Energy Complexity of Convolutional Neural Networks

Jiří Šíma\textsuperscript{a,}\textsuperscript{*}, Petra Vidnerová\textsuperscript{a}, Vojtěch Mrázek\textsuperscript{b}

\textsuperscript{a}Institute of Computer Science of the Czech Academy of Sciences, Pod Vodárenskou
věží 271/2, Prague 8, 182 00, Czechia
\textsuperscript{b}Faculty of Information Technology, Brno University of
Technology, Božetěchova 1/2, Brno, 612 00, Czechia

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 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 power consumption of CNN implementations on the Simba and Eyeriss hardware platforms, estimated by the Time/Accelergy program, which validates the proposed energy complexity model for CNNs.

Keywords: Convolutional neural networks, Energy complexity, Dataflow.

\textsuperscript{*}Corresponding author

Email addresses: sima@cs.cas.cz (Jiří Šíma), petra@cs.cas.cz (Petra Vidnerová), mrazek@fit.vutbr.cz (Vojtěch Mrázek)
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 (Mittal, 2016), e.g. low float precision (Gupta et al., 2015), approximate multipliers (Ansari et al., 2020), etc. The second approach is related to recent major advances in techniques (Sze et al., 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 power 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 for machine-independent exploration of DNN energy measures. Nevertheless, there are software tools that can optimize the power 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 power consumption estimated by Accelergy tool (Wu et al., 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 et al., 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), 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 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 (Šima and 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 con-
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 power consumption of evaluating CNNs on the Simba (Shao et al., 2019) and Eyeriss (Chen et al., 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 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. These individual upper bounds match the corresponding lower bounds for constant Buffer capacity.

In addition, the minimum real power consumption of CNN implementations on Simba estimated by Timeloop/Accelergy fit 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 (Uchizawa et al., 2006; Šíma, 2014). 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
power consumption estimated by the Timeloop/Accelergy program for the Simba and Eyeriss hardware platforms. Section 7 summarizes the results. A preliminary conference version (Šíma et al., 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 any layer \(\lambda \geq 1\), a macro-unit \(g \in \mathcal{N}_{\lambda-1}\) can only be connected 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 \(m_\lambda\), \(n_\lambda\), and \(d_\lambda\) are usually called the height, width, and depth of layer \(\lambda \in \{0, \ldots, D\}\), 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 \(\mathcal{N}_0\) of size \(m_0 \times n_0\) input neurons, each encoding one pixel of the respective channel in the RGB color model.

In addition, each non-input layer \(\lambda \geq 1\) 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 macro-units \(g \in \mathcal{N}_{\lambda-1}\) from which the connections lead to individual neurons in a macro-unit \(f \in \mathcal{N}_\lambda\). Thus, each neuron in macro-unit \(f\) is associated with a specific receptive field of the same size, representing its scanning window to feature maps \(g \in \mathcal{N}_{\lambda-1}\), 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 layer \(\lambda\). This is illustrated on a convolutional layer which is schematically depicted in Figure 1.
In order to compensate for underrepresenting the neurons that are located at the edge of feature maps, the macro-units in convolutional layers are formally *padded* by several rows and columns (of zero-state neurons) both from above and below, and to the left and right, respectively. For simplicity, we assume zero padding which is sufficient for our energy considerations. Altogether, the size $m_\lambda \times n_\lambda$ of feature maps in the $\lambda$th layer can be calculated as

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

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

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 convolutional layer $\lambda \in \Gamma$, 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 $\lambda \notin \Gamma$ 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 $\lambda$ in $\Gamma$ are composed of single neurons $f$ that constitute trivial feature maps of size $1 \times 1$ (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 $\lambda_1$ which collects its inputs from all neurons in the previous layer $\lambda_1 - 1$, that is, $r_{\lambda_1} = m_{\lambda_1 - 1}$ and $s_{\lambda_1} = n_{\lambda_1 - 1}$.

Every macro-unit $f \in N_\lambda$ in a convolutional layer $\lambda \in \Gamma$ is associated with a real bias $b_f \in \mathbb{R}$, and any macro-connection leading from macro-unit $g \in N_{\lambda - 1}$ to $f$ is labeled with a so-called filter (or kernel) $W_{fg} \in \mathbb{R}^{r_\lambda \times s_\lambda}$ 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) $Y_f \in \mathbb{R}^{m_\lambda \times n_\lambda}$ of any macro-unit $f \in N_\lambda$ is an $m_\lambda \times n_\lambda$ matrix with real entries $y_f(k,\ell)$ for every $k = 1, \ldots, m_\lambda$ and $\ell = 1, \ldots, n_\lambda$.

For a convolutional layer $\lambda \in \Gamma$, the states $Y_f$ of $f \in N_\lambda$ are evaluated using the following weighted sums

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

for every $k = 1, \ldots, m_\lambda$ and $\ell = 1, \ldots, n_\lambda$. In particular, for a non-output layer $\lambda < D$, the rectified linear activation function ReLU is employed as

$$y_f(k,\ell) = \text{ReLU}(S_f(k,\ell)) = \max(0, S_f(k,\ell))$$

while for the output layer $\lambda = D$, the state $Y_f = (y_f(1,1)) = y_f$ of trivial macro-unit $f \in N_D$ with $m_D = n_D = 1$ is computed by the softmax function as $y_f = e^{S_f}/(\sum_{h \in N_D} e^{S_h}) \in (0,1)$ where $S_h = S_h(1,1)$ for every output neuron $h \in N_D$. For completeness note that for a max pooling layer $\lambda \notin \Gamma$ which satisfies $\sigma_\lambda = r_\lambda = s_\lambda$, the state $Y_f$ of $f \in N_\lambda$ with a single incoming macro-connection from $g \in N_{\lambda - 1}$, is computed as $y_g((k-1)r_\lambda + i, (\ell-1)r_\lambda + j)$ 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 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 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 implemented, especially the multiply-and-accumulate (MAC) operations $S \leftarrow S + wy$ for evaluating the weighted sums (2) 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.

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):

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

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 et al., 2016), which means energy com-
plexity (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:

$$
E = \sum_{\lambda \in \Gamma} \left( E_{\text{comp}}^\lambda + E_{\text{data}}^\lambda \right)
$$  \hspace{1cm} (5)

where $E_{\text{comp}}^\lambda$ is the computation energy and $E_{\text{data}}^\lambda$ is the data energy for evaluating a convolutional layer $\lambda \in \Gamma$. For a particular layer $\lambda \in \Gamma$, single-neuron states in layer $\lambda - 1$, $\lambda$, and corresponding filter weights are called inputs, outputs, and weights of layer $\lambda$, respectively.

The computation energy $E_{\text{comp}}^\lambda$ is defined as the number of MACs in layer $\lambda \in \Gamma$, 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 layer $\lambda$ 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 $\lambda$), multiplied by the number $d_{\lambda-1} r_\lambda s_\lambda$ of inputs $y_g((k-1)\sigma_\lambda+i,(\ell-1)\sigma_\lambda+j)$ to $\lambda$ 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

$$
E_{\text{comp}}^\lambda = C_b d_\lambda m_\lambda n_\lambda d_{\lambda-1} r_\lambda s_\lambda.
$$  \hspace{1cm} (6)

The data energy $E_{\text{data}}^\lambda$ is defined as the number of read and write accesses to DRAM when evaluating layer $\lambda \in \Gamma$, 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:

$$
E_{\text{data}}^\lambda = E_{\text{inputs}}^\lambda + E_{\text{outputs}}^\lambda + E_{\text{weights}}^\lambda.
$$  \hspace{1cm} (7)

### 4. A Simple Lower Bound on Energy Complexity

In this section, we derive a simple lower bound on the data energy $E_{\text{data}}^\lambda$ for any convolutional layer $\lambda \in \Gamma$. Obviously, the numbers of inputs, outputs,
and weights of $\lambda$ 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 $\lambda$:

$$E_{\text{data}}^\lambda \geq b \left( d_{\lambda-1}m_{\lambda-1}n_{\lambda-1} + d_{\lambda}m_{\lambda}n_{\lambda} + d_{\lambda}(d_{\lambda-1}r_{\lambda}s_{\lambda} + 1) \right)$$  \hspace{1cm} (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 layer $\lambda \in \Gamma$, which are the inputs $y_g((k-1)\sigma_{\lambda} + i, (\ell-1)\sigma_{\lambda} + j)$ for all $g \in N_{\lambda-1}$, $k \in \{1, \ldots, m_{\lambda}\}$, $\ell \in \{1, \ldots, n_{\lambda}\}$, $i \in \{1, \ldots, r_{\lambda}\}$, and $j \in \{1, \ldots, s_{\lambda}\}$, the weighted sums $S_f(k, \ell)$ accumulated for the outputs $y_f(k, \ell)$ for all $f \in N_{\lambda}$, $k \in \{1, \ldots, m_{\lambda}\}$, and $\ell \in \{1, \ldots, n_{\lambda}\}$, and the weights $w_{fg}(i, j)$ for all $f \in N_{\lambda}$, $g \in N_{\lambda-1}$, $i \in \{1, \ldots, r_{\lambda}\}$, 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 $f_1 \in N_{\lambda}$, $k_1 \in \{1, \ldots, m_{\lambda}\}$, and $\ell_1 \in \{1, \ldots, n_{\lambda}\}$, and the weight $w_{f_1g_1}(i_1, j_1)$ for some fixed $g_1 \in N_{\lambda-1}$, $i_1 \in \{1, \ldots, r_{\lambda}\}$, 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 $g_1 \in N_{\lambda-1}$, $k_1 \in \{1, \ldots, m_{\lambda}\}$, $\ell_1 \in \{1, \ldots, n_{\lambda}\}$, $i_1 \in \{1, \ldots, r_{\lambda}\}$, and $j_1 \in \{1, \ldots, s_{\lambda}\}$, and the weight $w_{f_1g_1}(i_2, j_2)$ for some fixed $f_1 \in N_{\lambda}$ 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 $w_{f_1g_1}(i_1, j_2) \cdot y_{g_1}((k_1-1)\sigma_{\lambda} + i_1, (\ell_1-1)\sigma_{\lambda} + j_1)$.  

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 $\lambda$ can be lower bounded by the number of MAC operations divided by the maximum number new MACs enabled by one DRAM access, that is,

$$E_{\text{data}}^\lambda \geq b \cdot \frac{d_{\lambda}m_{\lambda}n_{\lambda}d_{\lambda-1}r_{\lambda}s_{\lambda}}{\lfloor \beta-1 \rfloor}.$$  \hspace{1cm} (9)
Particularly for constant Buffer capacity $B$, we obtain the asymptotic lower bound
\[ E^\lambda_{\text{data}} = \Omega \left( d_\lambda d_{\lambda-1} m_\lambda n_\lambda r_\lambda s_\lambda \right) \] (10)
which can be rewritten as
\[ E^\lambda_{\text{data}} = \Omega \left( \frac{d_\lambda d_{\lambda-1} m_\lambda n_\lambda r_\lambda s_\lambda}{\sigma_\lambda^2} \right) \] (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 subsections, we present two common dataflows for evaluating a convolutional layer $\lambda \in \Gamma$ 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,
\[ E^\lambda_{\text{weights}} = b d_\lambda (d_{\lambda-1} r_\lambda s_\lambda + 1). \] (12)

We will assume a sufficiently large capacity of Buffer:
\[ B = b (2 m_\lambda n_\lambda + 1), \] (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,
\[ g = \bigcup_{i_0, j_0 \in \{1, \ldots, \sigma_\lambda\}} g(i_0, j_0) \] (14)
into $\sigma_\lambda^2$ disjoint grid submaps
\[ g(i_0, j_0) = \left\{ (i_0 + k \sigma_\lambda, j_0 + (\ell - 1) \sigma_\lambda) \mid k \in \{1, \ldots, m_\lambda\}, \ell \in \{1, \ldots, n_\lambda\} \right\}, \] (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
\[ 1 \leq i = i_0 + k_1 \sigma_\lambda \leq r_\lambda \quad \text{and} \quad 1 \leq j = j_0 + \ell_1 \sigma_\lambda \leq s_\lambda \] (16)
for integers $k_1, \ell_1 \in \mathbb{Z}$, since a weight $w_{fg}(i, j)$ is combined with an input $y_g((k-1)\sigma + i_0, (\ell-1)\sigma + j_0)$ 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 $(k-1)\sigma + i_0 = (k_2-1)\sigma + i$ and $(\ell-1)\sigma + j_0 = (\ell_2-1)\sigma + j$ iff $i = i_0 + k_1 \sigma$ and $j = j_0 + \ell_1 \sigma$ 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 $f \in N_\lambda$ representing the feature maps of convolutional layer $\lambda \in \Gamma$ 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 $g \in N_{\lambda-1}$ representing the input feature maps of layer $\lambda - 1$, 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 $m_\lambda \times n_\lambda$ 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).
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:             \{i.e. for all weights $w_{fg}(i_0 + k_1\sigma_\lambda, j_0 + \ell_1\sigma_\lambda)$ shared by submap $g(i_0, j_0)$\}
13:                 $i \leftarrow i_0 + k_1\sigma_\lambda$; $j \leftarrow j_0 + \ell_1\sigma_\lambda$
14:                 read weight $w_{fg}(i, j)$ into Buffer
15:                 for all $k \in \{1, \ldots, m_\lambda\}$, $\ell \in \{1, \ldots, n_\lambda\}$ do
16:                     \{i.e. for all MACs in (2) over the weight $w_{fg}(i, j)$\}
17:                         $S_f(k, \ell) \leftarrow S_f(k, \ell) + w_{fg}(i, j) \cdot y_g((k - 1)\sigma_\lambda + i, (\ell - 1)\sigma_\lambda + j)$
18:                     end for
19:                 end for
20:             \{next shared weight\}
21:         end for
22:     \{next input submap\}
23: end for
24: for all $k \in \{1, \ldots, m_\lambda\}$, $\ell \in \{1, \ldots, n_\lambda\}$ do
25:     write $y_f(k, \ell) \leftarrow \text{ReLU}(S_f(k, \ell))$ to DRAM \{writing $m_\lambda \times n_\lambda$ outputs\}
26: end for
27: \{next feature map\}
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

\[
E_\lambda^{\text{outputs}} = b d_\lambda m_\lambda n_\lambda. \quad (17)
\]

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 \( f \in \mathcal{N}_\lambda \) in layer \( \lambda \) (line 10 within the nested loops 1–26, 6–22, 7–21, and 9–11) due the partition (14), which implies

\[
E_\lambda^{\text{inputs}} = b d_\lambda d_{\lambda-1} m_{\lambda-1} n_{\lambda-1}. \quad (18)
\]

Hence, the dataflow provides the following upper bound on the data energy of layer \( \lambda \):

\[
E_\lambda^{\text{data}} \leq b d_\lambda (d_{\lambda-1} m_{\lambda-1} n_{\lambda-1} + m_\lambda n_\lambda + d_{\lambda-1} r_\lambda s_\lambda + 1) \quad (19)
\]

according to (7), (18), (17), and (12), which differs only in the number of DRAM accesses for reading inputs by factor \( d_\lambda \) from the trivial lower bound (8).
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 $f \in N_\lambda$ 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 $g \in N_{\lambda-1}$ 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 $W_{fg}$\}
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:         $S_f(k, \ell) \leftarrow S_f(k, \ell) + w_{fg}(i, j) \cdot y_g((k-1)\sigma_\lambda + i, (\ell-1)\sigma_\lambda + j)$
14:       end for
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 $y_f(k, \ell) \leftarrow \text{ReLU}(S_f(k, \ell))$ to DRAM \{writing $m_\lambda \times n_\lambda$ outputs\}
20: end for
21: end for \{next feature map\}
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

$$B = b(m_\lambda n_\lambda + r_\lambda s_\lambda + 1),$$

(20) cf. (13). The alternative dataflow differs in that we load the whole filter $W_{fg}$ 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 $\lambda - 1$ 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 of layer $\lambda - 1$, 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 $\lambda - 1$ (lines 1 and 5). 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 6–8).

The next nested loop 9–34 passes through all the macro-units $f \in \mathcal{N}_\lambda$ representing the feature maps of convolutional layer $\lambda \in \Gamma$. 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).
Algorithm 3 The dataflow with read-once inputs and \( B = b(2m_\lambda n_\lambda + 1) \).

1: \( \text{inp} \leftarrow 0 \)
2: for all input feature maps \( g \in \mathcal{N}_{\lambda - 1} \) do
3:   for all \( i_0, j_0 \in \{1, \ldots, \sigma_\lambda\} \) do
4:     \{i.e. for all grid submaps \( g(i_0, j_0) \) from the partition of \( g \)\}
5:     \( \text{inp} \leftarrow \text{inp} + 1 \) \{counts the current number of input submaps\}
6:   for all \( (k, \ell) \in g(i_0, j_0) \) do
7:     read \( g_k(\ell) \) into Buffer \{reading \( m_\lambda \times n_\lambda \) submap inputs\}
8: end for
9: for all feature maps \( f \in \mathcal{N}_\lambda \) do
10:   if \( \text{inp} = 1 \) \{first input submap\} then
11:     read bias \( b_f \) into Buffer
12:     for all \( k \in \{1, \ldots, m_\lambda\}, \ell \in \{1, \ldots, n_\lambda\} \) do
13:       \( S_f(k, \ell) \leftarrow b_f \) \{initialization of \( m_\lambda \times n_\lambda \) weighted sums\}
14:     end for
15:   else
16:     for all \( k \in \{1, \ldots, m_\lambda\}, \ell \in \{1, \ldots, n_\lambda\} \) do
17:       read \( S_f(k, \ell) \) into Buffer \{reading \( m_\lambda \times n_\lambda \) weighted sums\}
18:     end for
19:   end if
20: for all \( (k_1, \ell_1) \) s.t. \( 1 \leq i_0 + k_1 \sigma_\lambda \leq r_\lambda, 1 \leq j_0 + \ell_1 \sigma_\lambda \leq s_\lambda \) do
21: \{i.e. for all weights \( w_{fg}(i_0 + k_1 \sigma_\lambda, j_0 + \ell_1 \sigma_\lambda) \) shared by submap \( g(i_0, j_0) \)\}
22: \( i \leftarrow i_0 + k_1 \sigma_\lambda; j \leftarrow j_0 + \ell_1 \sigma_\lambda \)
23: read weight \( w_{fg}(i, j) \) into Buffer
24: for all \( k \in \{1, \ldots, m_\lambda\}, \ell \in \{1, \ldots, n_\lambda\} \) do
25: \{i.e. for all MACs in (2) over the weight \( w_{fg}(i, j) \)\}
26: \( S_f(k, \ell) \leftarrow S_f(k, \ell) + w_{fg}(i, j) \cdot g((k - 1) \sigma_\lambda + i, (\ell - 1) \sigma_\lambda + j) \)
27: if \( \text{inp} = d_\lambda - 1 \sigma_\lambda^2 \) \{last input submap\} then
28:   write \( y_f(k, \ell) \leftarrow \text{ReLU}(S_f(k, \ell)) \) to DRAM \{writing \( m_\lambda \times n_\lambda \) outputs\}
29: else
30:   write \( S_f(k, \ell) \) to DRAM \{writing \( m_\lambda \times n_\lambda \) weighted sums\}
31: end if
32: end for
33: end for \{next shared weight\}
34: end for \{next feature map\}
35: end for \{next input submap\}
36: end for \{next input feature map\}
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 \( (k, \ell) \in g(i_0, j_0) \) (lines 6–8), \( m_\lambda n_\lambda \) accumulated weighted sums \( S_f(k, \ell) \) for all \( k \in \{1, \ldots, m_\lambda \} \) and \( \ell \in \{1, \ldots, n_\lambda \} \) (lines 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 \( (k, \ell) \in \bigcup_{i_0,j_0 \in \{1, \ldots, \sigma_\lambda \}} g(i_0, j_0) \) (line 7 within the nested loops 2–36, 3–35, and 6–8), as well as each weight \( w_{fg}(i, j) \) for \( g \in \mathcal{N}_{\lambda-1}, f \in \mathcal{N}_\lambda, i_0, j_0 \in \{1, \ldots, \sigma_\lambda \} \), 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

\[
E_{\text{inputs}}^\lambda = b d_{\lambda-1} m_{\lambda-1} n_{\lambda-1}
\]

and (12), respectively.

Any accumulated weighted sum \( S_f(k, \ell) \) for \( f \in \mathcal{N}_\lambda, k \in \{1, \ldots, m_\lambda \}, \) and \( \ell \in \{1, \ldots, n_\lambda \} \), 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 \( g \in \mathcal{N}_{\lambda-1} \) in layer \( \lambda - 1 \) and each of the \( \sigma_\lambda^2 \) 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

\[
E_{\text{outputs}}^\lambda = b \left( 2d_{\lambda-1}\sigma_\lambda^2 - 1 \right) d_{\lambda} m_{\lambda} n_{\lambda}.
\]

Hence, this dataflow provides another upper bound on the data energy of layer \( \lambda \):

\[
E_{\text{data}}^\lambda \leq b \left( d_{\lambda-1} m_{\lambda-1} n_{\lambda-1} + (2d_{\lambda-1}\sigma_\lambda^2 - 1) d_{\lambda} m_{\lambda} n_{\lambda} + d_{\lambda} (d_{\lambda-1} r_{\lambda} s_{\lambda} + 1) \right)
\]
according to (7), (21), (22), and (12). This bound is comparable to (19) if the number of single neurons in layers $\lambda$ 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 $\sigma_{\lambda}^2m_{\lambda}n_{\lambda} \approx m_{\lambda-1}n_{\lambda-1}$ according to (1). Nevertheless, the multiplicative constant 2 of leading term $d_{\lambda}d_{\lambda-1}m_{\lambda-1}n_{\lambda-1}$ in (23), which is caused by storing partially evaluated weighted sums in DRAM, makes the upper bound (19) more tight than (23).

6. Experimental Validation

In this section, we compare the theoretical energy complexity introduced in Section 3 to the real power consumption estimated by the Timeloop/Accelryg 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 optimal in terms of power consumption estimated by Accelryg (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. 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\footnote{https://github.com/PetraVidnerova/timeloop-accelergy-test}.

The computation energy reported by Timeloop/Accelryg corresponds directly to the number of MACs calculated in (6) for $E_{\text{comp}}^\lambda$ 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 $\lambda \in \Gamma$, we measure empirical dependencies of the optimal data energy first separately on its depth $d_{\lambda}$, input feature map size $m_{\lambda-1} = n_{\lambda-1}$, kernel size $r_{\lambda} = s_{\lambda}$, and stride $\sigma_{\lambda}$, which is minimized by using the Timeloop/Accelryg framework for the Simba and Eyeriss architectures. These dependencies are then compared to the corresponding upper bound (19) on $E_{\text{data}}^\lambda$ achieved by the dataflows in the energy complexity model as presented in Section 5. This theoretical upper bound assumes a sufficient
Table 1: Required buffer capacities for AlexNet convolutional layers in kilobytes.

<table>
<thead>
<tr>
<th>( \lambda )</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>( m_\lambda = n_\lambda )</td>
<td>55</td>
<td>27</td>
<td>13</td>
<td>13</td>
<td>13</td>
</tr>
<tr>
<td>( d_\lambda )</td>
<td>64</td>
<td>192</td>
<td>384</td>
<td>256</td>
<td>256</td>
</tr>
<tr>
<td>( r_\lambda = s_\lambda )</td>
<td>11</td>
<td>5</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>( \sigma_\lambda )</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

(13): \( 2m^2_\lambda + 1 \)

\[
\begin{array}{cccccc}
\text{b = 8 bits} & 5.91 \text{kB} & 1.42 \text{kB} & 0.33 \text{kB} & 0.33 \text{kB} & 0.33 \text{kB} \\
\text{b = 16 bits} & 11.82 \text{kB} & 2.85 \text{kB} & 0.66 \text{kB} & 0.66 \text{kB} & 0.66 \text{kB} \\
\text{b = 32 bits} & 23.64 \text{kB} & 5.7 \text{kB} & 1.32 \text{kB} & 1.32 \text{kB} & 1.32 \text{kB} \\
\end{array}
\]

(20): \( m^2_\lambda + r^2_\lambda + 1 \)

\[
\begin{array}{cccccc}
\text{b = 8 bits} & 3.07 \text{kB} & 0.74 \text{kB} & 0.17 \text{kB} & 0.17 \text{kB} & 0.17 \text{kB} \\
\text{b = 16 bits} & 6.15 \text{kB} & 1.47 \text{kB} & 0.35 \text{kB} & 0.35 \text{kB} & 0.35 \text{kB} \\
\text{b = 32 bits} & 12.29 \text{kB} & 2.95 \text{kB} & 0.7 \text{kB} & 0.7 \text{kB} & 0.7 \text{kB} \\
\end{array}
\]

Buffer capacity satisfying (13) or (20). Table 1 shows required Buffer capacities for the AlexNet convolutional layers (Krizhevsky et al., 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).

In particular, for the comparison of empirical energy consumptions to the theoretical data energy \( E^\lambda_{\text{data}} \), we use the following asymptotic upper bounds:

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

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 (24) 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 \( \lambda \in \Gamma \) with increasing parameters \( d_\lambda, m_{\lambda-1} = n_{\lambda-1}, r_\lambda = s_\lambda, \) and \( \sigma_\lambda \), each separately. Namely, the values for \( d_\lambda \) (or \( d_{\lambda-1} \) for which the results omitted in Figure 4 were similar) and \( m_{\lambda-1} \)
Figure 4: The data energy estimates by Timeloop/Accelergy (displayed by bars) for convolutional layer $\lambda$ with increasing parameters $d_\lambda$, $m_{\lambda-1}$, $r_\lambda$, and $\sigma_\lambda$, each separately (from top to bottom), on the Simba (left) and Eyeriss (right) architectures, which fit the asymptotic trends (24) in the energy complexity model (dashed lines).
were taken from the interval 8 to 512 and 56 to 448, respectively, with the step 8, while for $r_\lambda$ and $\sigma_\lambda$ 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 (24), 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 $E_{\text{data}}^\lambda$ (dashed lines) fits energy estimates by Timeloop/Accelergy (displayed by bars), which confirms the asymptotic trends (24) 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 $d_\lambda$, $m_{\lambda - 1}$, $r_\lambda$, and $\sigma^{-1}_\lambda$, respectively. These statistical tests have approved the linearity in $d_\lambda$ ($p$-value 0.556 accepting the null hypothesis of $a = 0$) and the quadraticity in $m_{\lambda - 1}$ $r_\lambda$, and $\sigma^{-1}_\lambda$ ($p$-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 $d_\lambda$, $d_{\lambda - 1}$, $m_{\lambda - 1}$, $r_\lambda$, and $\sigma_\lambda$ of convolutional layer $\lambda \in \Gamma$ are changed simultaneously, in order to fit the empirical optimal data energy to its theoretical asymptotic lower bound (11):

$$E_{\text{data}}^\lambda = \Omega \left( d_\lambda d_{\lambda - 1} m_{\lambda - 1}^2 r_{\lambda}^2 \sigma_{\lambda}^{-2} \right).$$

For this purpose we employ 3000 randomly chosen convolutional layers whose parameters $d_\lambda$, $d_{\lambda - 1}$, $m_{\lambda - 1}$, $r_\lambda$, and $\sigma_\lambda$ are sampled from the intervals 8 to 512, 8 to 512, 56 to 448, 3 to 15, and, 1 to 5, respectively. The points in Figure 5.a show power consumption of these convolutional layers on the Simba architecture estimated by the Timeloop/Accelergy program, depending on the product $d_\lambda d_{\lambda - 1} m_{\lambda - 1}^2 r_{\lambda}^2 \sigma_{\lambda}^{-2}$ from (25).

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 $d_\lambda d_{\lambda - 1} m_{\lambda - 1}^2 r_{\lambda}^2 \sigma_{\lambda}^{-2}$. This is due to specific properties of the used Simba hardware architecture (e.g. power 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
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 $\lambda$ on the Simba architecture (a), whose fluctuations are filtered out by selecting the energy minima over each 14 consecutive values of the independent variable (points in boldface), which fit the asymptotic lower bound (25) on $E^\lambda_{\text{data}}$ (dashed line) in the energy complexity model (b).
compare the empirical optimal data energy to its theoretical lower bound (25), we filter out these fluctuations by selecting the points with the minimum energy over each 14 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 14 was chosen as a compromise between a sufficient number of sampled points and outliers, yet the results for values of 10 to 15 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 (25), 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 $E_{\text{data}}^{\lambda}$ (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 0.486 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 power consumption estimated by the Timeloop/Accelergy program for the Simba and Eyeriss hardware platforms.

7. 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. 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 power consumption estimated by the Timeloop/Accelergy program for CNNs on the Simba and Eyeriss hardware platforms.
In future research we plan to prove optimal bounds on the data energy for 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 and Cabessa, 2023). The proposed model thus allows to determine the principal asymptotic limits to which heuristic optimizers e.g. based on evolutionary algorithms (Kao and Krishna, 2020) can reach.

Acknowledgements

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


