Energy Complexity of Convolutional Neural Networks

Jiří Šíma[1]
sima@cs.cas.cz
Petra Vidnerová[1]
petra@cs.cas.cz
Vojtěch Mrázek[2]
mrazek@fit.vutbr.cz

[1]Institute of Computer Science of the Czech Academy of Sciences, Prague, Czechia
[2]Faculty of Information Technology, Brno University of Technology, Brno, Czechia

Keywords: Convolutional neural networks, energy complexity, dataflow

Abstract

The energy efficiency of hardware implementations of convolutional neural networks (CNNs) is critical to their widespread deployment in low-power mobile devices. Recently, a plethora of methods have been proposed providing energy-optimal mappings of CNNs onto diverse hardware accelerators. Their estimated energy consumption is related to specific implementation details and hardware parameters, which does not allow for machine-independent exploration of CNN energy measures. In this paper, we introduce a simplified theoretical energy complexity model for CNNs, based on only two-level memory hierarchy that captures asymptotically all important sources of energy consumption for different CNN hardware implementations. In this model, we derive a simple energy lower bound and calculate the energy complexity of evaluating a CNN layer for two common dataflows, providing corresponding upper bounds. According to statistical tests, the presented theoretical energy upper and lower bounds fit asymptotically very well the real energy consumption of CNN implementations on the Simba and Eyeriss hardware platforms, estimated by the Timeloop/Accelergy program, which validates the proposed energy complexity model for CNNs.

1 Introduction

Deep neural networks (DNNs) represent a cutting-edge machine learning technology with countless applications in artificial intelligence (AI), including computer vision, speech recognition, natural language processing, robotics, etc. In many cases such as

smart glasses, smartwatches, and mobile phone apps, DNNs have to be implemented in low-power hardware operated on batteries. In contrast, the inference process of already trained DNNs which typically consist of tens of layers, hundreds of thousands of neurons, and tens of millions of weight parameters, is computationally very demanding and highly-energy consuming. Thus, it is often accelerated efficiently in hardware employing massive parallelism in order to meet real-time requirements and energy constraints, which is critical to the widespread deployment of DNNs in mobile AI applications.

Basically, there are two main approaches how to reduce the energy cost of DNNs. The first approach is suitable for error-tolerant applications such as image classification where enormous amount of energy can be saved at the cost of only a small loss in accuracy by using *approximate computing* methods (Armeniakos, Zervakis, Soudris, & Henkel, 2023; Mittal, 2016), e.g. low float precision (Gupta, Agrawal, Gopalakrishnan, & Narayanan, 2015), approximate multipliers (Ansari et al., 2020), etc. The second approach is related to recent major advances in techniques (Sze, Chen, Yang, & Emer, 2017, 2020) that enable energy-efficient DNN processing on a variety of hardware platforms such as GPUs, FPGAs (Mittal, 2020), in-memory computing architectures, which reduce the computational cost of DNNs through *hardware design*.

For a given hardware implementation of DNN, the actual energy consumption of its inference process can be measured or analytically estimated using physical laws. However, it depends on parameters and constants related to the specific hardware architecture and its evaluation varies for different hardware implementations, which prevents from machine-independent exploration of DNN energy measures. Nevertheless, there are software tools that can optimize the energy consumption for a particular DNN on various hardware platforms using different dataflow mapping methods. For example, the Timeloop program (Parashar et al., 2019) maps a convolutional layer specified by its parameters onto a given hardware architecture that is optimal in terms of energy consumption estimated by Accelergy tool (Wu, Emer, & Sze, 2019) which reports the energy statistics.

It has been empirically observed that the energy cost of evaluating DNNs mainly consists of two components, the computation energy and the data energy where the later can be 70% of the total cost (Yang, Chen, Emer, & Sze, 2017). The *computation energy* is needed for performing arithmetic operations, especially the so-called multiply-and-accumulate (MAC) operations ($S \leftarrow S + wy$ on floats S, w, y), which are used for computing weighted sums of inputs in neurons. The *data energy* is required for moving data inside a memory hierarchy (i.e. the dataflow) in hardware implementations of DNNs, which is related to the number of memory accesses.

The aim of this study is to introduce a theoretical hardware-independent model of energy complexity for DNNs that abstracts from their hardware implementation details and ignores specific aspects and constants of real machines, while preserving the asymptotic energy complexity of DNN inference. The use of abstract computational models (such as Turing machines) is fundamental to the field of computational complexity theory to define robust complexity measures, e.g. commonly associated with the usage of the big O notation. Namely, this hardware-independent model will allow to design energy-efficient dataflows providing asymptotic upper bounds on their energy consumption and to establish the principal energy limits that any hardware accelerator must consume by proving corresponding lower bounds. We will validate the universal-

ity of this model by comparing the derived theoretical bounds to the empirical estimates of optimal energy consumptions by particular hardware architectures.

In this paper, we define an energy complexity measure for convolutional neural networks (CNNs) which are widely used DNN models. The computation energy is naturally determined by the number of MACs during the CNN inference, multiplied by a non-uniform circuit constant related to the number of bits in floating-point operations.

To define the data energy of CNN, we introduce an abstract computational model which is composed of only two memory levels called *DRAM* and *Buffer*. The CNN parameters and states are stored in DRAM while arithmetic operations are executed only over numerical data stored in Buffer which is of a limited capacity. The main idea behind this model is that the three arguments of each MAC operation (i.e. float values of an input, weight, and accumulated output) carried out in evaluating a given CNN, must occur together at one time in the arithmetic processor Buffer. This requirement is common to all conceivable hardware implementations of CNNs, making the model universal. The CNN inference thus requires a certain number of data transfers between DRAM and Buffer, which defines the data energy measure.

The proposed model can be used for proving lower bounds on energy complexity of CNNs in order to establish asymptotic limits on energy efficiency of any CNN hardware accelerators. Optimal energy bounds have already been proven for fully-connected layers as a special case of convolutional layers (Šíma & Cabessa, 2023). In this paper, we derive a simple lower bound on the data energy of evaluating general convolutional layers.

Furthermore, we calculate the theoretical energy complexity in this model for two common energy-efficient dataflows under realistic Buffer capacity constraints. For the first dataflow, an output value of each neuron is accumulated in Buffer and written to DRAM only once, and for the second one, any input to each neuron is read into Buffer only once. In both cases, each weight of the CNN is read into Buffer only once. The two dataflows provide upper bounds on energy complexity of the inference process for each convolutional layer separately in terms of its parameters.

These upper bounds on energy complexity are compared to the actual energy consumption of evaluating CNNs on the Simba (Shao et al., 2019) and Eyeriss (Chen, Emer, & Sze, 2016) hardware platforms which is estimated by using the Timeloop/Accelergy software tool. The program optimizes the energy over dataflow mappings onto a given hardware platform. It turns out that the theoretical upper bounds fit asymptotically very well the estimated empirical energy consumption, when the depth, feature map size, filter size, and stride of convolutional layers are varied each separately, which is validated by the statistical linearity and quadraticity tests. Moreover, let us note that these individual upper bounds are optimal for constant Buffer capacity since they match the corresponding lower bounds in this case.

In addition, the minimum real energy consumption of CNN implementations on Simba estimated by Timeloop/Accelergy fits asymptotically the simple energy lower bound also in the case where all the parameters of convolutional layers change simultaneously. Hence, the simplified energy complexity model captures asymptotically all important sources of energy consumption that are common to diverse hardware implementations of CNNs.

A related line of study concerns another energy complexity measure counting the

maximum number of simultaneously active neurons in the course of computation taken over all possible inputs (Šíma, 2014; Uchizawa, Douglas, & Maass, 2006). This is inspired by the fact documented by the fMRI that in biological neural networks the energy cost to transmit a spike is relatively high while the oxygen (energy) supplied to the brain is limited. Hence, the activity of neurons in the brain is quite sparse, with only about 1% of neurons firing at the same time. This is in contrast to a typical design of artificial neural circuits where on average, half of the neurons fire.

The paper is organized as follows. After a formal definition of CNNs in Section 2, the energy complexity model for CNNs is introduced in Section 3 where the computation energy is calculated. In Section 4, a simple lower bound on the data complexity is shown, while Section 5 presents two common energy-efficient dataflows which provide upper bounds on the data energy. Section 6 validates the energy complexity model by comparing the theoretical energy upper and lower bounds for a convolutional layer to its energy consumption estimated by the Timeloop/Accelergy program for the Simba and Eyeriss hardware platforms. Section 7 summarizes the results. A preliminary conference version (Šíma, Vidnerová, & Mrázek, 2023) of this paper is substantially expanded here to include a detailed description of dataflows, a general lower bound on energy complexity, and its experimental validation.

2 Convolutional Neural Networks

In order to define an energy complexity measure, we first formalize and introduce notations for a *convolutional neural network* (CNN) $\mathcal{N}$. The network $\mathcal{N}$ is composed of so-called *macro-units* which are matrices of neurons, representing *feature maps*. The macro-units are grouped into a sequence of $D+1$ disjoint *layers* $\mathcal{N}_\lambda$, indexed by $\lambda = 0, \ldots, D$, starting with the *input* layer $\mathcal{N}_0$, followed by *convolutional* layers, which are at times interlaced with *(max) pooling* layers, and ending with *fully-connected* layers including the last *output* layer $\mathcal{N}_D$. We assume a *multi-layered* architecture of $\mathcal{N}$ in which directed *macro-connections* are between adjacent layers $\mathcal{N}_{\lambda-1}$ and $\mathcal{N}_\lambda$ so that for $\lambda \geq 1$, any macro-unit $g \in \mathcal{N}_{\lambda-1}$ can only be connected through macro-connections to macro-units $f \in \mathcal{N}_\lambda$ in the subsequent layer.

Thus, layer $\mathcal{N}_\lambda$ is composed of $d_\lambda = |\mathcal{N}_\lambda| > 0$ macro-units which are $m_\lambda \times n_\lambda$ matrices of neurons arranged in $m_\lambda > 0$ rows and $n_\lambda > 0$ columns. The parameters m_λ, n_λ, and d_λ are usually called the *height*, *width*, and *depth* of layer $\mathcal{N}_\lambda$ for $\lambda \in \{0, \ldots, D\}$, respectively. For example, an $m_0 \times n_0$-pixel image with three RGB color channels can be presented to $\mathcal{N}$ through $d_0 = 3$ maps in the input layer $\mathcal{N}_0$ of size $m_0 \times n_0$ neurons, each encoding one pixel of the respective channel in the RGB color model.

For $\lambda \geq 1$, a macro-connection from $g \in \mathcal{N}_{\lambda-1}$ to $f \in \mathcal{N}_\lambda$ is a bundle of connections leading from neurons in macro-unit g that form so-called *receptive fields*, to individual neurons in macro-unit f. The receptive fields are rectangular (usually square) local regions of size $r_\lambda \times s_\lambda$, which serve as a scanning window to the feature map represented by macro-unit g in the previous layer $\mathcal{N}_{\lambda-1}$. This scanning window is shifted vertically or horizontally by a so-called *stride* $\sigma_\lambda > 0$ in terms of the number of rows or columns in g, respectively. Namely, to each neuron in macro-unit f, located on the kth row and

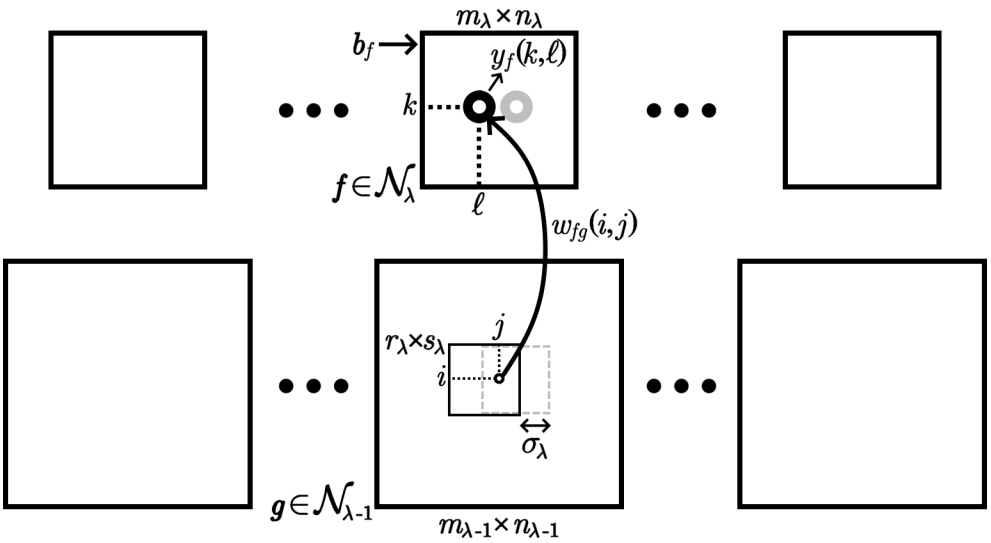


Figure 1: A convolutional layer $\mathcal{N}_\lambda$.

ℓth column where $1 \leq k \leq m_\lambda$ and $1 \leq \ell \leq n_\lambda$, the connections lead from all neurons in an $r_\lambda \times s_\lambda$ submatrix of g with the upper left corner on the row $(k-1)\sigma_\lambda + 1$ and the column $(\ell-1)\sigma_\lambda + 1$. This is illustrated on a convolutional layer which is schematically depicted in Figure 1. Note that the parameters $r_\lambda, s_\lambda, \sigma_\lambda$ are unique to layer $\mathcal{N}_\lambda$.

In order to compensate for underrepresenting the neurons that are located at the edge of feature maps, the macro-units in convolutional layers are usually *padded* by several rows and columns of neurons both from above and below, and to the left and right, respectively. For simplicity, we assume no padding which is sufficient for our energy considerations. It follows that the size $m_\lambda \times n_\lambda$ of feature maps in the λth layer can be calculated as

$$m_\lambda = \left\lceil \frac{m_{\lambda-1} - r_\lambda}{\sigma_\lambda} \right\rceil + 1\,, \quad n_\lambda = \left\lceil \frac{n_{\lambda-1} - s_\lambda}{\sigma_\lambda} \right\rceil + 1 \tag{1}$$

in terms of the size $m_{\lambda-1} \times n_{\lambda-1}$ of feature maps in the preceding layer $\mathcal{N}_{\lambda-1}$.

In the definition of energy complexity, we will consider only the convolutional layers in $\mathcal{N}$ whose indices are collected in $\Gamma \subset \{1, \ldots, D\}$ (including the fully-connected layers as a special case), which are densely interconnected. Namely, for any index $\lambda \in \Gamma$ of convolutional layer, every macro-unit $g \in \mathcal{N}_{\lambda-1}$ is connected to each macro-unit $f \in \mathcal{N}_\lambda$ which thus has $d_{\lambda-1}$ incoming macro-connections. On the other hand, we neglect the sparsely interconnected pooling layers $\mathcal{N}_\lambda$ with $\lambda \notin \Gamma$, whose macro-units f have only one unique incoming macro-connection from a corresponding macro-unit in $\mathcal{N}_{\lambda-1}$ (i.e. $d_\lambda = d_{\lambda-1}$), which represents a feature map that is partitioned into square non-overlapping receptive fields of f (i.e. $\sigma_\lambda = r_\lambda = s_\lambda$). The fully-connected layers $\mathcal{N}_\lambda$ with λ in Γ are composed of single neurons f that constitute trivial feature maps of size 1×1 (i.e. $m_\lambda = n_\lambda = \sigma_\lambda = 1$) satisfying $r_\lambda = s_\lambda = 1$ except for the first fully-connected (so-called *flattening*) layer $\mathcal{N}_{\lambda_1}$ which collects outputs from all neurons in the previous layer $\mathcal{N}_{\lambda_1-1}$, that is, $r_{\lambda_1} = m_{\lambda_1-1}$ and $s_{\lambda_1} = n_{\lambda_1-1}$.

Every macro-unit $f \in \mathcal{N}_\lambda$ for $\lambda \in \Gamma$, is associated with a real *bias* $b_f \in \mathbb{R}$, and

any macro-connection leading from macro-unit $g \in \mathcal{N}_{\lambda-1}$ to f is labeled with a so-called *filter* (or *kernel*) $\mathbf{W}_{fg} \in \mathbb{R}^{r_\lambda \times s_\lambda}$ which is an $r_\lambda \times s_\lambda$ matrix with real entries $w_{fg}(i,j)$ called *weights*, for every $i = 1, \ldots, r_\lambda$ and $j = 1, \ldots, s_\lambda$. The *state (output)* $\mathbf{Y}_f \in \mathbb{R}^{m_\lambda \times n_\lambda}$ of any macro-unit $f \in \mathcal{N}_\lambda$ is an $m_\lambda \times n_\lambda$ matrix with real entries $y_f(k,\ell)$ which are *states (outputs)* of individual neurons in f, for every $k = 1, \ldots, m_\lambda$ and $\ell = 1, \ldots, n_\lambda$.

For a convolutional layer $\mathcal{N}_\lambda$ where $\lambda \in \Gamma$, the states $\mathbf{Y}_f$ of $f \in \mathcal{N}_\lambda$ are evaluated using the following weighted sums

$$S_f(k,\ell) = b_f + \sum_{g \in \mathcal{N}_{\lambda-1}} \sum_{i=1}^{r_\lambda} \sum_{j=1}^{s_\lambda} w_{fg}(i,j)\, y_g\Big((k-1)\sigma_\lambda + i\,, (\ell-1)\sigma_\lambda + j\Big) \tag{2}$$

for every $k = 1, \ldots, m_\lambda$ and $\ell = 1, \ldots, n_\lambda$. Note that the weight $w_{fg}(i,j)$ in (2) is associated with a connection incoming to a neuron located on the kth row and ℓth column in macro-unit $f \in \mathcal{N}_\lambda$ for any $1 \le k \le m_\lambda$ and $1 \le \ell \le n_\lambda$, that lead from a neuron located in the receptive field of f on the row $(k-1)\sigma_\lambda + i$ and the column $(\ell-1)\sigma_\lambda + j$ in macro-unit $g \in \mathcal{N}_{\lambda-1}$. This means that the receptive fields of $f \in \mathcal{N}_\lambda$ in $g \in \mathcal{N}_{\lambda-1}$ share the same filter $\mathbf{W}_{fg}$ associated with the macro-connection from g to f.

Furthermore, for a non-output layer $\mathcal{N}_\lambda$ with $\lambda < D$, the *rectified linear activation function* ReLU is employed as

$$y_f(k,\ell) = \mathrm{ReLU}(S_f(k,\ell)) = \max\big(0, S_f(k,\ell)\big)\,, \tag{3}$$

while for the output layer $\mathcal{N}_D$, the state $\mathbf{Y}_f = (y_f(1,1)) = y_f$ of trivial macro-unit $f \in \mathcal{N}_D$ with $m_D = n_D = 1$ neuron, is computed by the *softmax function* as $y_f = e^{S_f} / (\sum_{h \in \mathcal{N}_D} e^{S_h}) \in (0,1)$ where $S_h = S_h(1,1)$ for every $h \in \mathcal{N}_D$. For completeness note that for a max pooling layer $\mathcal{N}_\lambda$ where $\lambda \notin \Gamma$, which satisfies $\sigma_\lambda = r_\lambda = s_\lambda$, the state $\mathbf{Y}_f$ of $f \in \mathcal{N}_\lambda$ with a single incoming macro-connection from $g \in \mathcal{N}_{\lambda-1}$, is computed as $y_f(k,\ell) = \max_{i,j \in \{1,\ldots,r_\lambda\}} y_g\big((k-1)r_\lambda + i\,, (\ell-1)r_\lambda + j\big)$ for every $k = 1, \ldots, m_\lambda$ and $\ell = 1, \ldots, n_\lambda$.

3 Energy Complexity Model

In this section, we introduce a simplified hardware-independent energy complexity model for evaluating a CNN formalized in Section 2, which captures the main sources of energy consumption in practical hardware implementations of CNNs. This model has a memory hierarchy with only two levels, called *DRAM* and *Buffer*, as schematically depicted in Figure 2.

The DRAM memory has an unlimited capacity (corresponding to a large, slow, and cheap memory) which is used for storing the entire CNN $\mathcal{N}$ including its filters $\mathbf{W}_{fg}$ and current states $\mathbf{Y}_f$ for all macro-units f, g. In contrast, the Buffer memory has a limited capacity of B bits (corresponding to a small, fast, and expensive memory) over which arithmetic operations are executed, especially the *multiply-and-accumulate (MAC)* operations $S \leftarrow S + wy$ for evaluating the weighted sums (2) where w is a filter weight, y is a neuron state from a previous-layer feature map, and S is a partial sum

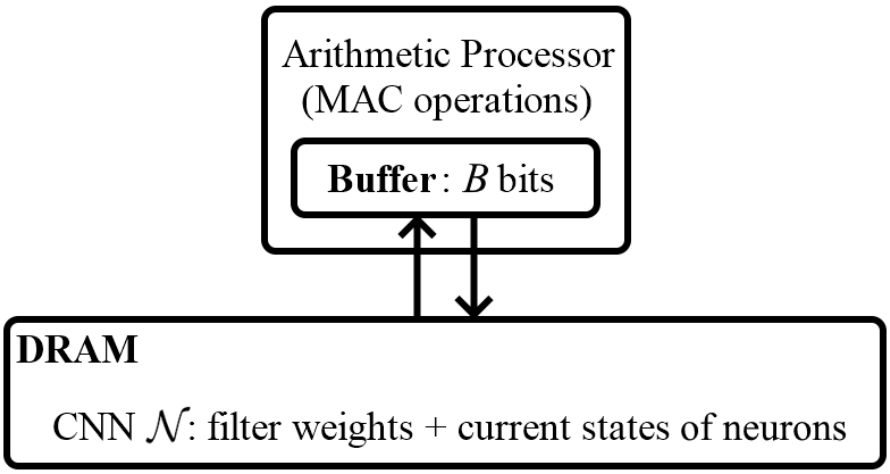


Figure 2: The energy complexity model.

accumulating an output of a current-layer neuron. Thus, in order to perform a MAC operation, the respective float values of its three arguments must simultaneously occur in Buffer which means they must be read from DRAM into Buffer at some point. On the other hand, the results of MACs are later written to DRAM due to the limited Buffer capacity. This requires (read/write) accesses to DRAM memory, which are energy consuming.

As has been discussed in Section 1, the energy complexity of evaluating $\mathcal{N}$ consists of the *computation energy* and the *data energy* (Yang et al., 2017):

$$E = E_{\text{comp}} + E_{\text{data}} \tag{4}$$

which are related to the number of MACs and the number of DRAM accesses, respectively. For simplicity, we do not consider the energy optimization across multiple layers (Alwani, Chen, Ferdman, & Milder, 2016), which means energy complexity (4) is defined as a simple sum of energy costs only over separate convolutional layers in $\mathcal{N}$ (including fully-connected layers), while the less energy-intensive max pooling layers are omitted:

$$E = \sum_{\lambda \in \Gamma} \left(E^{\lambda}_{\text{comp}} + E^{\lambda}_{\text{data}} \right) \tag{5}$$

where $E^{\lambda}_{\text{comp}}$ is the computation energy and $E^{\lambda}_{\text{data}}$ is the data energy for evaluating a convolutional layer $\mathcal{N}_\lambda$ for $\lambda \in \Gamma$. Hereafter, for brevity, the states of single neurons in macro-units in $\mathcal{N}_{\lambda-1}$ are called the *inputs* of layer $\mathcal{N}_\lambda$, the states of single neurons in macro-units in $\mathcal{N}_\lambda$ are called the *outputs* of layer $\mathcal{N}_\lambda$, and the individual filter weights associated with connections between neurons in layers $\mathcal{N}_{\lambda-1}$ and $\mathcal{N}_\lambda$ according to (2), are called the *weights* of $\mathcal{N}_\lambda$.

The computation energy $E^{\lambda}_{\text{comp}}$ is defined as the number of MACs in layer $\mathcal{N}_\lambda$ for $\lambda \in \Gamma$, multiplied by a parameter C_b that depends on the number of bits b in floating-point MAC operations. This dependence is apparently not uniform (e.g. not linear) since the design of a MAC circuit inside a microprocessor differs for each b, which

means there is no program generating a MAC circuit for each b (i.e. a nonuniformity assumption known from circuit complexity theory).

The number of MACs in the λth layer equals to the number $d_\lambda m_\lambda n_\lambda$ of weighted sums $S_f(k,\ell)$ in (2) for all $f \in \mathcal{N}_\lambda$ (where $|\mathcal{N}_\lambda| = d_\lambda$), $k \in \{1,\ldots,m_\lambda\}$, and $\ell \in \{1,\ldots,n_\lambda\}$ (i.e. the number of outputs of $\mathcal{N}_\lambda$), multiplied by the number $d_{\lambda-1}r_\lambda s_\lambda$ of inputs $y_g((k-1)\sigma_\lambda + i\,,(\ell-1)\sigma_\lambda + j)$ of $\mathcal{N}_\lambda$ for all $g \in \mathcal{N}_{\lambda-1}$ (where $|\mathcal{N}_{\lambda-1}| = d_{\lambda-1}$), $i \in \{1,\ldots,r_\lambda\}$, and $j \in \{1,\ldots,s_\lambda\}$, that contribute to each of these sums, which gives

$$E^\lambda_{\text{comp}} = C_b\, d_\lambda m_\lambda n_\lambda\, d_{\lambda-1} r_\lambda s_\lambda\,. \tag{6}$$

The data energy E^λ_{data} is defined as the number of read and write accesses to DRAM when evaluating the λth layer for $\lambda \in \Gamma$, multiplied by the number b of bits in a floating-point representation of numbers to be transferred between DRAM and Buffer. Note that in our model we assume that any floating-point number is transferred as a separate, indivisible, and uncompressed block of b bits. This energy complexity can be split into three components that count the DRAM accesses separately for the inputs, outputs, and weights of $\mathcal{N}_\lambda$:

$$E^\lambda_{\text{data}} = E^\lambda_{\text{inputs}} + E^\lambda_{\text{outputs}} + E^\lambda_{\text{weights}}\,. \tag{7}$$

4 A Simple Lower Bound on Energy Complexity

In this section, we derive a simple lower bound on the data energy E^λ_{data} for any convolutional layer $\mathcal{N}_\lambda$ where $\lambda \in \Gamma$. Obviously, the numbers of inputs, outputs, and weights of $\mathcal{N}_\lambda$ which can be calculated as $d_{\lambda-1}m_{\lambda-1}n_{\lambda-1}$, $d_\lambda m_\lambda n_\lambda$, and $d_\lambda(d_{\lambda-1}r_\lambda s_\lambda + 1)$ (including biases), respectively, altogether provide a trivial lower bound on the data energy complexity of layer $\mathcal{N}_\lambda$:

$$E^\lambda_{\text{data}} \geq b\,(d_{\lambda-1}m_{\lambda-1}n_{\lambda-1} + d_\lambda m_\lambda n_\lambda + d_\lambda(d_{\lambda-1}r_\lambda s_\lambda + 1)) \tag{8}$$

since all the inputs and weights must be read into Buffer at least once and all the evaluated outputs are eventually written to DRAM.

The trivial lower bound (8) can slightly be improved. For this purpose observe that any *two* fixed MAC arguments chosen from the three ones that occur in (2) for $\lambda \in \Gamma$, which are the *inputs* $y_g((k-1)\sigma_\lambda + i\,,(\ell-1)\sigma_\lambda + j)$ for all $g \in \mathcal{N}_{\lambda-1}$, $k \in \{1,\ldots,m_\lambda\}$, $\ell \in \{1,\ldots,n_\lambda\}$, $i \in \{1,\ldots,r_\lambda\}$, and $j \in \{1,\ldots,s_\lambda\}$, the weighted sums $S_f(k,\ell)$ accumulated for the *outputs* $y_f(k,\ell)$ for all $f \in \mathcal{N}_\lambda$, $k \in \{1,\ldots,m_\lambda\}$, and $\ell \in \{1,\ldots,n_\lambda\}$, and the *weights* $w_{fg}(i,j)$ for all $f \in \mathcal{N}_\lambda$, $g \in \mathcal{N}_{\lambda-1}$, $i \in \{1,\ldots,r_\lambda\}$, and $j \in \{1,\ldots,s_\lambda\}$, determine uniquely the remaining third MAC argument in (2) if it exists.

For instance, the sum $S_{f_1}(k_1,\ell_1)$ for some fixed $f_1 \in \mathcal{N}_\lambda$, $k_1 \in \{1,\ldots,m_\lambda\}$, and $\ell_1 \in \{1,\ldots,n_\lambda\}$, and the weight $w_{f_1g_1}(i_1,j_1)$ for some fixed $g_1 \in \mathcal{N}_{\lambda-1}$, $i_1 \in \{1,\ldots,r_\lambda\}$, and $j_1 \in \{1,\ldots,s_\lambda\}$, represent two arguments of the only one MAC operation used to evaluate (2) over the corresponding unique input $y_{g_1}((k_1-1)\sigma_\lambda + i_1\,,(\ell_1-1)\sigma_\lambda + j_1)$ as the third argument. On the other hand, the input $y_{g_1}((k_1-1)\sigma_\lambda + i_1\,,(\ell_1-1)\sigma_\lambda + j_1)$ for some fixed $g_1 \in \mathcal{N}_{\lambda-1}$, $k_1 \in \{1,\ldots,m_\lambda\}$, $\ell_1 \in \{1,\ldots,n_\lambda\}$, $i_1 \in \{1,\ldots,r_\lambda\}$, and $j_1 \in \{1,\ldots,s_\lambda\}$, and the weight $w_{f_1g_1}(i_1,j_2)$ for some fixed

$f_1 \in \mathcal{N}_\lambda$ and $j_2 \in \{1, \ldots, s_\lambda\}$ such that $j_1 \neq j_2$, do not occur together as arguments of any MAC operation, since the sum $S_{f_1}(k_1, \ell_1)$ in (2) does not contain the incompatible term $w_{f_1 g_1}(i_1, j_2) \cdot y_{g_1}((k_1 - 1)\sigma_\lambda + i_1\,, (\ell_1 - 1)\sigma_\lambda + j_1)$.

Suppose that $\beta \geq 3$ floating-point numbers of size b bits can be stored in Buffer which means $B = b \cdot \beta$. It follows from the preceding observation that after reading one number into Buffer, the maximum number of new triple arguments (input, output, weight) of MACs that meet in Buffer which are used for evaluating (2), is $\lfloor (\beta - 1)/2 \rfloor$, since there can be at most $\lfloor (\beta - 1)/2 \rfloor$ pairs of MAC arguments in Buffer combined with the read unique third MAC argument. Hence, the data energy complexity of layer $\mathcal{N}_\lambda$ can be lower bounded by the number of MAC operations divided by the maximum number new MACs enabled by one DRAM access, that is,

$$E^\lambda_{\text{data}} \geq b \cdot \frac{d_\lambda m_\lambda n_\lambda d_{\lambda-1} r_\lambda s_\lambda}{\left\lfloor \frac{\beta-1}{2} \right\rfloor}\,. \tag{9}$$

Particularly for *constant* Buffer capacity B, we obtain the asymptotic lower bound

$$E^\lambda_{\text{data}} = \Omega\left(d_\lambda d_{\lambda-1} m_\lambda n_\lambda r_\lambda s_\lambda\right) \tag{10}$$

which can be rewritten as

$$E^\lambda_{\text{data}} = \Omega\left(\frac{d_\lambda d_{\lambda-1} m_{\lambda-1} n_{\lambda-1} r_\lambda s_\lambda}{\sigma_\lambda^2}\right) \tag{11}$$

due to $m_\lambda n_\lambda \approx m_{\lambda-1} n_{\lambda-1}/\sigma_\lambda^2$ according to (1).

5 Upper Bounds on Energy Complexity

In the following two Sections 5.1 and 5.2, we present two common dataflows for evaluating a convolutional layer $\mathcal{N}_\lambda$ where $\lambda \in \Gamma$, and calculate their theoretical data energy complexity E^λ_{data}, which provides upper bounds on energy. In Section 5.3, these upper bounds are then compared to the lower bounds derived in Section 4.

In the first dataflow, each output is written to DRAM only once and in the second one, each input is read into Buffer only once, while each weight (including bias) is read into Buffer just one time in both dataflows, that is,

$$E^\lambda_{\text{weights}} = b\, d_\lambda (d_{\lambda-1} r_\lambda s_\lambda + 1)\,. \tag{12}$$

We will assume a sufficiently large capacity of Buffer:

$$B = b\,(2 m_\lambda n_\lambda + 1)\,, \tag{13}$$

which is a realistic assumption in practical hardware implementations of CNNs as will be illustrated in Section 6.

For the purpose of describing the dataflows, we define a *partition* of an input feature map $g \in \mathcal{N}_{\lambda-1}$ composed of $m_{\lambda-1} \times n_{\lambda-1}$ neurons,

$$g = \bigcup_{i_0, j_0 \in \{1, \ldots, \sigma_\lambda\}} g(i_0\,, j_0) \tag{14}$$

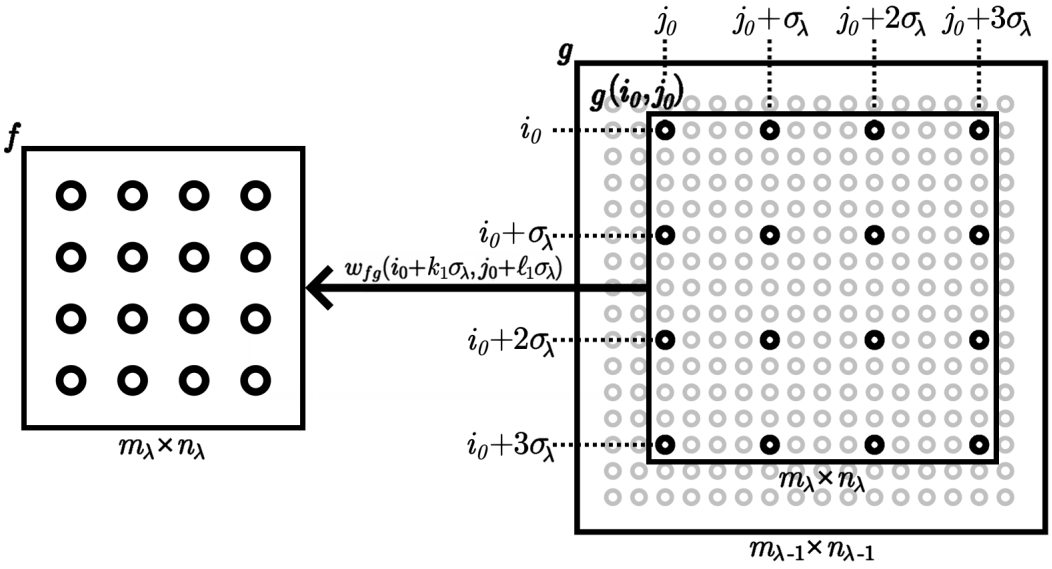


Figure 3: The partition of an input feature map.

into σ_λ^2 disjoint grid submaps

$$g(i_0, j_0) = \left\{ \big(i_0 + (k-1)\sigma_\lambda \,,\, j_0 + (\ell-1)\sigma_\lambda\big) \;\middle|\; \begin{array}{l} k \in \{1, \ldots, m_\lambda\} \\ \ell \in \{1, \ldots, n_\lambda\} \end{array} \right\}, \tag{15}$$

each composed of $m_\lambda \times n_\lambda$ neurons that share the same weights $w_{fg}(i,j)$ in the weighted sum (2) for each $f \in \mathcal{N}_\lambda$, as is schematically depicted Figure 3. These shared weights $w_{fg}(i,j)$ satisfy

$$1 \le i = i_0 + k_1\sigma_\lambda \le r_\lambda \quad \text{and} \quad 1 \le j = j_0 + \ell_1\sigma_\lambda \le s_\lambda \tag{16}$$

for integers $k_1, \ell_1 \in \mathbb{Z}$, since a weight $w_{fg}(i,j)$ is combined with an input $y_g\big((k-1)\sigma_\lambda + i_0\,, (\ell-1)\sigma_\lambda + j_0\big)$ in (2) for some $k \in \{1, \ldots, m_\lambda\}$, $\ell \in \{1, \ldots, n_\lambda\}$, and $i_0, j_0 \in \{1, \ldots, \sigma_\lambda\}$, iff there is $k_2 \in \{1, \ldots, m_\lambda\}$, $\ell_2 \in \{1, \ldots, n_\lambda\}$ such that $(k-1)\sigma_\lambda + i_0 = (k_2-1)\sigma_\lambda + i$ and $(\ell-1)\sigma_\lambda + j_0 = (\ell_2-1)\sigma_\lambda + j$ iff $i = i_0 + k_1\sigma_\lambda$ and $j = j_0 + \ell_1\sigma_\lambda$ where $k_1 = k - k_2$ and $\ell_1 = \ell - \ell_2$.

5.1 The Dataflow with Write-Once Outputs

The dataflow in which each output is written to DRAM only once is described in Algorithm 1. This algorithm is basically composed of five nested *for* loops 1–26, 6–22, 7–21, 12–20, and 16–19. The outermost loop 1–26 goes through all the macro-units $f \in \mathcal{N}_\lambda$ representing the feature maps of convolutional layer $\mathcal{N}_\lambda$ where $\lambda \in \Gamma$, for which the bias b_f is first read (line 2) in order to initialize the $m_\lambda \times n_\lambda$ weighted sums $S_f(k,\ell)$ of feature map f accumulated in Buffer, for all $k \in \{1, \ldots, m_\lambda\}$ and $\ell \in \{1, \ldots, n_\lambda\}$ (lines 3–5). The next two nested loops 6–22 and 7–21 pass through all the macro-units $g \in \mathcal{N}_{\lambda-1}$ representing the *input* feature maps in the $(\lambda-1)$th layer, and through all the grid submaps $g(i_0, j_0)$ parameterized by $i_0, j_0 \in \{1, \ldots, \sigma_\lambda\}$ from the partition (14) of

Algorithm 1 The dataflow with write-once outputs and $B = b(2m_\lambda n_\lambda + 1)$.

1: **for all** feature maps $f \in \mathcal{N}_\lambda$ **do**
2: **read** bias b_f into `Buffer`
3: **for all** $k \in \{1, \ldots, m_\lambda\}$, $\ell \in \{1, \ldots, n_\lambda\}$ **do**
4: $S_f(k, \ell) \leftarrow b_f$ {*initialization of* $m_\lambda \times n_\lambda$ *weighted sums*}
5: **end for**
6: **for all** input feature maps $g \in \mathcal{N}_{\lambda-1}$ **do**
7: **for all** $i_0, j_0 \in \{1, \ldots, \sigma_\lambda\}$ **do**
8: {*i.e. for all grid submaps* $g(i_0, j_0)$ *from the partition of* g}
9: **for all** $(k, \ell) \in g(i_0, j_0)$ **do**
10: **read** $y_g(k, \ell)$ into `Buffer` {*reading* $m_\lambda \times n_\lambda$ *submap inputs*}
11: **end for**
12: **for all** (k_1, ℓ_1) s.t. $1 \le i_0 + k_1\sigma_\lambda \le r_\lambda$, $1 \le j_0 + \ell_1\sigma_\lambda \le s_\lambda$ **do**
13: {*i.e. for all weights* $w_{fg}(i_0 + k_1\sigma_\lambda\,, j_0 + \ell_1\sigma_\lambda)$ *shared by submap* $g(i_0, j_0)$}
14: $i \leftarrow i_0 + k_1\sigma_\lambda$; $j \leftarrow j_0 + \ell_1\sigma_\lambda$
15: **read** weight $w_{fg}(i, j)$ into `Buffer`
16: **for all** $k \in \{1, \ldots, m_\lambda\}$, $\ell \in \{1, \ldots, n_\lambda\}$ **do**
17: {*i.e. for all MACs in (2) over the weight* $w_{fg}(i, j)$}
18: $S_f(k, \ell) \leftarrow S_f(k, \ell) + w_{fg}(i, j) \cdot y_g\big((k-1)\sigma_\lambda + i\,, (\ell-1)\sigma_\lambda + j\big)$
19: **end for**
20: **end for**{*next shared weight*}
21: **end for**{*next input submap*}
22: **end for**{*next input feature map*}
23: **for all** $k \in \{1, \ldots, m_\lambda\}$, $\ell \in \{1, \ldots, n_\lambda\}$ **do**
24: **write** $y_f(k, \ell) \leftarrow \mathrm{ReLU}\big(S_f(k, \ell)\big)$ to `DRAM` {*writing* $m_\lambda \times n_\lambda$ *outputs*}
25: **end for**
26: **end for**{*next feature map*}

g, respectively. The $m_\lambda \times n_\lambda$ inputs $y_g(k, \ell)$ of submap $g(i_0, j_0)$ for all $(k, \ell) \in g(i_0, j_0)$ are read from DRAM into Buffer (lines 9–11).

The next nested loop 12–20 goes through all the weights $w_{fg}(i, j)$ that are shared by the submap $g(i_0, j_0)$ for all integers $k_1, \ell_1 \in \mathbb{Z}$ satisfying (16). The single weight $w_{fg}(i, j)$ is read from DRAM into Buffer (line 15) which is then used in the innermost loop 16–19 to carry out all the MAC operations in Buffer over this weight, updating all the weighted sums $S_f(k, \ell)$ of feature map f, for all $k \in \{1, \ldots, m_\lambda\}$ and $\ell \in \{1, \ldots, n_\lambda\}$ (line 18). After these weighted sums $S_f(k, \ell)$ are eventually evaluated for feature map f, the corresponding $m_\lambda \times n_\lambda$ outputs $y_f(k, \ell)$ are computed according to (3) and written to DRAM, for all $k \in \{1, \ldots, m_\lambda\}$ and $\ell \in \{1, \ldots, n_\lambda\}$ (lines 23–25).

Algorithm 1 uses $B = b(2m_\lambda n_\lambda + 1)$ bits of Buffer memory satisfying (13), which is occupied by $m_\lambda n_\lambda$ accumulated weighted sums $S_f(k, \ell)$ for all $k \in \{1, \ldots, m_\lambda\}$ and $\ell \in \{1, \ldots, n_\lambda\}$ (lines 3–5, 16–19, and 23–25), $m_\lambda n_\lambda$ inputs $y_g(k, \ell)$ of grid submap $g(i_0, j_0)$ for all $(k, \ell) \in g(i_0, j_0)$ (lines 9–11), and one weight $w_{fg}(i, j)$ shared by submap $g(i_0, j_0)$ (line 15).

Each output $y_f(k, \ell)$ for $f \in \mathcal{N}_\lambda$ ($|\mathcal{N}_\lambda| = d_\lambda$), $k \in \{1, \ldots, m_\lambda\}$, and $\ell \in \{1, \ldots, n_\lambda\}$ is written to DRAM only once (line 24 within the nested loops 1–26 and 23–25), which

gives

$$E^{\lambda}_{\text{outputs}} = b\, d_\lambda m_\lambda n_\lambda \,. \tag{17}$$

The partition (14) ensures that each weight $w_{fg}(i,j)$ for $f \in \mathcal{N}_\lambda$, $g \in \mathcal{N}_{\lambda-1}$, $i_0, j_0 \in \{1,\ldots,\sigma_\lambda\}$, and $k_1, \ell_1 \in \mathbb{Z}$ satisfying (16) (line 15 within the nested loops 1–26, 6–22, 7–21, and 12–20) including bias b_f for $f \in \mathcal{N}_\lambda$ (line 2 within the outermost loop 1–26) is read into Buffer only once, which implies (12).

Every input $y_g(k,\ell)$ for $g \in \mathcal{N}_{\lambda-1}$ and $(k,\ell) \in \bigcup_{i_0,j_0 \in \{1,\ldots,\sigma_\lambda\}} g(i_0,j_0)$ is read once for each macro-unit $f \in \mathcal{N}_\lambda$ (line 10 within the nested loops 1–26, 6–22, 7–21, and 9–11) due the partition (14), which implies

$$E^{\lambda}_{\text{inputs}} = b\, d_\lambda d_{\lambda-1} m_{\lambda-1} n_{\lambda-1} \,. \tag{18}$$

Hence, the dataflow provides the following upper bound on the data energy of layer $\mathcal{N}_\lambda$:

$$E^{\lambda}_{\text{data}} \le b\, d_\lambda \left(d_{\lambda-1} m_{\lambda-1} n_{\lambda-1} + m_\lambda n_\lambda + d_{\lambda-1} r_\lambda s_\lambda + 1\right) \tag{19}$$

according to (7), (18), (17), and (12).

In addition, we also present an alternative dataflow with write-once outputs of the same data energy (19) in Algorithm 2, whose Buffer capacity is

$$B = b\left(m_\lambda n_\lambda + r_\lambda s_\lambda + 1\right) \,, \tag{20}$$

cf. (13). The alternative dataflow differs in that we load the whole filter $\mathbf{W}_{fg}$ of $r_\lambda \times s_\lambda$ weights from DRAM into Buffer at once for each $f \in \mathcal{N}_\lambda$ and $g \in \mathcal{N}_{\lambda-1}$ (lines 7–9 within the nested loops 1–21 and 6–17), while only the single inputs $y_g(k_1,\ell_1)$ from the input feature map $g \in \mathcal{N}_{\lambda-1}$ are read from DRAM into Buffer one by one for each $k_1 \in \{1,\ldots,m_{\lambda-1}\}$, $\ell_1 \in \{1,\ldots,n_{\lambda-1}\}$ (line 11 within the loop 10–16) which is then used in the innermost loop 12–15 to carry out all the MAC operations in Buffer over this input, updating all the weighted sums $S_f(k,\ell)$ of feature map f, for all $k \in \{1,\ldots,m_\lambda\}$ and $\ell \in \{1,\ldots,n_\lambda\}$ (line 14).

It follows that the Buffer memory is used for $m_\lambda n_\lambda$ accumulated weighted sums $S_f(k,\ell)$ for all $k \in \{1,\ldots,m_\lambda\}$ and $\ell \in \{1,\ldots,n_\lambda\}$ (lines 3–5, 12–15, and 18–20), $r_\lambda s_\lambda$ weights $w_{fg}(i,j)$ for all $i \in \{1,\ldots,r_\lambda\}$, $j \in \{1,\ldots,s_\lambda\}$ (lines 7–9), and one input $y_g(k_1,\ell_1)$ (line 11), which implies (20). Moreover, every input in layer $\mathcal{N}_{\lambda-1}$ is read into Buffer once for each macro-unit $f \in \mathcal{N}_\lambda$ (line 11 within the outermost loop 1–21), which proves the upper bound (19) also for this alternative dataflow.

5.2 The Dataflow with Read-Once Inputs

The dataflow in which each input is read into Buffer only once is described in Algorithm 3. This algorithm is basically composed of five nested *for* loops 2–36, 3–35, 9–34, 20–33, and 24–32. The two outermost loops 2–36 and 3–35 go through all the macro-units $g \in \mathcal{N}_{\lambda-1}$ representing the *input* feature maps in the $(\lambda-1)$th layer, and through all the grid submaps $g(i_0,j_0)$ parameterized by $i_0, j_0 \in \{1,\ldots,\sigma_\lambda\}$ from the partition (14) of g, respectively. The variable inp is used to count the total number of currently processed input submaps in layer $\mathcal{N}_{\lambda-1}$ (lines 1 and 5). The $m_\lambda \times n_\lambda$ inputs

Algorithm 2 The dataflow with write-once outputs and $B = b(m_\lambda n_\lambda + r_\lambda s_\lambda + 1)$.

1: **for all** feature maps $f \in \mathcal{N}_\lambda$ **do**
2: **read** bias b_f into `Buffer`
3: **for all** $k \in \{1, \ldots, m_\lambda\}$, $\ell \in \{1, \ldots, n_\lambda\}$ **do**
4: $S_f(k, \ell) \leftarrow b_f$ {*initialization of* $m_\lambda \times n_\lambda$ *weighted sums*}
5: **end for**
6: **for all** input feature maps $g \in \mathcal{N}_{\lambda-1}$ **do**
7: **for all** $i \in \{1, \ldots, r_\lambda\}$, $j \in \{1, \ldots, s_\lambda\}$ **do**
8: **read** weight $w_{fg}(i, j)$ into `Buffer` {*reading filter* $\mathbf{W}_{fg}$}
9: **end for**
10: **for all** $k_1 \in \{1, \ldots, m_{\lambda-1}\}$, $\ell_1 \in \{1, \ldots, n_{\lambda-1}\}$ **do**
11: **read** $y_g(k_1, \ell_1)$ into `Buffer`
12: **for all** $k \in \{1, \ldots, m_\lambda\}$, $\ell \in \{1, \ldots, n_\lambda\}$, $i \in \{1, \ldots, r_\lambda\}$, $j \in \{1, \ldots, s_\lambda\}$ s.t. $k_1 = (k-1)\sigma_\lambda + i$, $\ell_1 = (\ell - 1)\sigma_\lambda + j$ **do**
13: {*i.e. for all MACs in (2) over the input* $y_g(k_1, \ell_1)$}
14: $S_f(k, \ell) \leftarrow S_f(k, \ell) + w_{fg}(i, j) \cdot y_g\big((k-1)\sigma_\lambda + i\,, (\ell - 1)\sigma_\lambda + j\big)$
15: **end for**
16: **end for**{*next input*}
17: **end for**{*next input feature map*}
18: **for all** $k \in \{1, \ldots, m_\lambda\}$, $\ell \in \{1, \ldots, n_\lambda\}$ **do**
19: **write** $y_f(k, \ell) \leftarrow \mathrm{ReLU}\big(S_f(k, \ell)\big)$ to `DRAM` {*writing* $m_\lambda \times n_\lambda$ *outputs*}
20: **end for**
21: **end for**{*next feature map*}

$y_g(k, \ell)$ of submap $g(i_0, j_0)$ for all $(k, \ell) \in g(i_0, j_0)$ are read from DRAM into Buffer (lines 6–8).

The next nested loop 9–34 passes through all the macro-units $f \in \mathcal{N}_\lambda$ representing the feature maps in the $(\lambda - 1)$th convolutional layer for $\lambda \in \Gamma$. For the first input submap (line 10), the bias b_f is read (line 11) in order to initialize the $m_\lambda \times n_\lambda$ weighted sums $S_f(k, \ell)$ of feature map f accumulated in Buffer, for all $k \in \{1, \ldots, m_\lambda\}$ and $\ell \in \{1, \ldots, n_\lambda\}$ (lines 12–14), while for the next input submaps (line 15) these weighted sums currently stored in DRAM (line 30) are read into Buffer (lines 16–18).

The following nested loop 20–33 goes through all the weights $w_{fg}(i, j)$ that are shared by the submap $g(i_0, j_0)$ for all integers $k_1, \ell_1 \in \mathbb{Z}$ satisfying (16). The single weight $w_{fg}(i, j)$ is read from DRAM into Buffer (line 23) which is then used in the innermost loop 24–32 to carry out all the MAC operations in Buffer over this weight, updating all the weighted sums $S_f(k, \ell)$ of feature map f, for all $k \in \{1, \ldots, m_\lambda\}$ and $\ell \in \{1, \ldots, n_\lambda\}$ (line 26). For the last input submap (line 27) when these weighted sums $S_f(k, \ell)$ have eventually been evaluated for feature map f, the corresponding $m_\lambda \times n_\lambda$ outputs $y_f(k, \ell)$ are computed according to (3) and written to DRAM, for all $k \in \{1, \ldots, m_\lambda\}$ and $\ell \in \{1, \ldots, n_\lambda\}$ (line 28 within the loop 24–32). Otherwise (line 29), these weighted sums $S_f(k, \ell)$ of feature map f are stored in DRAM for all $k \in \{1, \ldots, m_\lambda\}$ and $\ell \in \{1, \ldots, n_\lambda\}$ (line 30 within the loop 24–32).

Algorithm 3 uses $B = b(2m_\lambda n_\lambda + 1)$ bits of Buffer memory satisfying (13), which is occupied by $m_\lambda n_\lambda$ inputs $y_g(k, \ell)$ of grid submap $g(i_0, j_0)$ for all $(k, \ell) \in g(i_0, j_0)$

Algorithm 3 The dataflow with read-once inputs and $B = b(2m_\lambda n_\lambda + 1)$.

$inp \leftarrow 0$
for all input feature maps $g \in \mathcal{N}_{\lambda-1}$ **do**
  for all $i_0, j_0 \in \{1, \dots, \sigma_\lambda\}$ **do**
    {*i.e. for all grid submaps $g(i_0, j_0)$ from the partition of g*}
    $inp \leftarrow inp + 1$ {*counts the current number of input submaps*}
    for all $(k, \ell) \in g(i_0, j_0)$ **do**
      read $y_g(k, \ell)$ into `Buffer` {*reading $m_\lambda \times n_\lambda$ submap inputs*}
    end for
    for all feature maps $f \in \mathcal{N}_\lambda$ **do**
      if $inp = 1$ {*first input submap*} **then**
        read bias b_f into `Buffer`
        for all $k \in \{1, \dots, m_\lambda\}$, $\ell \in \{1, \dots, n_\lambda\}$ **do**
          $S_f(k, \ell) \leftarrow b_f$ {*initialization of $m_\lambda \times n_\lambda$ weighted sums*}
        end for
      else
        for all $k \in \{1, \dots, m_\lambda\}$, $\ell \in \{1, \dots, n_\lambda\}$ **do**
          read $S_f(k, \ell)$ into `Buffer` {*reading $m_\lambda \times n_\lambda$ weighted sums*}
        end for
      end if
      for all (k_1, ℓ_1) s.t. $1 \le i_0 + k_1\sigma_\lambda \le r_\lambda$, $1 \le j_0 + \ell_1\sigma_\lambda \le s_\lambda$ **do**
        {*i.e. for all weights $w_{fg}(i_0 + k_1\sigma_\lambda\,, j_0 + \ell_1\sigma_\lambda)$ shared by submap $g(i_0, j_0)$*}
        $i \leftarrow i_0 + k_1\sigma_\lambda$; $j \leftarrow j_0 + \ell_1\sigma_\lambda$
        read weight $w_{fg}(i, j)$ into `Buffer`
        for all $k \in \{1, \dots, m_\lambda\}$, $\ell \in \{1, \dots, n_\lambda\}$ **do**
          {*i.e. for all MACs in (2) over the weight $w_{fg}(i, j)$*}
          $S_f(k, \ell) \leftarrow S_f(k, \ell) + w_{fg}(i, j) \cdot y_g\big((k-1)\sigma_\lambda + i\,, (\ell-1)\sigma_\lambda + j\big)$
          if $inp = d_{\lambda-1}\sigma_\lambda^2$ {*last input submap*} **then**
            write $y_f(k, \ell) \leftarrow \mathrm{ReLU}\big(S_f(k, \ell)\big)$ to `DRAM` {*writing $m_\lambda \times n_\lambda$ outputs*}
          else
            write $S_f(k, \ell)$ to `DRAM` {*writing $m_\lambda \times n_\lambda$ weighted sums*}
          end if
        end for
      end for{*next shared weight*}
    end for{*next feature map*}
  end for{*next input submap*}
end for{*next input feature map*}

(lines 6–8), $m_\lambda n_\lambda$ accumulated weighted sums $S_f(k,\ell)$ for all $k \in \{1,\dots,m_\lambda\}$ and $\ell \in \{1,\dots,n_\lambda\}$ (lines 12–14, 16–18, and 24–32), and one weight $w_{fg}(i,j)$ shared by submap $g(i_0,j_0)$ (line 23).

Moreover, the partition (14) ensures that each input $y_g(k,\ell)$ for $g \in \mathcal{N}_{\lambda-1}$ and $(k,\ell) \in \bigcup_{i_0,j_0\in\{1,\dots,\sigma_\lambda\}} g(i_0,j_0)$ (line 7 within the nested loops 2–36, 3–35, and 6–8), as well as each weight $w_{fg}(i,j)$ for $g \in \mathcal{N}_{\lambda-1}$, $f \in \mathcal{N}_\lambda$, $i_0,j_0 \in \{1,\dots,\sigma_\lambda\}$, and $k_1,\ell_1 \in \mathbb{Z}$ satisfying (16) (line 23 within the nested loops 2–36, 3–35, 9–34, and 20–33) including bias b_f for $f \in \mathcal{N}_\lambda$ (line 11 within the first run of loop 9–34) is read into Buffer only once, which implies

$$E^\lambda_{\text{inputs}} = b\, d_{\lambda-1} m_{\lambda-1} n_{\lambda-1} \tag{21}$$

and (12), respectively.

Any accumulated weighted sum $S_f(k,\ell)$ for $f \in \mathcal{N}_\lambda$, $k \in \{1,\dots,m_\lambda\}$, and $\ell \in \{1,\dots,n_\lambda\}$, is read (lines 16–18 within the loop 9–34) except for its initialization by bias b_f at the beginning (lines 11–14), and written once for each of the $d_{\lambda-1}$ macro-units $g \in \mathcal{N}_{\lambda-1}$ and each of the σ_λ^2 submaps $g(i_0,j_0)$ from the partition (14) of g (lines 28 and 30 within the nested loops 2–36 and 3–35), which gives

$$E^\lambda_{\text{outputs}} = b\left(2d_{\lambda-1}\sigma_\lambda^2 - 1\right) d_\lambda m_\lambda n_\lambda\,. \tag{22}$$

Hence, this dataflow provides another upper bound on the data energy of layer $\mathcal{N}_\lambda$:

$$E^\lambda_{\text{data}} \le b\left(d_{\lambda-1}m_{\lambda-1}n_{\lambda-1} + \left(2d_{\lambda-1}\sigma_\lambda^2 - 1\right) d_\lambda m_\lambda n_\lambda + d_\lambda\left(d_{\lambda-1}r_\lambda s_\lambda + 1\right)\right) \tag{23}$$

according to (7), (21), (22), and (12).

5.3 Comparison of Energy Complexity Bounds

The upper bound (19) on the data energy E^λ_{data} for evaluating a convolutional layer $\mathcal{N}_\lambda$ ($\lambda \in \Gamma$), achieved by Algorithms 1 and 2 that are based on the dataflow with write-once outputs, with Buffer capacity (13) and (20), respectively, differs only in the number of DRAM accesses for reading inputs by factor d_λ from the trivial lower bound (8):

$$\begin{aligned} b\left(d_{\lambda-1}m_{\lambda-1}n_{\lambda-1} + d_\lambda m_\lambda n_\lambda + d_\lambda(d_{\lambda-1}r_\lambda s_\lambda + 1)\right) &\le E^\lambda_{\text{data}} && (24)\\ \le\ b\left(\boldsymbol{d_\lambda}\, d_{\lambda-1}m_{\lambda-1}n_{\lambda-1} + d_\lambda m_\lambda n_\lambda + d_\lambda(d_{\lambda-1}r_\lambda s_\lambda + 1)\right)&\,. && (25)\end{aligned}$$

The used non-constant Buffer size $B = b(2m_\lambda n_\lambda + 1)$ according to (13) (similarly for (20)) devalues the improved lower bound (9) to $E^\lambda_{\text{data}} \ge b \cdot d_\lambda d_{\lambda-1} r_\lambda s_\lambda$ which is weaker than the trivial lower bound (24). It is an open problem whether the gap between the lower bound (24) and the upper bound (25) on the data energy E^λ_{data} can be removed. We conjecture that the trivial lower bound bound (24) can possibly be improved to match the upper bound (25), which would provide the optimal energy complexity for the assumed Buffer capacity $B = b(2m_\lambda n_\lambda + 1)$.

Furthermore, suppose that the number of single neurons in layers $\mathcal{N}_{\lambda-1}$ and $\mathcal{N}_\lambda$ is of the same order, that is, $d_{\lambda-1}m_{\lambda-1}n_{\lambda-1} \approx d_\lambda m_\lambda n_\lambda$, as typically assumed in practical

CNNs. Then the upper bound (23) on the data energy $E^{\lambda}_{\text{data}}$ for evaluating a convolutional layer $\mathcal{N}_\lambda$ ($\lambda \in \Gamma$), achieved by Algorithm 3 that is based on the dataflow with write-once inputs, using the Buffer capacity (13), can be rewritten as

$$E^{\lambda}_{\text{data}} = O\left(d_\lambda m_\lambda n_\lambda + d_\lambda d_{\lambda-1} m_{\lambda-1} n_{\lambda-1} + d_\lambda \left(d_{\lambda-1} r_\lambda s_\lambda + 1\right)\right) \tag{26}$$

since $\sigma^2_\lambda m_\lambda n_\lambda \approx m_{\lambda-1} n_{\lambda-1}$ according to (1). The upper bound (26) coincides asymptotically with (25), which means that the three proposed Algorithm 1–3 are comparable from the data energy complexity point of view. Nevertheless, the multiplicative constant 2 of leading term $d_\lambda d_{\lambda-1} m_{\lambda-1} n_{\lambda-1}$ in (23), which is caused by storing partially evaluated weighted sums in DRAM, makes the upper bound (25) more tight than (23).

6 Experimental Validation

In this section, we compare the theoretical energy complexity introduced in Section 3 to the real energy consumption estimated by the Timeloop/Accelergy software tool for evaluating DNN accelerator designs. The Timeloop (Parashar et al., 2019) finds a mapping of a convolutional layer specified by its architectural parameters (e.g. high, width, depth, size of receptive fields, stride) onto a given hardware platform, which is empirically optimal in terms of energy consumption estimated by Accelergy (Wu et al., 2019) reporting the energy statistics.

Namely, we have employed Simba (Shao et al., 2019) and Eyeriss (Chen et al., 2016) as the target hardware platforms onto which convolutional layers with increasing parameters have been mapped so that the least energy intensive dataflows are achieved. These platforms have been chosen as prominent examples of accelerators based on the systolic array of processing elements which are often implemented in practice as they are general and not tied to a specific CNN. All configuration files used in experiments are publicly available at Github[1].

The *computation energy* reported by Timeloop/Accelergy corresponds directly to the number of MACs calculated in (6) for $E^{\lambda}_{\text{comp}}$ where C_b is the energy per one MAC operation which was estimated as $C_8 = 0.56\,pJ$ and $C_{16} = 2.20\,pJ$ for 8-bit Simba and 16-bit Eyeriss architectures, respectively.

For a convolutional layer $\mathcal{N}_\lambda$ where $\lambda \in \Gamma$, we measure empirical dependencies of the optimal *data energy* first separately on its depth d_λ, input feature map size $m_{\lambda-1} = n_{\lambda-1}$, kernel size $r_\lambda = s_\lambda$, and stride σ_λ, which is minimized by using the Timeloop/Accelergy framework exploring various types of possible dataflows for the Simba and Eyeriss architectures. These dependencies are then compared to the corresponding upper bound (19) on $E^{\lambda}_{\text{data}}$ achieved by the proposed dataflows in the energy complexity model as presented in Section 5.3. This theoretical upper bound assumes a sufficient Buffer capacity satisfying (13) or (20). Table 1 shows required Buffer capacities for the AlexNet convolutional layers (Krizhevsky, Sutskever, & Hinton, 2017) in kilobytes (kB) which appear in an order of magnitude to be realistic to common hardware architectures such as Eyeriss (Chen et al., 2016).

[1] `https://github.com/PetraVidnerova/timeloop-accelergy-test`

Table 1: Required buffer capacities for AlexNet convolutional layers in kilobytes.

λ	1	2	3	4	5
$m_\lambda = n_\lambda$	55	27	13	13	13
d_λ	64	192	384	256	256
$r_\lambda = s_\lambda$	11	5	3	3	3
σ_λ	4	1	1	1	1
(13): $2m_\lambda^2 + 1$	6051	1459	339	339	339
$b = 8$ bits	5.91 kB	1.42 kB	0.33 kB	0.33 kB	0.33 kB
$b = 16$ bits	11.82 kB	2.85 kB	0.66 kB	0.66 kB	0.66 kB
$b = 32$ bits	23.64 kB	5.7 kB	1.32 kB	1.32 kB	1.32 kB
(20): $m_\lambda^2 + r_\lambda^2 + 1$	3147	755	179	179	179
$b = 8$ bits	3.07 kB	0.74 kB	0.17 kB	0.17 kB	0.17 kB
$b = 16$ bits	6.15 kB	1.47 kB	0.35 kB	0.35 kB	0.35 kB
$b = 32$ bits	12.29 kB	2.95 kB	0.7 kB	0.7 kB	0.7 kB

In particular, for the comparison of empirical energy consumption to the theoretical data energy E^λ_{data}, we use the following asymptotic upper bounds:

$$E^\lambda_{\text{data}} = O\left(d_\lambda\right),\ E^\lambda_{\text{data}} = O\left(m^2_{\lambda-1}\right),\ E^\lambda_{\text{data}} = O\left(r^2_\lambda\right),\ E^\lambda_{\text{data}} = O\left(\sigma^{-2}_\lambda\right), \tag{27}$$

which are derived from (19) for individual variables (when the other independent parameters are considered to be constant) by using the approximation $m^2_{\lambda-1} \approx \sigma^2_\lambda m^2_\lambda$ due to (1). Note that the upper bounds in (27) match asymptotically the corresponding lower bounds derived from (11) for individual variables and constant Buffer capacity B.

Figure 4 presents the results of experimental comparison of energy-efficient CNN hardware implementations to our theoretical energy complexity model separately for individual parameters of a convolutional layer. By using the Timeloop/Accelergy tool applied to the Simba and Eyeriss hardware architectures, the optimal values of their data energy consumption have been estimated for a convolutional layer $\mathcal{N}_\lambda$ where $\lambda \in \Gamma$, with increasing parameters d_λ, $m_{\lambda-1} = n_{\lambda-1}$, $r_\lambda = s_\lambda$, and σ_λ, each separately. Namely, the values for d_λ (or $d_{\lambda-1}$ for which the results omitted in Figure 4 were similar) and $m_{\lambda-1}$ were taken from the interval 8 to 512 and 56 to 448, respectively, with the step 8, while for r_λ and σ_λ we took the values 3, 5, 7, 9, 11 and 1, 2, 3, 4, 5, respectively.

These parameters serve as independent variables in regression analysis where the relationships between the data energy and the independent variables are modeled as functions with asymptotics (27), including multiplicative and additive coefficients c_2 and c_1, respectively. As depicted in Figure 4, these coefficients are approximated by the method of least squares so that the theoretical data energy E^λ_{data} (dashed lines) fits energy estimates by Timeloop/Accelergy (displayed by bars), which confirms the asymptotic trends (27) in the energy complexity model. However, note that the calculated values of coefficients c_2 and c_1 do not have any special meaning in the Simba and Eyeriss hardware architectures that are not related to the proposed hardware-independent energy complexity model.

In addition, the energy complexity model has been validated by statistical tests using quadratic regression with the function model $ax^2 + bx + c$ for independent variable x to

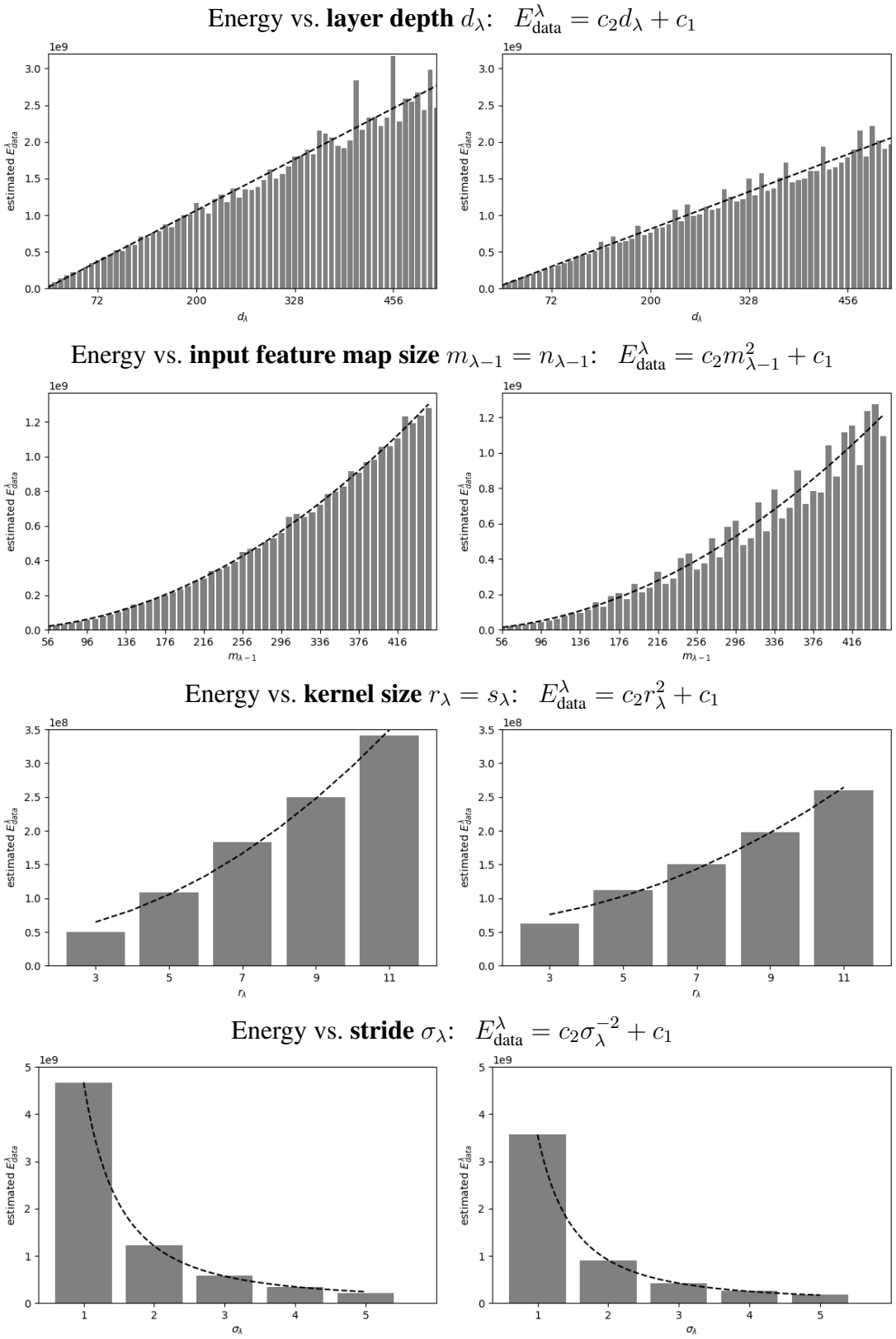


Figure 4: The data energy estimates by Timeloop/Accelergy (displayed by bars) for convolutional layer $\mathcal{N}_\lambda$ with increasing parameters d_λ, $m_{\lambda-1}$,r_λ, and σ_λ, each separately (from top to bottom), on the Simba (left) and Eyeriss (right) architectures, which fit the asymptotic trends (27) in the energy complexity model (dashed lines).

be d_λ, $m_{\lambda-1}$, r_λ, and σ_λ^{-1}, respectively. These statistical tests have approved the linearity in d_λ (p-value 0.556 accepting the null hypothesis of $a = 0$) and the quadraticity in $m_{\lambda-1}$ r_λ, and σ_λ^{-1} (p-value 0.000, 0.001, and 0.000, respectively, rejecting the null hypothesis of $a = 0$).

Furthermore, we have also extended our experiments to the case where all the parameters d_λ, $d_{\lambda-1}$, $m_{\lambda-1} = n_{\lambda-1}$, $r_\lambda = s_\lambda$, and σ_λ of convolutional layer $\mathcal{N}_\lambda$ where $\lambda \in \Gamma$, are changed simultaneously, in order to fit the empirical optimal data energy to its theoretical asymptotic lower bound (11):

$$E^\lambda_{\text{data}} = \Omega\left(d_\lambda d_{\lambda-1} m^2_{\lambda-1} r^2_\lambda \sigma^{-2}_\lambda\right) \tag{28}$$

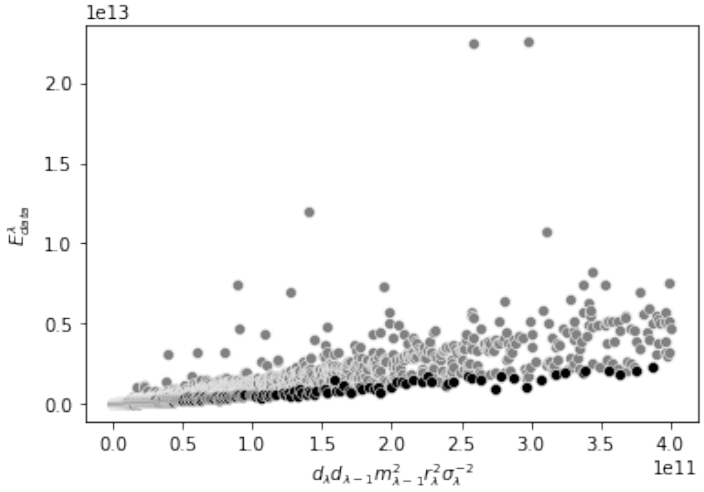
for constant Buffer capacity B. For this purpose, we employ 4840 convolutional layers whose parameters d_λ, $d_{\lambda-1}$, $m_{\lambda-1}$, r_λ, and σ_λ are sampled uniformly at random from the intervals 8 to 512, 8 to 512, 56 to 448, 3 to 15, and, 1 to 5, respectively, since there are too many combinations of these parameters' values to run the time consuming Timeloop/Accelergy tool on each. Anyway, different samplings in our repeated experiments led to the same results. The points in Figure 5.a show energy consumption of these convolutional layers on the Simba architecture estimated by the Timeloop/Accelergy program, depending on the product $d_\lambda d_{\lambda-1} m^2_{\lambda-1} r^2_\lambda \sigma^{-2}_\lambda$ from (28).

It appears that the energy estimates in Figure 5.a which represent empirical upper bounds on the data energy, fluctuate for some combinations of convolutional layer parameters even for the same value of product $d_\lambda d_{\lambda-1} m^2_{\lambda-1} r^2_\lambda \sigma^{-2}_\lambda$. This is due to specific properties of the used Simba hardware architecture (e.g. energy consumption can be significantly increased by a systematic step widening of the systolic array required for certain critical combinations of increasing convolutional layer parameters). In order to asymptotically compare the empirical optimal data energy to its theoretical lower bound (28), we filter out these fluctuations by selecting the points with the minimum energy over each 10 consecutive distinct products $d_\lambda d_{\lambda-1} m^2_{\lambda-1} r^2_\lambda \sigma^{-2}_\lambda$ which are indicated in boldface in Figure 5.a. The constant 10 was chosen as a compromise between a sufficient number of sampled points and outliers, yet the results for values of 10 to 50 are very similar.

The selected points are then used in regression analysis where the relationship between the data energy and the product $d_\lambda d_{\lambda-1} m^2_{\lambda-1} r^2_\lambda \sigma^{-2}_\lambda$ as the independent variable is modeled as a function with asymptotics (28), including multiplicative and additive coefficients c_2 and c_1, respectively. As depicted in Figure 5.b, these coefficients are approximated by the method of least squares so that the theoretical data energy E^λ_{data} (dashed line) fits energy estimates (displayed by points) by the Timeloop/Accelergy program for the Simba architecture. This has been validated by the statistical test using quadratic regression with the function model $ax^2 + bx + c$ for independent variable $x = d_\lambda d_{\lambda-1} m^2_{\lambda-1} r^2_\lambda \sigma^{-2}_\lambda$, which has approved the linearity with the p-value 0.3084 accepting the null hypothesis of $a = 0$.

The presented experiments have thus validated the energy complexity model whose upper and lower bounds on theoretical energy for a convolutional layer fit asymptotically very well the energy consumption estimated by the Timeloop/Accelergy program for the Simba and Eyeriss hardware platforms. Of course, this perfect asymptotic match does no mean that for some critical values of parameters, the energy consumption of practical CNN accelerators cannot deviate substantially from the theoretical bounds, as

(a) Energy vs. $d_\lambda d_{\lambda-1} m_{\lambda-1}^2 r_\lambda^2 \sigma_\lambda^{-2}$



(b) $E_{\text{data}}^\lambda = c_2 \cdot d_\lambda d_{\lambda-1} m_{\lambda-1}^2 r_\lambda^2 \sigma_\lambda^{-2} + c_1$

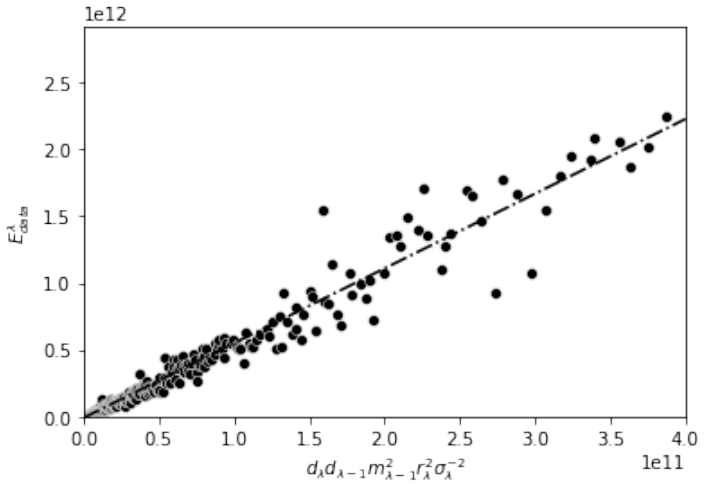


Figure 5: The data energy estimates by Timeloop/Accelergy (displayed by points) depending on $d_\lambda d_{\lambda-1} m_{\lambda-1}^2 r_\lambda^2 \sigma_\lambda^{-2}$ for convolutional layer $\mathcal{N}_\lambda$ on the Simba architecture (a), whose fluctuations are filtered out by selecting the energy minima over each 10 consecutive values of independent variable (points in boldface), which fit the asymptotic lower bound (28) on E_{data}^λ (dashed line) in the energy complexity model (b).

illustrated by fluctuations in Figure 5.a on the Simba architecture. Nevertheless, constants are hidden in the big O notation within the theoretical fairly tight *asymptotic* dependencies.

7 Conclusion

In this paper, we have introduced a hardware-independent energy complexity model for CNNs that captures asymptotically all important sources of energy consumption of their diverse hardware implementations. In this model, we have proven a simple lower bound on energy complexity which establishes asymptotic limits on energy efficiency of any CNN hardware accelerators. Moreover, we have derived corresponding upper bounds for two common energy-efficient dataflows with write-once outputs and read-once inputs, respectively. The underlying theoretical asymptotic trends in energy complexity both for individual CNN parameters and their combination, have been validated by statistical tests to fit the energy consumption optimized empirically by the Timeloop/Accelergy program for CNNs on the Simba and Eyeriss hardware platforms for various dataflows.

In future research we plan to prove optimal bounds on the data energy of CNNs in the model with non-constant Buffer capacity B such as (13). Partial results along this direction have already been achieved for a special case of fully-connected layers (Šíma & Cabessa, 2023). The proposed model thus allows to determine the principal asymptotic limits to which heuristic optimizers e.g. based on evolutionary algorithms (Kao & Krishna, 2020) can reach.

Another important challenge is to generalize the proposed methodology to transformers that represent now the dominating workloads for many applications (Zhou, Liu, Gu, & Sun, 2023). This includes the derivation of new energy bounds for transformers whose architecture differs from that of CNNs, for which our previous analysis for fully-connected layers can possibly be exploited (Šíma & Cabessa, 2023). In addition, a method of estimating real energy consumption for transformers should also be developed.

Acknowledgements

The presentation of this paper benefited from valuable suggestions of an anonymous reviewer. This work was supported by the Czech Science Foundation grant GA22-02067S and the institutional support RVO: 67985807 (J. Šíma, P. Vidnerová). We thank Jan Kalina for expert consultation regarding statistical tests.

References

Alwani, M., Chen, H., Ferdman, M., & Milder, P. A. (2016). Fused-layer CNN accelerators. In *Proceedings of the 49th annual IEEE/ACM international symposium on microarchitecture (MICRO 2016)* (pp. 22:1–22:12). doi: 10.1109/MICRO.2016.7783725

Ansari, M. S., Mrazek, V., Cockburn, B. F., Sekanina, L., Vasicek, Z., & Han, J. (2020). Improving the accuracy and hardware efficiency of neural networks using approximate multipliers. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, *28*(2), 317–328. doi: 10.1109/TVLSI.2019.2940943

Armeniakos, G., Zervakis, G., Soudris, D., & Henkel, J. (2023). Hardware approximate techniques for deep neural network accelerators: A survey. *ACM Computing Surveys*, *55*(4), 83:1–83:36. doi: 10.1145/3527156

Chen, Y., Emer, J. S., & Sze, V. (2016). Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks. In *Proceedings of the 43rd annual ACM/IEEE international symposium on computer architecture (ISCA 2016)* (pp. 367–379). doi: 10.1109/ISCA.2016.40

Gupta, S., Agrawal, A., Gopalakrishnan, K., & Narayanan, P. (2015). Deep learning with limited numerical precision. In F. Bach & D. Blei (Eds.), *Proceedings of the 32nd international conference on machine learning (ICML 2015), JMLR workshop and conference proceedings* (Vol. 37, pp. 1737–1746). Retrieved from `http://proceedings.mlr.press/v37/gupta15.html`

Kao, S.-C., & Krishna, T. (2020). GAMMA: Automating the HW mapping of DNN models on accelerators via genetic algorithm. In *Proceedings of the 39th international conference on computer-aided design (ICCAD 2020)* (pp. 44:1–44:9). doi: 10.1145/3400302.3415639

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2017). Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, *60*(6), 84–90. doi: 10.1145/3065386

Mittal, S. (2016). A survey of techniques for approximate computing. *ACM Computing Surveys*, *48*(4), 62:1–62:33. doi: 10.1145/2893356

Mittal, S. (2020). A survey of FPGA-based accelerators for convolutional neural networks. *Neural Computing and Applications*, *32*(4), 1109–1139. doi: 10.1007/s00521-018-3761-1

Parashar, A., Raina, P., Shao, Y. S., Chen, Y., Ying, V. A., Mukkara, A., ... Emer, J. S. (2019). Timeloop: A systematic approach to DNN accelerator evaluation. In *Proceedings of the IEEE international symposium on performance analysis of systems and software (ISPASS 2019)* (pp. 304–315). doi: 10.1109/ISPASS.2019.00042

Shao, Y. S., Clemons, J., Venkatesan, R., Zimmer, B., Fojtik, M., Jiang, N., ... Keckler, S. W. (2019). Simba: Scaling deep-learning inference with multi-chip-module-based architecture. In *Proceedings of the 52nd annual IEEE/ACM international symposium on microarchitecture (MICRO 2019)* (pp. 14–27). doi: 10.1145/3352460.3358302

Šíma, J. (2014). Energy complexity of recurrent neural networks. *Neural Compututation*, *26*(5), 953–973. doi: 10.1162/NECO_a_00579

Šíma, J., & Cabessa, J. (2023). Energy complexity of fully-connected layers. In I. Rojas, G. Joya, & A. Catala (Eds.), *Proceedings of the 17th international work-conference on artificial neural networks (IWANN 2023), Part I, LNCS* (Vol. 14134, pp. 3–15). Springer. doi: 10.1007/978-3-031-43085-5_1

Šíma, J., Vidnerová, P., & Mrázek, V. (2023). Energy complexity model for convolutional neural networks. In L. Iliadis, A. Papaleonidas, P. Angelov, & C. Jayne

(Eds.), *Proceedings of the 32nd international conference on artificial neural networks (ICANN 2023), Part X, LNCS* (Vol. 14263, pp. 186–198). Springer. doi: 10.1007/978-3-031-44204-9_16

Sze, V., Chen, Y., Yang, T., & Emer, J. S. (2017). Efficient processing of deep neural networks: A tutorial and survey. *Proceedings of the IEEE*, *105*(12), 2295–2329. doi: 10.1109/JPROC.2017.2761740

Sze, V., Chen, Y., Yang, T., & Emer, J. S. (2020). *Efficient processing of deep neural networks*. Morgan & Claypool Publishers. doi: 10.2200/S01004ED1V01Y202004CAC050

Uchizawa, K., Douglas, R. J., & Maass, W. (2006). On the computational power of threshold circuits with sparse activity. *Neural Computation*, *18*(12), 2994–3008. doi: 10.1162/neco.2006.18.12.2994

Wu, Y. N., Emer, J. S., & Sze, V. (2019). Accelergy: An architecture-level energy estimation methodology for accelerator designs. In D. Z. Pan (Ed.), *Proceedings of the IEEE/ACM international conference on computer aided design (ICCAD 2019)*. doi: 10.1109/ICCAD45719.2019.8942149

Yang, T., Chen, Y., Emer, J. S., & Sze, V. (2017). A method to estimate the energy consumption of deep neural networks. In M. B. Matthews (Ed.), *Proceedings of the IEEE 51st asilomar conference on signals, systems, and computers (ACSSC 2017)* (pp. 1916–1920). doi: 10.1109/ACSSC.2017.8335698

Zhou, Z., Liu, J., Gu, Z., & Sun, G. (2023). Energon: Toward efficient acceleration of transformers using dynamic sparse attention. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, *42*(1), 136–149. doi: 10.1109/TCAD.2022.3170848