Energy Complexity Model for Convolutional Neural Networks

Jiří Šíma$^{1[0000-0001-8248-9425]}$, Petra Vidnerová$^{1[0000-0003-3879-3459]}$, and Vojtěch Mrázek$^{2[0000-0002-9399-9313]}$

^{1} Institute of Computer Science of the Czech Academy of Sciences, Prague, Czechia
`{sima,petra}@cs.cas.cz`

^{2} Faculty of Information Technology, Brno University of Technology, Brno, Czechia
`mrazek@fit.vutbr.cz`

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 power 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 power consumption of different CNN hardware implementations. We calculate energy complexity in this model for two common dataflows which, according to statistical tests, fits asymptotically very well the power consumption estimated by the Time/Accelergy program for convolutional layers on the Simba and Eyeriss hardware platforms. The model opens the possibility of proving principal limits on the energy efficiency of CNN hardware accelerators.

Keywords: Convolutional neural networks · Energy complexity · Dataflow.

1 Introduction

Deep neural networks (DNNs) represent a cutting-edge machine learning technology with countless applications in artificial intelligence (AI). In many cases such as smart glasses 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. For instance, in error-tolerant applications such as image classification, the use of approximate computing methods [4] (e.g. low float precision, approximate multipliers) can save enormous amount of energy at the cost of only a small loss in accuracy.

Recently, there have been great advances in techniques [8] that enable energy-efficient DNN processing on a variety of hardware platforms (e.g. GPUs, FPGAs, in-memory computing architectures) which reduce the computational cost of DNNs through hardware design and/or approximation of DNN models. For a given hardware implementation of DNN, the actual power consumption of the 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 for machine-independent exploration of DNN energy measures.

Some computer programs [5, 9] can optimize the power consumption for a particular DNN on various hardware platforms using different dataflow mapping methods. 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 [10]. 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), to identify efficient algorithms and to establish their principal limits by proving lower bounds.

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 performed 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 Buffer. This process requires a certain number of data transfers between DRAM and Buffer which defines the data energy measure.

The energy complexity model of CNNs is exploited for calculating the theoretical energy in the context of 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.

The upper bounds on energy complexity are compared to the actual power consumption of evaluating CNNs on the Simba [6] and Eyeriss [2] hardware platforms which is estimated by using the Timeloop/Accelergy software tool [5, 9]. The used 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. 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 empirical power consumptions, 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. Hence, the simplified energy complexity model captures asymptotically all important sources of energy consumption that are common to diverse hardware implementations of CNNs. The model can also be exploited for proving lower bounds on energy complexity of CNNs in order to establish asymptotic limits on energy efficiency of any CNN hardware accelerators. The optimal energy bounds have already been proven for fully-connected layers [7] as a special case and starting point for convolutional layers.

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 and a trivial lower bound on the data energy is shown. Section 4 presents two common energy-efficient dataflows which provide upper bounds on the data energy. Section 5 validates the energy complexity model by comparing the theoretical energy bounds for AlexNet-like architectures to its power consumption estimated by the Timeloop/Accelergy program for the Simba and Eyeriss hardware platforms. Section 6 summarizes the results.

2 Convolutional Neural Networks

In order to define an energy complexity measure, we first formalize and introduce notations for a *convolutional neural network* (CNN) $\mathcal{N}$. Its *multi-layered* architecture can be described by a directed acyclic graph (V, E) whose vertices in V, called *macro-units*, are matrices of neurons, while its directed edges in $E \subset V \times V$ are incident on macro-units whose neurons are connected. The macro-units are grouped into $D+1$ disjoint *layers*, indexed by level $\lambda = 0, \dots, D$, starting with the zeroth *input* layer, followed by *hidden* layers, and ending with the *output* layer $Y \subset V$ at the level D. We assume that the edges are only between adjacent layers, which means macro-units in any layer $\lambda \in \{0, \dots, D-1\}$ can only be connected to macro-units in the subsequent layer $\lambda + 1$. Denote $f_{\leftarrow} = \{g \in V \mid (g, f) \in E\}$ and $f^{\rightarrow} = \{h \in V \mid (f, h) \in E\}$ for any macro-unit $f \in V$.

Each layer $\lambda \in \{0, \dots, D\}$ is composed of $d_\lambda > 0$ macro-units which are $m_\lambda \times n_\lambda$ matrices of neurons arranged in $m_\lambda > 0$ rows and $n_\lambda > 0$ columns, representing so-called *feature maps*. The parameters m_λ, n_λ, and d_λ are usually called the *height*, *width*, and *depth* of layer λ, respectively. In addition, each non-input layer $\lambda \in \{1,\ldots,D\}$ is characterized by the size $r_\lambda \times s_\lambda$ of its so-called *receptive fields* which are rectangular (usually square) local regions in feature maps represented by $g \in f_\leftarrow$ in layer $\lambda - 1$ from which the connections lead to individual neurons in a macro-unit f belonging to layer λ. Thus, each neuron in f is associated with a specific receptive field of the same size, representing its scanning window to feature maps $g \in f_\leftarrow$ in the preceding layer $\lambda - 1$, which assumes $0 < r_\lambda \leq m_{\lambda-1}$ and $0 < s_\lambda \leq n_{\lambda-1}$. The length of vertical or horizontal shifts of the scanning window on feature map g for adjacent neurons in matrix f, is given by a so-called *stride* $\sigma_\lambda > 0$ in terms of the number of rows or columns in g, respectively, which is a parameter unique to the layer λ.

In order to compensate for underrepresenting the neurons that are located at the edge of feature maps, the macro-units $g \in f_\leftarrow$ in layer $\lambda - 1$ are formally extended by π_λ rows and π_λ columns (of zero-state neurons) both from above and below, and to the left and right, respectively, where the parameter π_λ called *padding* satisfies $0 \leq \pi_\lambda \leq \max(r_\lambda, s_\lambda)$, which results in their extended formal size $(m_{\lambda-1} + 2\pi_\lambda) \times (n_{\lambda-1} + 2\pi_\lambda)$. Altogether, the size $m_\lambda \times n_\lambda$ of feature maps in the λth layer can be calculated as

$$m_\lambda = \left\lceil \frac{m_{\lambda-1} - r_\lambda + 2\pi_\lambda}{\sigma_\lambda} \right\rceil + 1\,, \quad n_\lambda = \left\lceil \frac{n_{\lambda-1} - s_\lambda + 2\pi_\lambda}{\sigma_\lambda} \right\rceil + 1 \qquad (1)$$

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

Apart from input and output layers, we distinguish three types of layers in CNNs, which are called convolutional, pooling, and fully connected layers. In particular, the first $C < D$ hidden layers in $\mathcal{N}$ include general *convolutional* layers, at times interlaced with *(max) pooling* layers from $\Pi \subset \{1,\ldots,C\}$. We assume that any non-input layer $\lambda \in \{1,\ldots,D\} \setminus \Pi$ that is not pooling, is fully connected with the preceding layer $\lambda - 1$ at the macro-unit level, which means that for every macro-unit f in this layer λ, the set $f_\leftarrow$ contains all macro-units from layer $\lambda - 1$, and hence $|f_\leftarrow| = d_{\lambda-1}$. On the other hand, any macro-unit f in a pooling layer $\lambda \in \Pi$ has only one incoming edge that leads from unique macro-unit g in the preceding layer $\lambda - 1$, that is, $f_\leftarrow = \{g\}$ and $d_\lambda = d_{\lambda-1}$, and the feature map represented by g is partitioned into square non-overlapping receptive fields associated with f, which means $\sigma_\lambda = r_\lambda = s_\lambda$ and $\pi_\lambda = 0$. The remaining layers on the top of $\mathcal{N}$ from level $C + 1$ through D are usual *fully connected* layers of single neurons which constitute trivial feature maps f of size 1×1, that is, $m_\lambda = n_\lambda = \sigma_\lambda = 1$ and $\pi_\lambda = 0$ for every $\lambda = C + 1,\ldots,D$. Each neuron f in the (so-called *flattening*) layer $C + 1$ is a trivial macro-unit that collects its inputs from all neurons in the Cth layer, which means the receptive fields of f coincide with feature maps represented by $g \in f_\leftarrow$, that is, $r_{C+1} = m_C$ and $s_{C+1} = n_C$, whereas $r_\lambda = s_\lambda = 1$ for every $\lambda = C + 2,\ldots,D$.

Every macro-unit $f \in V$ in a non-pooling layer $\lambda \in \{1,\ldots,D\} \setminus \Pi$ is associated with a real *bias* $b_f \in \mathbb{R}$, whereas any edge $(g, f) \in E$ leading from $g \in f_\leftarrow$ 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 $w_{fg}(i, j)$ for every $i = 1,\ldots,r_\lambda$ and $j = 1,\ldots,r_\lambda$. The *state (output)* $\mathbf{Y}_f \in \mathbb{R}^{m_\lambda \times n_\lambda}$ of any macro-unit $f \in V$ is an $m_\lambda \times n_\lambda$ matrix with real entries $y_f(k,\ell)$ for every $k = 1,\dots,m_\lambda$ and $\ell = 1,\dots,n_\lambda$. We formally define $y_f(k,\ell) = 0$ if $k < 1$ or $k > m_\lambda$ or $\ell < 1$ or $\ell > n_\lambda$ for padding neurons in f.

At the beginning of a computation, an external input is presented to $\mathcal{N}$ by setting the states $\mathbf{Y}_f$ of macro-units f in the input layer. In general step, assume that for a macro-unit $f \in V$ in a layer λ, the states $\mathbf{Y}_g$ have been computed for every $g \in f_\leftarrow$. First assume that $\lambda \in \{1,\dots,D\} \setminus \Pi$ is not a pooling layer. Then the *excitation* $\mathbf{\Xi}_f \in \mathbb{R}^{m_\lambda \times n_\lambda}$ of f, which is an $m_\lambda \times n_\lambda$ matrix with real entries $\xi_f(k,\ell)$ for every $k = 1,\dots,m_\lambda$ and $\ell = 1,\dots,n_\lambda$, is evaluated as

$$\xi_f(k,\ell) = b_f + \sum_{g \in f_\leftarrow} \sum_{i=1}^{r_\lambda} \sum_{j=1}^{s_\lambda} w_{fg}(i,j)\, y_g\big((k-1)\sigma_\lambda - \pi_\lambda + i\,,(\ell-1)\sigma_\lambda - \pi_\lambda + j\big) \quad (2)$$

for every $k = 1,\dots,m_\lambda$ and $\ell = 1,\dots,n_\lambda$. The state $\mathbf{Y}_f$ of f in the hidden layer $\lambda < D$ is computed by applying the *rectified linear activation function* (ReLU) to its excitation $\mathbf{\Xi}_f$ element-wise, that is,

$$y_f(k,\ell) = \max\big(0\,,\xi_f(k,\ell)\big) \quad \text{for every } k = 1,\dots,m_\lambda \text{ and } \ell = 1,\dots,n_\lambda\,. \quad (3)$$

For the output layer $\lambda = D$, the state $\mathbf{Y}_f = (y_f(1,1)) = y_f$ of trivial macro-unit f with $m_\lambda = n_\lambda = 1$ is computed as the *softmax function* of excitations $\mathbf{\Xi}_h = (\xi_h(1,1)) = \xi_h$ for every output neuron $h \in Y$, that is, $y_f = e^{\xi_f}/(\sum_{h\in Y} e^{\xi_h}) \in (0,1)$. For the pooling layer $\lambda \in \Pi$ which satisfies $\sigma_\lambda = r_\lambda = s_\lambda$ and $\pi_\lambda = 0$, the state $\mathbf{Y}_f$ of f with $f_\leftarrow = \{g\}$, is computed as $y_f(k,\ell) = \max_{i,j\in\{1,\dots,r_\lambda\}} y_g\big((k-1)r_\lambda + i\,,(\ell-1)r_\lambda + j\big)$ for every $k = 1,\dots,m_\lambda$ and $\ell = 1,\dots,n_\lambda$.

3 Energy Complexity Model

In this section, we introduce a simplified hardware-independent energy complexity model for evaluating a CNN defined in Section 2, which captures the main sources of power 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 1. 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 $f \in V$ and $g \in f_\leftarrow$. 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 implemented, especially the *multiply-and-accumulate (MAC)* operations $S \leftarrow S + wy$ for evaluating excitations (2) of $\mathcal{N}$ where w, y, and S is a filter weight, a neuron state from a previous-layer feature map, and a partial sum accumulating an output of a current-layer neuron, respectively. 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.

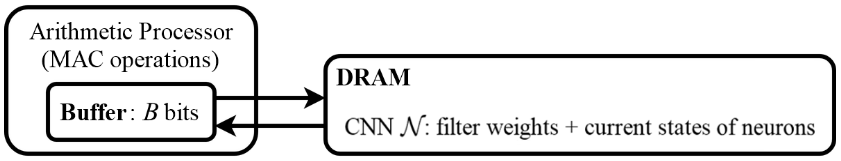

Fig. 1. The energy complexity model

As has been discussed in Section 1, the energy complexity of evaluating $\mathcal{N}$ consists of the *computation energy* and the *data energy* [10]:

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

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 as e.g. in [1], which means energy complexity (4) is defined as a simple sum of energy costs over separate convolutional and fully-connected layers in $\mathcal{N}$ only, while the less energy-intensive max pooling layers are omitted:

$$E = \sum_{\lambda \in \{1,\ldots,D\} \setminus \Pi} \left(E^{\lambda}_{\mathrm{comp}} + E^{\lambda}_{\mathrm{data}} \right) \tag{5}$$

where $E^{\lambda}_{\mathrm{comp}}$ is the computation energy and $E^{\lambda}_{\mathrm{data}}$ is the data energy for evaluating a convolutional (or fully-connected) layer λ. For a particular layer $\lambda \in \{1,\ldots,D\} \setminus \Pi$, single-neuron states in layer $\lambda - 1$, λ, and corresponding filter weights are called *inputs*, *outputs*, and *weights of layer* λ, respectively.

The computation energy $E^{\lambda}_{\mathrm{comp}}$ is defined as the number of MACs in layer $\lambda \in \{1,\ldots,D\} \setminus \Pi$, 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 layer λ equals to the number $d_\lambda m_\lambda n_\lambda$ of single-neuron excitations (2) in layer λ (i.e. the number of outputs of λ), multiplied by the number $d_{\lambda-1} r_\lambda s_\lambda$ of inputs to λ that contribute to each of these excitations, which gives

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

The data energy $E^{\lambda}_{\mathrm{data}}$ is defined as the number of read and write accesses to DRAM when evaluating layer $\lambda \in \{1,\ldots,D\} \setminus \Pi$, multiplied by the number b of bits in a floating-point representation of numbers to be transferred between DRAM and Buffer. This energy complexity can be split into three components that count the DRAM accesses separately for the inputs, outputs, and weights:

$$E^{\lambda}_{\mathrm{data}} = E^{\lambda}_{\mathrm{inputs}} + E^{\lambda}_{\mathrm{outputs}} + E^{\lambda}_{\mathrm{weights}} \, . \tag{7}$$

Obviously, the numbers of inputs, outputs, and weights of λ 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 λ:

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

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

4 Upper Bounds on Energy Complexity

In the following two subsections, we present two common dataflows for evaluating a convolutional layer $\lambda \in \{1,\ldots,D\} \setminus \Pi$ and calculate their theoretical data energy complexity $E^{\lambda}_{\text{data}}$. 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,

$$E^{\lambda}_{\text{weights}} = b\,d_{\lambda}(d_{\lambda-1}r_{\lambda}s_{\lambda}+1)\,. \tag{9}$$

We will assume a sufficiently large capacity of Buffer:

$$B \geq b\left(2m_{\lambda}n_{\lambda}+1\right)\,. \tag{10}$$

Moreover, let $K = \{1,\ldots,r_{\lambda}\} \times \{1,\ldots,s_{\lambda}\}$ be a set of all kernel indices in layer λ and for any $(i_0,j_0) \in K$, define a subset of K,

$$[(i_0,j_0)] = \{(i_0+k\sigma_{\lambda}, j_0+\ell\sigma_{\lambda}) \in K \mid k,\ell \in \mathbb{Z}\} \tag{11}$$

where σ_{λ} is a corresponding stride and $\mathbb{Z}$ denotes the set of integers. Observe that the set $P = \{[(i_0,j_0)] \mid (i_0,j_0) \in K\}$ creates a partition of K which consists of $|P| = \sigma^2_{\lambda}$ disjoint parts $[i,j]$ for every $i = 1,\ldots,\sigma_{\lambda}$ and $j = 1,\ldots,\sigma_{\lambda}$.

4.1 The Dataflow with Write-Once Outputs

We describe the dataflow in which each output is written to DRAM only once. For each macro-unit f in layer λ, one after the other, its excitation $\boldsymbol{\Xi}_f$ of size $m_{\lambda}n_{\lambda}$ floats is accumulated in Buffer (cf. its capacity (10)) according to (2) as follows. At the beginning, the bias b_f is read into Buffer (corresponding to 1 DRAM access inside the parentheses in (9)), which initializes the evaluation of $\boldsymbol{\Xi}_f$. Then for each macro-unit $g \in f_{\leftarrow}$ and for each part $[(i_0,j_0)]$ of the partition P, one by one, the collection of $m_{\lambda}n_{\lambda}$ inputs to f,

$$y_g\big((k-1)\sigma_{\lambda} - \pi_{\lambda} + i_0\,,(\ell-1)\sigma_{\lambda} - \pi_{\lambda} + j_0\big) \tag{12}$$

for every $k = 1,\ldots,m_{\lambda}$ and $\ell = 1,\ldots,n_{\lambda}$, is read into Buffer with space left still for at least one float according to (10), which is reserved for one weight. The indices of inputs in (12), $(k_1-1)\sigma_{\lambda} - \pi_{\lambda} + i_0 = (k_2-1)\sigma_{\lambda} - \pi_{\lambda} + i$ and $(\ell_1-1)\sigma_{\lambda} - \pi_{\lambda} + j_0 = (\ell_2-1)\sigma_{\lambda} - \pi_{\lambda} + j$ coincide for some $k_1,k_2 \in \{1,\ldots,m_{\lambda}\}$,

$\ell_1, \ell_2 \in \{1, \ldots, n_\lambda\}$, and $(i,j) \in K$ iff $i = i_0 + k\sigma_\lambda$ and $j = j_0 + \ell\sigma_\lambda$ for $k = k_1 - k_2$ and $\ell = \ell_1 - \ell_2$ iff $(i,j) \in [i_0, j_0]$ due to (11). This means that in (2) each input (12) from this collection is multiplied only by weights $w_{fg}(i,j)$ such that $(i,j) \in [i_0, j_0]$, which are read one by one into Buffer and the respective MACs are performed. The partition P ensures that each weight and each input is read into Buffer only once for one macro-unit f over all parts of P and $g \in f_\leftarrow$, which implies (9). After the excitation $\mathbf{\Xi}_f$ is eventually evaluated, the state $\mathbf{Y}_f \in \mathbb{R}^{m_\lambda \times n_\lambda}$ of macro-unit f is computed according to (3) and written to DRAM.

Altogether, each output is thus written to DRAM only once, which gives

$$E^\lambda_{\text{outputs}} = b\, d_\lambda m_\lambda n_\lambda \,, \tag{13}$$

while each input is read once for every macro-unit f in layer λ, which implies

$$E^\lambda_{\text{inputs}} = b\, d_\lambda d_{\lambda-1} m_{\lambda-1} n_{\lambda-1} \,. \tag{14}$$

The dataflow provides the following upper bound on the data energy of layer λ:

$$E^\lambda_{\text{data}} \le b\, d_\lambda \left(d_{\lambda-1} m_{\lambda-1} n_{\lambda-1} + m_\lambda n_\lambda + d_{\lambda-1} r_\lambda s_\lambda + 1\right) \tag{15}$$

according to (7), (14), (13), and (9), which differs only in the number of DRAM accesses for reading inputs by factor d_λ from the trivial lower bound (8).

In addition, we will introduce an alternative dataflow of the same data energy (15), provided that Buffer capacity is bounded as

$$B \ge b\left(m_\lambda n_\lambda + r_\lambda s_\lambda + 1\right) \,, \tag{16}$$

cf. (10). For each macro-unit f in layer λ, one after the other, its excitation $\mathbf{\Xi}_f$ of size $m_\lambda n_\lambda$ floats is accumulated in Buffer according to (2), starting with the bias b_f which is read into Buffer to initialize the evaluation of $\mathbf{\Xi}_f$. Next, for each $g \in f_\leftarrow$, one by one, the filter $\mathbf{W}_{fg} \in \mathbb{R}^{r_\lambda \times s_\lambda}$ of size $r_\lambda s_\lambda$ is first read into Buffer, followed by single inputs $y_g\big(k_0, \ell_0\big)$, one after the other, for every $k_0 = 1, \ldots, m_{\lambda-1}$ and $\ell_0 = 1, \ldots, n_{\lambda-1}$. For each such an input $y_g\big(k_0, \ell_0\big)$, all partially evaluated excitations $\xi_f(k,\ell)$ such that $k_0 = (k-1)\sigma_\lambda - \pi_\lambda + i$ and $\ell_0 = (\ell-1)\sigma_\lambda - \pi_\lambda + j$ for some $(i,j) \in K$, are updated in Buffer by performing MACs with corresponding weights $w_{fg}(i,j)$ etc. After $\mathbf{\Xi}_f$ is eventually evaluated, the corresponding output $\mathbf{Y}_f \in \mathbb{R}^{m_\lambda \times n_\lambda}$ is computed by (3) and written to DRAM. Clearly, each weight and each input is read into Buffer once for one macro-unit f, which proves the upper bound (15) also for this alternative dataflow.

4.2 The Dataflow with Read-Once Inputs

We describe the dataflow in which each input is read into Buffer only once. For each macro-unit g in layer $\lambda - 1$ and each part $[(i_0, j_0)]$ of the partition P, one after the other, the collection (12) of $m_\lambda n_\lambda$ inputs are read into Buffer to accumulate excitations $\mathbf{\Xi}_f$ for every $f \in g^\rightarrow$ according to (2) as follows. For each macro-unit $f \in g^\rightarrow$, one by one, either its bias b_f is read into Buffer to initialize the evaluation of $\boldsymbol{\Xi}_f$ at the beginning (when the very first collection (12) is in Buffer) or its partially evaluated excitation $\boldsymbol{\Xi}_f$ of size $m_\lambda n_\lambda$ floats is read into Buffer, over which the respective MACs are performed. For this purpose, the corresponding weights $w_{fg}(i,j)$ such that $(i,j) \in [i_0, j_0]$ are one by one read into Buffer. Then, either the partially evaluated excitation $\boldsymbol{\Xi}_f$ is written to DRAM, or at the end when $\boldsymbol{\Xi}_f$ is completely evaluated (after the very last collection (12) over all pairs of macro-units g and parts of P, is in Buffer), $\boldsymbol{\Xi}_f$ is used for computing the state $\mathbf{Y}_f \in \mathbb{R}^{m_\lambda \times n_\lambda}$ of macro-unit f according to (3), which is written to DRAM. The dataflow is thus implemented within Buffer capacity (10).

Moreover, the partition P ensures that each weight (including bias) as well as each input is read into Buffer only once, which implies (9) and

$$E^\lambda_{\text{inputs}} = b\, d_{\lambda-1} m_{\lambda-1} n_{\lambda-1}\,, \tag{17}$$

respectively. Any accumulated excitation is read (except for its initialization by a bias at the beginning) and written once for each of the $d_{\lambda-1}$ macro-units in layer $\lambda - 1$ and each part of the σ_λ^2 parts of partition P, which gives

$$E^\lambda_{\text{outputs}} = b \left(2 d_{\lambda-1} \sigma_\lambda^2 - 1\right) d_\lambda m_\lambda n_\lambda\,. \tag{18}$$

Hence, this dataflow provides another upper bound on the data energy of layer λ:

$$E^\lambda_{\text{data}} \le b \left(d_{\lambda-1} m_{\lambda-1} n_{\lambda-1} + \left(2 d_{\lambda-1} \sigma_\lambda^2 - 1\right) d_\lambda m_\lambda n_\lambda + d_\lambda \left(d_{\lambda-1} r_\lambda s_\lambda + 1\right)\right) \tag{19}$$

according to (7), (17), (18), and (9). This bound is comparable to (15) if the number of single neurons in layers λ and $\lambda - 1$ is roughly the same, that is, $d_{\lambda-1} m_{\lambda-1} n_{\lambda-1} \approx d_\lambda m_\lambda n_\lambda$, since $\sigma_\lambda^2 m_\lambda n_\lambda \approx m_{\lambda-1} n_{\lambda-1}$ according to (1). Nevertheless, the multiplicative constant 2 of leading term $d_\lambda d_{\lambda-1} m_{\lambda-1} n_{\lambda-1}$ in (19), which is caused by storing partially evaluated excitations into DRAM, makes the upper bound (15) more tight than (19).

5 Experimental Validation

In this section, we compare the theoretical energy complexity introduced in Section 3 to the real power consumption estimated by the Timeloop/Accelergy software tool [5, 9] for evaluating DNN accelerator designs. The Timeloop finds an optimal mapping of a convolutional layer specified by its parameters onto a given hardware platform in terms of power consumption estimated by Accelergy which reports the energy statistics. Namely, we have employed Simba [6] and Eyeriss [2] as the target platforms onto which convolutional layers with increasing architectural parameters (i.e. not filter weights) have been mapped. All configuration files used in experiments are publicly available at Github[3].

The computation energy reported by Timeloop/Accelergy corresponds directly to the number of MACs calculated in (6) for E^λ_{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.

[3] 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
(10): $2m_\lambda^2 + 1$	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
(16): $m_\lambda^2 + r_\lambda^2 + 1$	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

For a convolutional layer λ, we measure empirical dependencies of the optimal data energy 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 (starting with the parameter values of the first AlexNet layer), by using the Timeloop/Accelergy framework for the Simba and Eyeriss architectures. These dependencies are then compared to corresponding asymptotic upper bounds on E_{data}^λ in the energy complexity model:

$$E_{\text{data}}^\lambda = O\left(d_\lambda\right) \,,\; E_{\text{data}}^\lambda = O\left(m_{\lambda-1}^2\right) \,,\; E_{\text{data}}^\lambda = O\left(r_\lambda^2\right) \,,\; E_{\text{data}}^\lambda = O\left(\sigma_\lambda^{-2}\right) \,, \quad (20)$$

which are derived from (15) for individual variables (when the other independent parameters are considered to be constant) by using the approximation $m_{\lambda-1}^2 \approx \sigma_\lambda^2 m_\lambda^2$ due to (1). Nevertheless, the asymptotic bounds (20) assume a sufficient Buffer capacity satisfying (10) or (16). Table 1 shows required Buffer capacities for AlexNet convolutional layers in kilobytes (kB) which appear in an order of magnitude to be realistic to common hardware architectures such as Eyeriss [2].

Figure 2 presents the results of experimental comparison of energy-efficient CNN hardware implementations to our theoretical energy complexity model. 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 AlexNet-like convolutional layers λ with increasing parameters d_λ, $m_{\lambda-1} = n_{\lambda-1}$, $r_\lambda = s_\lambda$, and σ_λ, each separately. 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 (20), including multiplicative and additive coefficients c_2 and c_1, respectively. As depicted in Figure 2, these coefficients are approximated by the method of least squares so that the theoretical data energy E_{data}^λ (dashed lines) fits energy estimates by Timeloop/Accelergy (displayed by bars), which confirms the asymptotic trends (20) in the 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 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 $a = 0$).

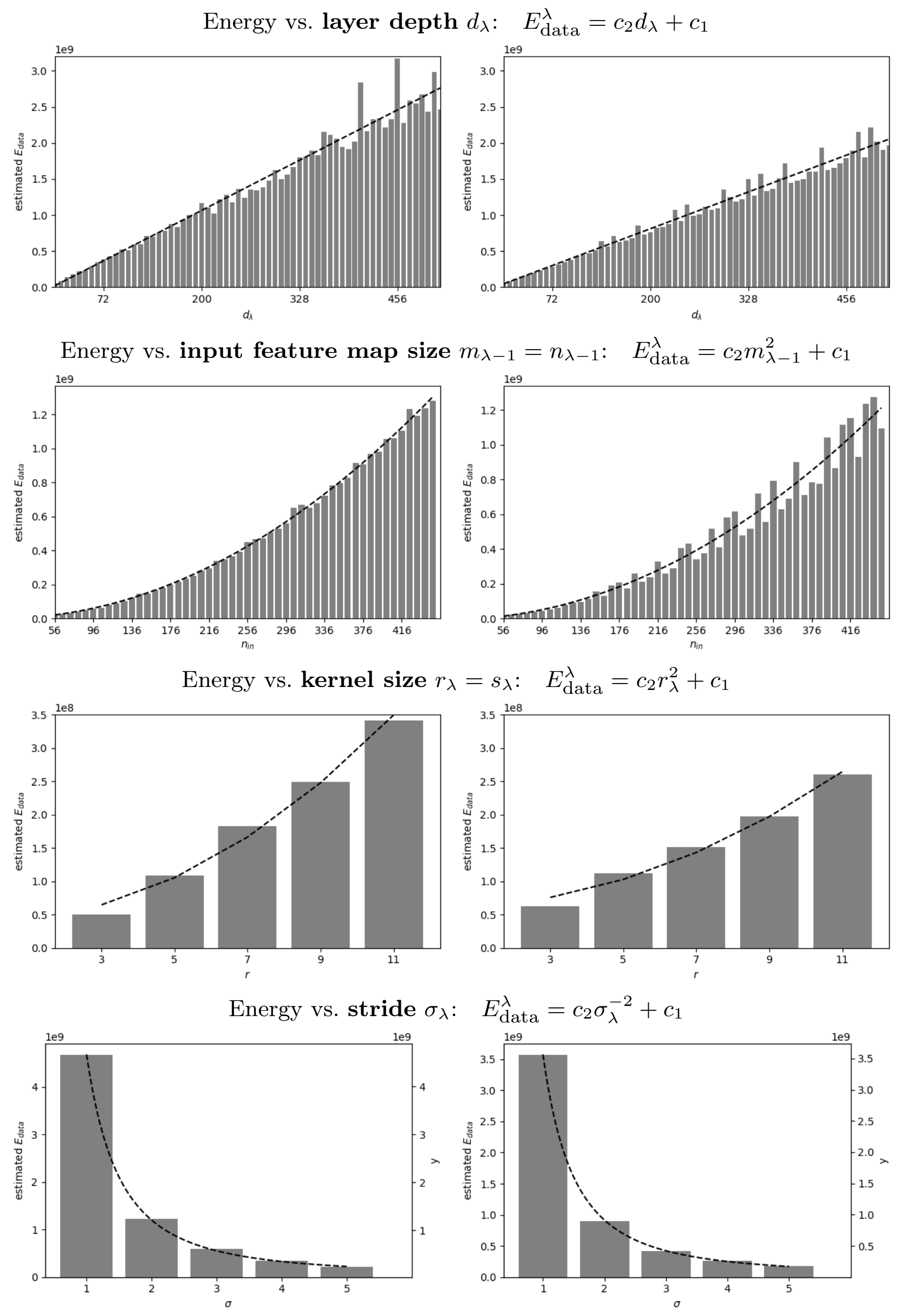

Fig. 2. The data energy estimates by Timeloop/Accelergy (displayed by bars) for AlexNet-like convolutional layer λ with increasing parameters d_λ, $m_{\lambda-1}$,r_λ, and σ_λ, each separately (from top to bottom), on the Simba (left) and Eyeriss (right) architectures, which fit the asymptotic trends (20) in the energy complexity model (dashed lines).

6 Conclusion

In this paper, we have introduced a hardware-independent energy complexity model for CNNs that captures asymptotically all important sources of power consumption of their diverse hardware implementations. Upper bounds on energy complexity have been derived in this model for two common energy-efficient dataflows. The underlying theoretical asymptotic trends have been validated by statistical tests to fit the energy consumption estimated by the Timeloop/Accelergy program for CNNs of AlexNet-like architectures on the Simba and Eyeriss hardware platforms. In future research we plan to prove matching lower bounds on the data energy for convolutional layers. Partial results along this direction have already been achieved for a special case of fully-connected layers [7]. The proposed model thus allows to determine the principal limits to which heuristic optimizers e.g. based on evolutionary algorithms [3] can reach.

Acknowledgements The research 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 his advice on statistical tests.

References

1. Alwani, M., et al.: Fused-layer CNN accelerators. In: Proc. of IEEE/ACM MICRO 2016. pp. 22:1–22:12 (2016). https://doi.org/10.1109/MICRO.2016.7783725
2. Chen, Y., Emer, J.S., Sze, V.: Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks. In: Proc. of ACM/IEEE ISCA 2016. pp. 367–379 (2016). https://doi.org/10.1109/ISCA.2016.40
3. Kao, S.C., Krishna, T.: GAMMA: automating the HW mapping of DNN models on accelerators via genetic algorithm. In: Proc. of ACM/IEEE ICCAD 2020. pp. 44:1–44:9 (2020). https://doi.org/10.1145/3400302.3415639
4. Mittal, S.: A survey of techniques for approximate computing. ACM Comput. Surv. **48**(4), 62:1–62:33 (2016). https://doi.org/10.1145/2893356
5. Parashar, A., et al.: Timeloop: A systematic approach to DNN accelerator evaluation. In: Prof. of IEEE ISPASS 2019. pp. 304–315 (2019). https://doi.org/10.1109/ISPASS.2019.00042
6. Shao, Y.S., et al.: Simba: Scaling deep-learning inference with multi-chip-module-based architecture. In: Proc. of IEEE/ACM MICRO 2019. pp. 14–27 (2019). https://doi.org/10.1145/3352460.3358302
7. Šíma, J., Cabessa, J.: Energy complexity of fully-connected layers. In: Proc. of IWANN 2023. LNCS, Springer (2023)
8. Sze, V., et al.: Efficient Processing of Deep Neural Networks. Synthesis Lectures on Computer Architecture, Morgan & Claypool Publishers (2020). https://doi.org/10.2200/S01004ED1V01Y202004CAC050
9. Wu, Y.N., Emer, J.S., Sze, V.: Accelergy: An architecture-level energy estimation methodology for accelerator designs. In: Proc. of IEEE/ACM ICCAD 2019 (2019). https://doi.org/10.1109/ICCAD45719.2019.8942149
10. Yang, T., Chen, Y., Emer, J.S., Sze, V.: A method to estimate the energy consumption of deep neural networks. In: Proc. of IEEE ACSSC 2017. pp. 1916–1920 (2017). https://doi.org/10.1109/ACSSC.2017.8335698