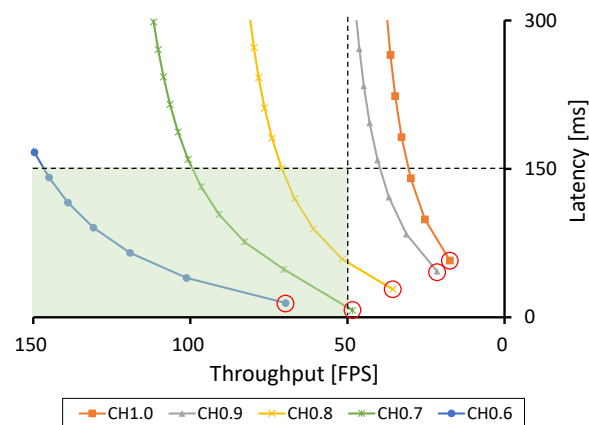


Figure 11. Throughput and latency with different channel multipliers and batch sizes. Each line represents the change in throughput and latency when the channel multiplier is fixed and the batch size changes. The red circle in each line represents the result when the batch size is 1, and each successive point in a line corresponds to an increase in batch size of 1. The green rectangular area shows the acceptable throughput/latency region.

6. Conclusions

This paper has introduced a new analytical model that can be used to evaluate the performance and energy usage of always-on neural network applications on a prototypical PE-array-based neural accelerator. The model can quickly and accurately estimate the hardware execution efficiency of a neural network, helping to quickly explore the high-dimensional design space of neural network topologies and design parameters for designs that satisfy specific latency, throughput, and energy usage constraints. Comparisons of the

model with actual hardware created for the Eyeriss neural accelerator showed relatively high accuracy levels (errors of 3.33 to 13.54 percent). When the model was compared with simulation results of actual gate-level synthesized hardware, it was possible to reduce the simulation time from several hours to under a millisecond while estimating execution efficiency with an error of less than 9.44%. The proposed analytical model can be used as a fast pre-synthesis design space exploration tool for AI designers interested in new network models for their specific applications, including those applications meant to be used in always-on scenarios with specific latency, throughput, and energy usage constraints.

Author Contributions: Conceptualization, J.K.; methodology, J.K.; software, J.K.; validation, J.K.; formal analysis, J.K.; investigation, J.K. and S.L.; writing—original draft preparation, J.K.; writing—review and editing, S.L.; visualization, J.K.; supervision, S.L. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: The data presented in this study are available upon request after the publication of this paper.

Conflicts of Interest: The authors declare no conflicts of interest.

References

- Samajdar, A.; Joseph, J.M.; Zhu, Y.; Whatmough, P.; Mattina, M.; Krishna, T. A systematic methodology for characterizing scalability of DNN accelerators using SCALE-Sim. In Proceedings of the 2020 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), Virtual, 23–26 August 2020; IEEE: Piscataway Township, NJ, USA, 2020; pp. 58–68.
- Muñoz-Martínez, F.; Abellán, J.L.; Acacio, M.E.; Krishna, T. Stonne: Enabling cycle-level microarchitectural simulation for dnn inference accelerators. In Proceedings of the 2021 IEEE International Symposium on Workload Characterization (IISWC), Storrs, CT, USA, 7–9 November 2021; IEEE: Piscataway Township, NJ, USA, 2021; pp. 201–213.
- Xi, S.; Yao, Y.; Bhardwaj, K.; Whatmough, P.; Wei, G.Y.; Brooks, D. SMAUG: End-to-end full-stack simulation infrastructure for deep learning workloads. *ACM Trans. Archit. Code Optim. (TACO)* **2020**, *17*, 1–26. [\[CrossRef\]](#)
- Kang, J.; Jung, D.; Chung, K.; Ha, S. Fast Performance Estimation and Design Space Exploration of Manycore-based Neural Processors. In Proceedings of the 56th Annual Design Automation Conference 2019, Las Vegas, NV, USA, 2–6 June 2019; pp. 1–6.
- Geoffrey, X.Y.; Gao, Y.; Golikov, P.; Pekhimenko, G. Habitat: A {Runtime-Based} computational performance predictor for deep neural network training. In Proceedings of the 2021 USENIX Annual Technical Conference (USENIX ATC 21), Virtual, 14–16 July 2021; pp. 503–521.
- Ma, Y.; Cao, Y.; Vruthula, S.; Seo, J.S. Performance modeling for CNN inference accelerators on FPGA. *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.* **2019**, *39*, 843–856. [\[CrossRef\]](#)
- Wu, Y.N.; Tsai, P.A.; Parashar, A.; Sze, V.; Emer, J.S. Sparseloop: An analytical approach to sparse tensor accelerator modeling. In Proceedings of the 2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO), Chicago, IL, USA, 1–5 October 2022; IEEE: Piscataway Township, NJ, USA, 2022; pp. 1377–1395.
- Ha, W.R.; Kim, H.; Paek, Y. Modeling and Library Support for Early-Stage Exploration of Sparse Tensor Accelerator Designs. *IEEE Access* **2023**, *11*, 55361–55369. [\[CrossRef\]](#)
- Song, J.; Liang, R.; Gong, Y.; Yuan, B.; Hu, J. DiMO-Sparse: Differentiable Modeling and Optimization of Sparse CNN Dataflow and Hardware Architecture. In Proceedings of the 2024 Design, Automation & Test in Europe Conference & Exhibition (DATE), Valencia, Spain, 25–27 March 2024; IEEE: Piscataway Township, NJ, USA, 2024; pp. 1–6.
- Viecke, A.; Pillana, S.; Memeti, S.; Kolodziej, J. Performance Modelling of Deep Learning on Intel Many Integrated Core Architectures. In Proceedings of the 2019 International Conference on High Performance Computing & Simulation (HPCS), Dublin, Ireland, 15–19 July 2019; IEEE: Piscataway Township, NJ, USA, 2019; pp. 724–731.
- Li, Y.; Sun, Y.; Jog, A. Path Forward Beyond Simulators: Fast and Accurate GPU Execution Time Prediction for DNN Workloads. In Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture, New York, NY, USA, 28 October–1 November 2023; MICRO '23, pp. 380–394. [\[CrossRef\]](#)
- Andrulis, T.; Emer, J.S.; Sze, V. CiMLoop: A Flexible, Accurate, and Fast Compute-In-Memory Modeling Tool. In Proceedings of the 2024 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), Indianapolis, IN, USA, 5–7 May 2024; pp. 10–23. [\[CrossRef\]](#)
- Juracy, L.R.; Moreira, M.T.; de Moraes Amory, A.; Hampel, A.F.; Moraes, F.G. A high-level modeling framework for estimating hardware metrics of CNN accelerators. *IEEE Trans. Circuits Syst. I Regul. Pap.* **2021**, *68*, 4783–4795. [\[CrossRef\]](#)
- Xu, R.; Ma, S.; Wang, Y.; Chen, X.; Guo, Y. Configurable Multi-directional Systolic Array Architecture for Convolutional Neural Networks. *ACM Trans. Archit. Code Optim. (TACO)* **2021**, *18*, 1–24. [\[CrossRef\]](#)

15. Stehle, K.; Schindler, G.; Fröning, H. On the Difficulty of Designing Processor Arrays for Deep Neural Networks. In *IoT Streams for Data-Driven Predictive Maintenance and IoT, Edge, and Mobile for Embedded Machine Learning*; Springer: Berlin/Heidelberg, Germany, 2020; pp. 229–240.
16. Mei, L.; Liu, H.; Wu, T.; Sumbul, H.E.; Verhelst, M.; Beigne, E. A uniform latency model for DNN accelerators with diverse architectures and dataflows. In Proceedings of the 2022 Design, Automation & Test in Europe Conference & Exhibition (DATE), Antwerp, Belgium, 14–23 March 2022; IEEE: Piscataway Township, NJ, USA, 2022; pp. 220–225.
17. Chen, Y.H.; Krishna, T.; Emer, J.S.; Sze, V. Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. *IEEE J. Solid-State Circuits* **2016**, *52*, 127–138. [\[CrossRef\]](#)
18. Jo, J.; Cha, S.; Rho, D.; Park, I.C. DSIP: A scalable inference accelerator for convolutional neural networks. *IEEE J. Solid-State Circuits* **2017**, *53*, 605–618. [\[CrossRef\]](#)
19. Hu, X.; Zeng, Y.; Li, Z.; Zheng, X.; Cai, S.; Xiong, X. A resources-efficient configurable accelerator for deep convolutional neural networks. *IEEE Access* **2019**, *7*, 72113–72124. [\[CrossRef\]](#)
20. Jang, J.; Kim, Y.; Lee, J.; Kim, J.J. FIGNA: Integer Unit-Based Accelerator Design for FP-INT GEMM Preserving Numerical Accuracy. In Proceedings of the 2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA), Edinburgh, UK, 2–6 March 2024; IEEE: Piscataway Township, NJ, USA, 2024; pp. 760–773.
21. Yin, S.; Jiang, Z.; Kim, M.; Gupta, T.; Seok, M.; Seo, J.S. Vesti: Energy-efficient in-memory computing accelerator for deep neural networks. *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.* **2019**, *28*, 48–61. [\[CrossRef\]](#)
22. Desoli, G.; Chawla, N.; Boesch, T.; Avodhyawasi, M.; Rawat, H.; Chawla, H.; Abhijith, V.; Zambotti, P.; Sharma, A.; Cappetta, C.; et al. 16.7 A 40-310TOPS/W SRAM-based all-digital up to 4b in-memory computing multi-tiled NN accelerator in FD-SOI 18nm for deep-learning edge applications. In Proceedings of the 2023 IEEE International Solid-State Circuits Conference (ISSCC), San Francisco, CA, USA, 19–23 February 2023; IEEE: Piscataway Township, NJ, USA, 2023; pp. 260–262.
23. Antolini, A.; Paolino, C.; Zavalloni, F.; Lico, A.; Scarselli, E.F.; Mangia, M.; Pareschi, F.; Setti, G.; Rovatti, R.; Torres, M.L.; et al. Combined HW/SW drift and variability mitigation for PCM-based analog in-memory computing for neural network applications. *IEEE J. Emerg. Sel. Top. Circuits Syst.* **2023**, *13*, 395–407. [\[CrossRef\]](#)
24. Kneip, A.; Lefebvre, M.; Verecken, J.; Bol, D. IMPACT: A 1-to-4b 813-TOPS/W 22-nm FD-SOI Compute-in-Memory CNN Accelerator Featuring a 4.2-POPS/W 146-TOPS/mm² CIM-SRAM With Multi-Bit Analog Batch-Normalization. *IEEE J. Solid-State Circuits* **2023**, *58*, 1871–1884. [\[CrossRef\]](#)
25. Wilton, S.J.; Jouppi, N.P. CACTI: An enhanced cache access and cycle time model. *IEEE J. Solid-State Circuits* **1996**, *31*, 677–688. [\[CrossRef\]](#)
26. Han, S.; Mao, H.; Dally, W.J. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv* **2015**, arXiv:1510.00149.
27. Kim, D.; Ahn, J.; Yoo, S. Zena: Zero-aware neural network accelerator. *IEEE Des. Test* **2017**, *35*, 39–46. [\[CrossRef\]](#)
28. Krizhevsky, A.; Sutskever, I.; Hinton, G.E. Imagenet classification with deep convolutional neural networks. *Adv. Neural Inf. Process. Syst.* **2012**, *25*, 1097–1105. [\[CrossRef\]](#)
29. Howard, A.G.; Zhu, M.; Chen, B.; Kalenichenko, D.; Wang, W.; Weyand, T.; Andreetto, M.; Adam, H. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv* **2017**, arXiv:1704.04861.
30. Tan, M.; Le, Q. Efficientnet: Rethinking model scaling for convolutional neural networks. In Proceedings of the International Conference on Machine Learning, PMLR, Long Beach, CA, USA, 9–15 June 2019; pp. 6105–6114.
31. He, K.; Zhang, X.; Ren, S.; Sun, J. Deep residual learning for image recognition. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Las Vegas, NV, USA, 27–30 June 2016; pp. 770–778.
32. Johnson, R.; Zhang, T. Effective use of word order for text categorization with convolutional neural networks. *arXiv* **2014**, arXiv:1412.1058.
33. Amodei, D.; Ananthanarayanan, S.; Anubhai, R.; Bai, J.; Battenberg, E.; Case, C.; Casper, J.; Catanzaro, B.; Cheng, Q.; Chen, G.; et al. Deep speech 2: End-to-end speech recognition in english and mandarin. In Proceedings of the International Conference on Machine Learning, PMLR, New York, NY, USA, 19–24 June 2016; pp. 173–182.
34. He, X.; Liao, L.; Zhang, H.; Nie, L.; Hu, X.; Chua, T.S. Neural collaborative filtering. In Proceedings of the 26th International Conference on World Wide Web, Perth, Australia, 3–7 April 2017; pp. 173–182.

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.