HAL
open science

# Energy-Accuracy Scalable Deep Convolutional Neural Networks: A Pareto Analysis

Valentino Peluso, Andrea Calimera

## To cite this version:

Valentino Peluso, Andrea Calimera. Energy-Accuracy Scalable Deep Convolutional Neural Networks: A Pareto Analysis. 26th IFIP/IEEE International Conference on Very Large Scale Integration - System on a Chip (VLSI-SoC), Oct 2018, Verona, Italy. pp.107-127, 10.1007/978-3-030-23425-6_6 . hal-02321763

**HAL Id: hal-02321763**

**https://inria.hal.science/hal-02321763**

Submitted on 21 Oct 2019

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Distributed under a Creative Commons Attribution 4.0 International License

# Energy-Accuracy Scalable Deep Convolutional Neural Networks: A Pareto Analysis

Valentino Peluso and Andrea Calimera

Department of Control and Computer Engineering,
Politecnico di Torino, 10129 Torino
{valentino.peluso; andrea.calimera}@polito.it

**Abstract.** This work deals with the optimization of Deep Convolutional Neural Networks (ConvNets). It elaborates on the concept of *Adaptive Energy-Accuracy Scaling* through *multi-precision* arithmetic, a solution that allows ConvNets to be adapted at run-time and meet different energy budgets and accuracy constraints. The strategy is particularly suited for embedded applications made run at the "edge" on resource-constrained platforms. After the very basics that distinguish the proposed adaptive strategy, the paper recalls the software-to-hardware vertical implementation of precision scalable arithmetic for ConvNets, then it focuses on the energy-driven per-layer precision assignment problem describing a meta-heuristic that searches for the most suited representation of both weights and activations of the neural network. The same heuristic is then used to explore the optimal trade-off providing the Pareto points in the energy-accuracy space. Experiments conducted on three different ConvNets deployed in real-life applications, i.e. Image Classification, Keyword Spotting, and Facial Expression Recognition, show adaptive ConvNets reach better energy-accuracy trade-off w.r.t. conventional static fixed-point quantization methods.

## 1 Introduction

Deep Neural Networks (DNNs) are computational models that emulate the activity of the human brain during pattern recognition. They consist of deep chains of neural layers that apply non-linear transformations on the input data [1]. The projection on the new feature-space enables a more efficient classification, achieving accuracies that are close, and in some cases even above, those scored by the human brain. Convolutional Neural Networks [2] (ConvNets hereafter) are the first example of DNNs applied to problems of human-level complexity. They have brought about breakthroughs in computer vision [3] and voice recognition [4], improving the state-of-the-art in many application domains. From a practical viewpoint, the forward pass through a ConvNet is nothing more than matrix multiplications between pre-trained parameters (the synaptic weights of the hidden neurons) and the input data.
The most common use-case for ConvNets is image classification where a multi-channel image (e.g. RGB) is processed producing as output the probability that

the subject depicted in the picture belongs to a specific class of objects or concepts (e.g. car, dog, airplane, etc.). One can see this end-to-end inference process as a kind of data compression: high-volume raw-data (the pixels of the image) are compressed into a highly informative tag (the resulting class). In this regard, the adoption on the Internet-of-Things (IoT) is disruptive: distributed smart-objects with embedded ConvNets may implement data-analytics at the edge, near the source of data [5], with advantages in terms of predictability of the service response time, energy efficiency, privacy and, in general, scalability of the IoT infrastructure.

The design of embedded ConvNets encompasses a training stage during which the synaptic weights of the hidden neurons are learned using a back-propagation algorithm (e.g. the Stochastic Gradient Descent [6]). The learning is supervised and accuracy-driven, namely, it adjusts the weights such that an accuracy loss function evaluated over a set of labeled samples is minimized. Once trained, the ConvNet can be flashed on the smart-object and deployed at the edge, where it runs inference on never occurred samples. To notice that ConvNets presented in the literature show different depth (number of layers) and size (number of neurons per layer); also the topology may change due to optional layers used to reduce the cardinality of the intermediate activations, e.g., local pooling layers, or their sparsity, e.g. Rectified Linear Units (ReLU). Regardless of the internal structure, ConvNets show a common characteristic, complexity. Even the most simple model, e.g. AlexNet [2] or the more compact MobileNets [7], show millions of synaptic weights to be stored and tens of thousands of matrix convolutions to be run [5]. This prevents their use on low-power embedded platforms which offer low storage capacity, low compute power, and limited energy budget. How to design ConvNets that fit the stringent resource constraints while preserving classification accuracy is the new challenge indeed.

Recent works introduced several optimization strategies, both at the software level and hardware level [8]. They mainly exploit the intrinsic redundancy of ConvNets in order to reduce (*i*) the number of weights/neurons (the so-called *pruning* methods) or (*ii*) the arithmetic precision (*quantization* methods) or (*iii*) both [9]. Precision scaling is of practical interest due to its simplicity and the solid theories developed in the past for DSP applications. It concurrently reduces the memory footprint (the lower the bit-width, the lower the memory footprint) and the execution latency (the lower the bit-width, the faster the execution). The use of fixed-point arithmetic with 16- and 8-bit [10] instead of the 32-bit floating-point, or even below, e.g. 6 and 4-bit [11], has shown remarkable savings with no, or very marginal accuracy drop. Aggressive binarization [12] is an alternative approach provided that large accuracy loss is acceptable. Obviously, the implementation of quantized ConvNets asks for integer units that can process data with reduced representations; recent hardware designs, both from industry and academia, follow this trend [13][14][15].

Most of the existing optimizations, both pruning and quantization, were originally conceived as static methods. Let's consider quantization. For a given ConvNet the numeric precision of the weights is defined at design-time and then

kept constant during run-time. Therefore, the design effort is that of finding the proper bit-width such that accuracy losses are minimized [16]. Although effective, this approach is very conservative as inference always operates at full speed and hence under maximum resource usage. Adaptive strategies that speculate on the quality of results to reach higher energy efficiency are a more interesting option for portable devices deployed on non-critical missions [17]. There exist applications or use-cases for which the classification accuracy can be relaxed without affecting much the user perception, or, alternatively, conditions under which other extra-functional properties of the system, e.g. energy budget or latency, get higher priority. For such cases, one may use the arithmetic precision as a control knob to manage the resources. This concept of energy-accuracy scaling is a well-established technique for VLSI designs [18], while it represents a less explored option for ConvNets (and DNNs in general).

The idea of energy-accuracy scalable ConvNets through dynamic precision scaling was first introduced in [19] and then elaborated in [20] with the introduction of an energy-driven optimization framework. The method applies to software-programmable arithmetic accelerators where precision scaling is achieved through variable-latency Multiply&Accumulate (MAC) instructions. This implementation applies for any general purposes MCUs (e.g. [21]) or application-specific processors with a multi-precision instruction-set (e.g. Google TPU [22]); it can also be extended to dedicated architecture (both ASIC or FPGA accelerators [23]). This chapter further investigates on this strategy introducing a Pareto analysis of the energy-accuracy space. An optimization engine is used to identify the arithmetic precision that minimizes energy and accuracy loss concurrently. The obtained precision settings can be loaded at run-time with minimal overhead thus to allow ConvNets to reach the operating conditions that satisfy the requirements imposed at the system-level. As test-benches we used three real-life applications built upon state-of-the-art ConvNets, i.e. Image Classification [24], Keyword Spotting [25], and Facial Expression Recognition [26]. Experimental results suggest the proposed strategy is a practical solution for the development of flexible, yet efficient IoT applications.

The remaining sections are organized as follows. Section 2 gives an overview of related works in the field. Section 3 describes the implementation details for the single weight-set multi-precision arithmetic used in scalable ConvNets. Section 4 recalls the optimization engine and the energy-accuracy models adopted. Finally, Section 5 shows the Pareto analysis over the three benchmarks and the performance of the optimization heuristic.

## 2 Related Works

With the emerging of the edge-computing paradigm, the reduction of ConvNets complexity has become the new challenge for the IoT segment. The problem is being addressed from different perspectives: with the design of custom hardware that improves the execution of data-intensive loops achieving energy efficiencies of few pico-Joules/operation[27] [11]; with new learning strategies that generate

less complex networks [28]; with iso-accuracy compression techniques aimed at squeezing the model complexity. A thorough review is reported in [29]. To notice that while many existing techniques are conceived as static methods, the dynamic management of ConvNets is a less explored field. This work deals with this latter aspect.

### 2.1 Adaptive ConvNets

Following the recent literature, the concept of adaptive ConvNets may have multiple interpretations and hence different implementations. On the one hand, there are solutions that adapt to the complexity of the input data. On the other hand, solutions that adapt to external conditions or triggers, regardless of data complexity.

The former class is mainly represented by techniques that implement the general principle of coarse-to-fine computation [30]. These methods make use of branches in the internal network topology generating *conditional* deep neural nets [31]. In its most simple implementation, a conditional ConvNet is made up of a chain of two classifiers, a *coarse* classifier (for "easy" inputs) and a *fine* classifier (for "hard" inputs) [32]; the *coarse* classifier is always-on, while the *fine* classifier is occasionally activated for "hard" inputs (statistically less frequent). As a result, ConvNets can adapt to the complexity of data at run-time. An extension with deeper chains of quantized micro-classifiers is proposed in [33], while in [34] authors propose the use of Dynamic Voltage Accuracy Frequency Scaling (DVAFS) for the recognition of objects of different complexity.

Concerning the second class, that is the main target of this work, adaptivity is achieved by tuning the computational effort of the ConvNet depending on the desired accuracy. The control knob is the arithmetic precision of the convolutional layers. The work described in [19] is along this direction as it introduces an HW-SW co-design to implement multi-precision arithmetic at run-time. Depending on the parallelism of the HW integer units (e.g. 16- or 8-bits), weights can be loaded and processed using different bit-widths thus to achieve different degrees of accuracy under different energy budgets. This is the enabler for accuracy-energy scaling adaptive ConvNets. To notice that unlike static quantization methods where different accuracy levels could be achieved using multiple pre-trained weight-sets stored as separate entities, here the precision scaling is achieved using a single set of weights and incomplete arithmetic operations. The same strategy is adopted in this work.

Hybrid solutions may jointly exploit the complexity of the input problem with the accuracy imposed at the application level. For instance, the authors of [35] introduce the concept of multi-level classification where the classification task can be performed at different levels of semantic abstraction: the higher the abstraction, the easier the classification problem. Then, depending on the abstraction level and the desired accuracy, the ConvNet is tuned to achieve the maximum energy efficiency.

### 2.2 Fixed-Point Quantization

Since the multi-precision strategy adopted in this work encompasses the quantization to fixed-point, this subsection gives a brief taxonomy of the existing literature on the subject.
Complexity reduction through fixed-point quantization exploits the characteristics of the weight distributions across different convolutional layers in order to find the most efficient data representation [36]. Two main stages are involved: the definition of the *bit-width*, i.e. the data parallelism, and the *radix-point scaling*, i.e. the position of the radix point. A common practice is to define the bit-width depending on hardware availability (e.g. 16-, 8-bit for most of the architectures), then find the radix-point position that minimizes the quantization error. The existing techniques, mainly from the DSP theory, differ in the radix-point scaling scheme. A complete review is out of the scope of this work and the interested reader can refer to [8]. It is worth emphasizing that a one-size-fits-all solution does not exist as efficiency is affected by the kind of neural networks under analysis and the characteristics of the adopted hardware.
A more relevant discriminant factor is the spatial granularity at which the fixed-point format is applied, *per-net* or *per-layer*. In the former case all the layers share the same representation; in the latter case, each layer has its own representation. Since the weights distribution may substantially differ from layer to layer, a finer, i.e. per-layer, approach achieves lower accuracy loss [36].
Whatever the granularity is, existing works from the machine-learning community, e.g. [36] [37], focused on accuracy-driven optimal precision scaling. Only a few papers take hardware resources into account, which is paramount when dealing with embedded systems. The authors of [16] briefly describe a greedy approach where low precision is assigned starting from the first layer of the net (topological order) without considering the complexity of the layer. In [10] authors describe the design of embedded ConvNets for FPGAs and propose a per-layer precision scaling that is aware of the number of memory accesses. Only very few works, e.g. [29] and [20], bring energy consumption as a direct variable in the optimization loop.

## 3 Energy-Accuracy Scalable Convolution

The proposed adaptive ConvNet strategy leverages precision scalable arithmetic. This section introduces a possible implementation of matrix convolution using software-programmable multi-precision Multiply&Accumulate (MAC) instructions. It first describes the algorithmic details, then it presents a custom processing element that accelerates the variable-latency MAC with minimal design overhead.

### 3.1 SW: Multiprecision Convolution

For a given layer in a ConvNet, the convolution between the $M \times M$ input map matrix $\underline{\mathbf{I}}$ and the $M \times M$ weight matrix of a kernel $\underline{\mathbf{W}}$ is the dot-product of the

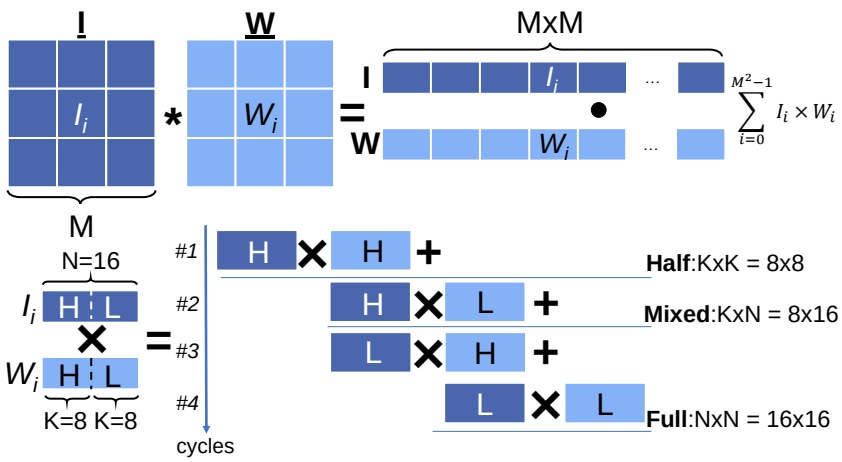


**Fig. 1:** Iterative multiply-accumulate algorithm.

two unrolled vectors $I$ and $W$ of length $(M \times M)$. The dot-product between $I$ and $W$ is the sum of the $(M \times M)$ products $I_i \times W_i$, as shown in fig. 1.

Assuming a $N$-bit fixed-point representation ($N = 16$ in this work), $I_i$ and $W_i$ can be seen as two concatenated halfwords of $K = N/2$ bits ($K$=8); the most significant parts $I_i^H$ and $W_i^H$ and the least significant parts $I_i^L$ and $W_i^L$. As pictorially described in fig. 1, each single product $I_i \times W_i$ is implemented by means of a four-cycles procedure where the most significant and least significant halfwords are iteratively multiplied, shifted and accumulated. To notice that $I_i^H$ and $W_i^H$ are signed integers, $I_i^L$ and $W_i^L$ are unsigned. Different precision options can be reached by stopping the execution at earlier cycles: *half* $(K \times K)$ 1 cycle, *mixed* $(K \times N)$ 2 cycles and *full* $(N \times N$-bit) 4 cycles; an additional *mixed* precision option $(N \times K)$ is also obtained by swapping the second and the third cycle (2 cycles).

The same four options can be extended to the dot-product procedure as described in Algorithm 1. At *half*-precision, both the operands $I_i$ and $W_i$ are reduced to $K$ bits. The first loop (lines 1-2) operates on the most significant parts $I_i^{\mathrm{H}}$, $W_i^{\mathrm{H}}$. The result is then returned (line 3). At *mixed*-precision, only one operand, the input $I_i$ (or the weight $W_i$, not shown in the pseudo-code), is reduced to $K$ bits. First, the partial result $r$ is shifted of $K$-bits (line 4), then the second loop (lines 5-6) iterates on $I_i^{\mathrm{H}}$ and $W_i^{\mathrm{L}}$ ($I_i^{\mathrm{L}}$ and $W_i^{\mathrm{H}}$) and the result is returned (line 7). At *full*-precision, both $W_i$ and $I_i$ are taken as $N$ bit operands. In this case the last two loops (lines 8-12) come into play and they iterate on the least significant parts $W_i^{\mathrm{L}}$ and $I_i$ (both $H$ and $L$) thus to complete the remaining part of the product. To summarize, with $N$=16, the available precision options are: *half* ($K \times K$, i.e. $8 \times 8$), *mixed* ($N \times K$, i.e. $16 \times 8$ or $K \times N$, i.e. $8 \times 16$), *full* ($N \times N$, i.e. $16 \times 16$). Given the regular structure of the algorithm, all them can be implemented on the same $K \times K$ MAC unit.

**Algorithm 1:** Iterative multiply-accumulate algorithm

**Input:** $I$, $W$, *precision*
**Output:** Dot-Product $r$

```
for i = 0; i < M; i = i + 1 do
    r = r + I_i^H × W_i^H
if (precision==half) then return r ;                    // half:KxK
r = r ≪ K
for i = 0; i < M; i = i + 1 do
    r = r + I_i^H × W_i^L
if (precision==mixed) then return r ;                   // mixed:KxN
for i = 0; i < M; i = i + 1 do
    r = r + I_i^L × W_i^H
r = r ≪ K
for i = 0; i < M; i = i + 1 do
    r = r + I_i^L × W_i^L
return r ;                                              // full:NxN
```

This straightforward algorithm offers a simple way to adjust the precision of the results and the resource usage. Firstly, it allows the computational effort, and hence the energy consumption, to scale with the arithmetic precision; secondly, it alleviates the memory bandwidth as less bits need to be moved from/to the memory banks at lower precisions[1].

### 3.2 HW: Variable-latency Processing Element

Figure 2 gives the RTL-view of the proposed processing element (PE) for $N$=16. The PE is composed by $9 \times 9$ multiplier, where the $9^{th}$ bit is used for the sign extension of the operands. As described in the previous subsection, the most significant parts ($I_i^{\mathrm{H}}$, $W_i^{\mathrm{H}}$) are signed, while the least significant parts ($I_i^{\mathrm{L}}$, $W_i^{\mathrm{L}}$) are unsigned. Therefore, the MSB of ($I_i^{\mathrm{L}}$, $W_i^{\mathrm{L}}$) belongs to the module, while that of ($I_i^{\mathrm{H}}$, $W_i^{\mathrm{H}}$) is the sign. In order to account for this issue we implemented the following mechanism: when ($I_i^{\mathrm{H}}$, $W_i^{\mathrm{H}}$) are processed, the sign is extended to the $9^{th}$ by concatenating the MSB (i.e. the sign) of $I$ and $W$; when ($I_i^{\mathrm{L}}$, $W_i^{\mathrm{L}}$) are processed a 0 is concatenated. The selection is done through the control signals *signed-I* and *signed-W* driven by the local control unit (omitted in the picture for the sake of space). The same control unit is in charge of feeding the MAC with the right sequence of data ($H$ or $L$) fetched from a local memory.
The accumulator has 16 guard bits and an embedded saturation logic to handle underflow and overflow. The role of the programmable shifter is two-fold. First, to shift the partial results when needed (see Algorithm 1). Second, to implement the dynamic fixed point arithmetic by moving the radix point of the final accumulation result depending on the desired fractional length [39]. A *range check* logic drives bit saturation if the result does not fit the word-length.

[1] We assume the availability of memories that support both word ($N$-bit) and halfword ($K$-bit) accesses [38].

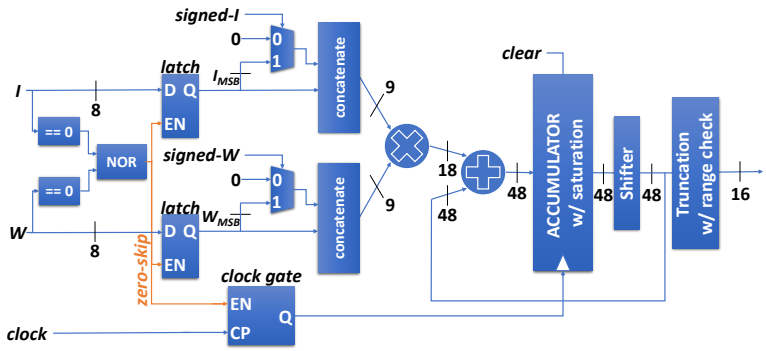


**Fig. 2:** $8 \times 8$ HW unit for multi-precision MAC.

In order to minimize the dynamic power consumption, a *zero-skipping* strategy [34] is implemented by means of latch-based operand isolation and clock-gating. If one of the operands is zero, then the latches prevent the propagation of inputs minimizing the switching activity, while the clock-gating cell disables the clock signal thus reducing the equivalent load capacitance of the clock signal.

### 3.3 Hardware Characterization

The proposed SW-HW precision scaling strategy can be implemented using both FPGA and ASIC technologies. In this work we designed and characterized the $8 \times 8$ MAC unit using a commercial 28 nm UTBB FDSOI technology and the Synopsys Galaxy Platform, versions L-2016.03. The frequency constraint is set to 1 GHz at 0.90 V in a typical process corner (compliant with recent works that used the same technology [40]). Power consumption is extracted using Synopsys PrimeTime L-2016.06 with SAIF back-annotation. Collected results show a standard cell area of 1443 $\mu m^2$ and total average power consumption of 0.95 mW. Compared to a traditional $8 \times 8$ MAC unit, the proposed architecture shows 3.7% area penalty.

Table 1 shows the latency ($N_{\text{cycles}}$) and the energy consumption per MAC operation ($E_{\text{MAC}}$) for the four precisions available. As one can see, each row in the table corresponds to a different implementation point in the precision-energy space. If one of the two operands is zero, energy $E_{\text{zero}}$ reduces substantially due to the zero-skipping logic: $E_{\text{zero}} = 0.103 E_{\text{MAC}}$.

**Table 1:** Energy/MAC vs Precision

| Precision ($I \times W$) | $N_{\text{cycles}}$ | $E_{\text{MAC}}$ (pJ) |
|---|---|---|
| $16 \times 16$ | 4 | 3.80 |
| $16 \times 8$ | 2 | 1.90 |
| $8 \times 16$ | 2 | 1.90 |
| $8 \times 8$ | 1 | 0.95 |

# 4 Energy-Driven Precision Assignment

## 4.1 Fixed-Point Quantization

The shift from floating-point to fixed-point is a well-known problem in the DSP domain. In this sub-section, we review the basic theory and the main aspects involving this work.
A floating-point value $V$ can be represented with a binary word $Q$ of $N$ bits using the following mapping function:

$$V = Q \cdot 2^{-FL} \tag{1}$$

$FL$ indicates the fraction length, i.e. the position of the radix-point in $Q$. Given a set of real values, the choice of $N$ and $FL$ affects the information loss due to quantization. Since the bit-width $N$ is usually given as a design constraint (e.g. 16-bit in this work), the problem reduces to searching the optimal $FL$ (the integer length $IL$ is then given by $N$-$IL$). The choice of $FL$ affects the maximum representable value $|V_{\max}|$ and the minimum quantization error $Q_{\text{step}}$. Concerning $|V_{\max}|$, the relationship is described in the following equation:

$$FL = \left\lfloor \log_2 \left( \frac{2^{BW-1} - 1}{|V_{\max}|} \right) \right\rfloor \tag{2}$$

A trade-off does exit: the lower the $FL$ the lower the $V_{\max}$; the larger the $FL$ the lower the $Q_{\text{step}}$. The decision of which constraint to guard more ($|V_{\max}|$ or $Q_{\text{step}}$) mainly depends on the distribution of the original floating-point weights and their importance in the neural model under quantization.
A dynamic fixed-point scheme is implemented where the fraction length is defined layer-by-layer. The $FL_{\text{opt}}$ that minimizes the L2 distance between the original 32-bit floating point values and the quantized values is searched among $N$-1 possible values. The search is done over a calibration set built by randomly picking 100 samples from the training set. To be noted that our problem formulation applies a symmetric linear quantization using a binary radix-point scaling. As an additional piece of information, it is important to underline that quantization is not followed by retraining, a very time-consuming procedure even for small ConvNets.

## 4.2 Multiprecision Fixed-Point ConvNets

**Problem Formulation** For a ConvNet of $L$ layers, the classification accuracy can be scaled to different values by optimally selecting the arithmetic precision of each layer. The choice of such optimal precision should be done for the *input map* ($\underline{\mathbf{I}}$) and the *weight* ($\underline{\mathbf{W}}$) matrices of each layer each layer $i$, and for the *output map* matrix ($\underline{\mathbf{O}}$) of the last layer[2].

[2] The precision of $\underline{\mathbf{O}}$ does not impact computation as it only affects the number of memory accesses.

Assuming the availability of the four accuracy options described in Section 3, i.e. *full* (16 × 16), *mixed* (16 × 8 or 8 × 16), *half* (8 × 8), the precision for **I** and **W** of each layer, and that of **O** for last layer, can be assigned to 8-bit or 16-bit. We encode the unknown of the problem as a vector $X$ of (2×L+1) Boolean variables $x_i$, where the variable $x_{2\times L+1}$ refers to **O**. The encoding map is: $x$=0→8-bit, $x$=1→16-bit. The optimal assignment is the one that minimizes the total energy consumption $E(X)$ while ensuring an accuracy loss $\lambda(X)$ lower than a user-defined constraint $\lambda_{\max}$.

**Energy-Driven Precision Assignment** The optimal precision assignment to each layer is carried out using a custom meta-heuristic based on Simulated Annealing (SA). Algorithm 2 shows the pseudo-code of the SA. It gets as inputs the parameters listed in table 2.

**Table 2:** Simulated Annealing Hyper-parameters

| | |
|---|---|
| $T_0$ | Initial temperature |
| $T_{\mathrm{f}}$ | Final temperature |
| $X_0$ | Starting solution |
| *cooling* | Temperature derating factor (*geometric* in our case) |
| $K_{\mathrm{b}}$ | Normalization factor of the acceptance probability |
| *iter* | Number of iterations for each temperature $T$ |
| $\lambda_{\max}$ | User-defined accuracy drop (percentage) |
| *cal_set* | Calibration set size |

In all the experiments, the starting solution $X_0$ is the full-precision (16-bit) to all the $L$ layers (both **I** and **W**, and **O**. The estimation of the accuracy drop is done on a subset of images randomly picked from the training set, referred to as the calibration set. Its size is defined by the *cal_set* parameter.

At each iteration, the next state is generated as a random perturbation of the current state (line 6). For those states that satisfy the accuracy constraint (line 7), the energy cost function $E$ is evaluated (line 8) through the function *energy*. If $\Delta$E (line 9) reduces (line 10), the new state is accepted (lines 11-12). If not, the new state is accepted following a Boltzmann probability function (lines 10-12); the acceptance ratio gets smaller as $T$ reduces. States that show minimum energy are iteratively saved as best solutions (lines 13-15). Once the total number of iterations is reached (line 5), the temperature $T$ is cooled down (line 17). The process iterates till the minimum temperature $T_{\mathrm{f}}$ is reached (line 4).

The bottleneck of the algorithm is the call to the function *accuracy_drop*. For this reason, the algorithm takes trace of already processed states; this information is fed to the *accuracy_drop* function which can eventually by-pass accuracy estimation (line 16).

**Algorithm 2:** Simulated Annealing

```
Input: T_0, T_f, X_0, cooling, K_b, iter, λ_t extrmmax, cal_set
Output: X
T = T_0
E = energy(X_0)
E_max = energy (ones (2L + 1)); E_min = energy (zeros (2L + 1))
while (T ≥ T_f) do
    for i = 0; i < iter; i = i+1 do
        next_state = move (current_state)
        if accuracy_drop(next_state, cal_set, tested) < λ_max then
            E_next = energy (next_state)
            ΔE = (E_next-E_current) / (E_max - E_min)
            if (dE < 0) or (exp[-ΔE/K_b·T] > random(0, 1)) then
                current_state = next_state
                E_current = E_new
                if E_current < E_best then
                    E_best = E_current
                    best_state = current_state
        update(tested)
    T = T · cooling
return best_state
```

**Energy** The system-level architecture depicted in Figure 3 serves as a general template to describe Application-Specific Processors for ConvNets computing, e.g. [11]. It consists of a planar array of processing elements (PE), in our case the MAC units described in Section 3, a set of SRAM buffers for storing temporal data (Input Buffer, Weight Buffer, and Output Buffer), an off-chip memory (DRAM) and its DMA, a control unit (RISC) that schedules the operations.

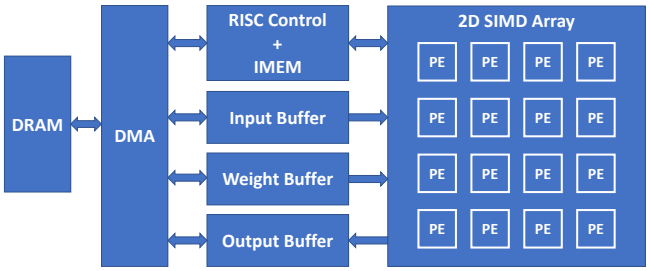


**Fig. 3:** Architectural template of ConvNet accelerators.

The total energy consumption $E$ is the sum of two main contributions: $E = E^{\text{comp}} + E^{\text{mem}}$. $E^{\text{comp}}$ is the energy consumed by the PE array, $E^{\text{mem}}$ is the energy consumed due to data movement through the memory hierarchy.

The first term is defined as:

$$E^{\text{comp}} = \sum_{i=1}^{L} E^{\text{MAC}} \cdot N_{\text{cycles}}(x_i) \cdot N_i^{\text{MAC}} + E^{\text{zero}} \cdot N_i^{\text{zero}} \tag{3}$$

$L$ is the number of layers of the ConvNet. $E^{\text{MAC}}$ is the energy consumption of the *half*-precision MAC (row 8×8 in table 1). $N_{\text{cycles}}$ is the latency of a single MAC operation of the $i$-th layer; it is given as multiple of the latency of the *half*-precision MAC (row 8×8 in Table 1) and it is function of the precision $x_i$. $N_i^{\text{MAC}}$ is the number of non-zero MAC operations of the $i$-th layer. $E_i^{\text{zero}}$ is the energy consumed under zero-skipping (mostly due to leakage). $N_i^{\text{zero}}$ is the number of zero MAC.
The second term is defined as:

$$E^{\text{mem}} = \sum_{i=1}^{2 \cdot L+1} E^{\text{MAC}} \cdot [\alpha_i(x_i) + \beta_i(x_i) + \gamma_i(x_i)] \tag{4}$$

$E^{\text{MAC}}$ is the same as in Equation 3, while $\alpha_i$, $\beta_i$ and $\gamma_i$ are three parameters that describe the energy consumed by the $i$-th layer due to reading/writing the *input map* ($\alpha_i$), the *weights* ($\beta_i$), the *output map* ($\gamma_i$). More specifically they represent the ratio between the energy consumption of the memory and the energy consumption of the PE array; here again, the energy unit is the *half*-precision MAC (row 8×8 in table 1) [11]. Obviously, $\alpha$ and $\beta$ do not contribute for the final output layer: $\alpha_{L+1}$=0 and $\beta_{L+1}$=0.
All the three parameters are function of the layer precision $x_i$: both fetch and write-back operations depend on ($i$) the accuracy of the MAC algorithm, and ($ii$) the number of zero-multiplications (switching activity to/from memory may change substantially). Moreover $\alpha_i$, $\beta_i$, $\gamma_i$ change depending on the ConvNet model: number and size of weights/channels per layer, stride and padding. Finally, they also differ depending on the size of the hardware components (PE array, and global buffers). Since the target of this work is not the energy model per se, not even the evaluation of different architectural solutions, $\alpha_i$, $\beta_i$, $\gamma_i$ are extracted for the architecture proposed in [11] and then scaled to our precision reduction strategy. The same $E^{\text{mem}}$ model applies to different architectures by proper tuning of the three parameters.

**Accuracy Drop** The accuracy drop is computed as the ratio between the number of miss-classified images and the total number of images in the calibration set (*cal_set*), hence its estimation implies the execution of $\mathcal{S}$ feed-forward inferences using the quantized fixed-point model ($\mathcal{S}$ as the cardinality of *cal_set*).
Unfortunately, common GPUs do not have integer units. To address this issue we implemented the *fake* quantization proposed in [37]. It is a SW strategy that emulates the loss of information due to fixed-point arithmetic still using floating-point data-type. Each layer is wrapped with a software module that converts its input data and weights (32-bit floating-point) into a *fake* integer, namely, still a 32-bit floating-point number subtracted of an amount equal to the error that the fixed-point representation would have brought. The advantage is that all the fixed-point operations are physically run by the high-performance FP units.

# 5 Results

## 5.1 Experimental Set-up

The objective of this work is to provide a Pareto analysis of adaptive ConvNets implemented with the proposed energy-accuracy scaling strategy. As benchmarks we adopted three different applications which are reaching widespread use in several domains: Image Classification (IC), Keyword Spotting (KWS), Facial Expression Recognition (FER). Additional details provided in the next subsection. The exploration in the energy-accuracy space is conducted using the SA engine introduced in Section 4. More specifically, the algorithm is made run under different accuracy loss constraints, from 1% to 15% with step 1%, and collecting the energy consumption reached by the optimal precision settings.
Table 3 summarizes the SA parameters used in the experiments. For all the networks we selected the same hyper-parameters, except for the number of iterations *iter* at a given temperature $T$. As described in the next sub-section, the three ConvNets have different number of layers, hence different complexity; as the cardinality of the search space increases, more iterations are needed to explore the cost function.

**Table 3:** Simulated Annealing Hyper-parameters values

| | **IC** | **KWS** | **FER** |
|---|---|---|---|
| $T_0$ | 512 | | |
| $T_f$ | 2.5 | | |
| $K_\mathrm{b}$ | 1e-2 | | |
| *cooling* | 2.5 | | |
| *iter* | 10 | $10^2$ | $10^3$ |
| $\lambda_\mathrm{max}$ | 1% – 15%, step 1% | | |
| *cal_set* | 2000 | | |

## 5.2 Benchmarks

The three benchmarks under analysis serve very different purposes; their functionality and main characteristics, as well as their training set, are described separately therefore.
**Image Classification (IC):** the typical image recognition on the popular *CIFAR-10* dataset. The dataset collects 60000 $32 \times 32$ RGB images [24] evenly split in 10 classes, with 50000 and 10000 samples for the train-set and test-set respectively. The adopted ConvNet is taken from the Caffe framework [41], which consists of three convolutional layers interleaved with max-pooling and one fully-connected layer.

**Table 4:** Benchmarks overview. Considering that each convolutional layer with shape $(ch, k_h, k_w)$, fully-connected with shape $(ch)$, and max-pooling layer with shape $(k_h, k_w)$. Where $k_h$ and $k_w$ are respectively the kernel height and width, while $ch$ denotes the number of output channels.

| Application | **IC** | **KWS** | **FER** |
|---|---|---|---|
| Dataset | CIFAR-10 [24] | Speech Commands [25] | FER2013 [26] |
| Input Shape | $3 \times 32 \times 32$ | $1 \times 32 \times 40$ | $1 \times 48 \times 48$ |
| Model Architecture | Conv2d (32,5,5)<br>MaxPool2d (3,3)<br>Conv2d (32,5,5)<br>MaxPool2d (3,3)<br>Conv2d (64,5,5)<br>MaxPool2d (3,3)<br>Linear (10) | Conv2d (186,32,8)<br>MaxPool2d (1,1)<br>Conv2d (64,10,4)<br>Linear (32)<br>Linear (128)<br>Linear (128)<br>Linear (12) | Conv2d (32,3,3)<br>Conv2d (32,3,3)<br>Conv2d (32,3,3)<br>MaxPool2d (2,2)<br>Conv2d (64,3,3)<br>Conv2d (64,3,3)<br>Conv2d (64,3,3)<br>MaxPool2d (2,2)<br>Conv2d (128,3,3)<br>Conv2d (128,3,3)<br>Conv2d (128,3,3)<br>MaxPool2d (2,2)<br>Linear (7) |
| Top-1 Acc. | 83.04% | 80.94% | 65.67% |
| #MACs | 12 298 240 | 504 128 | 149 331 456 |
| #Op. Points | 512 | 8 192 | 2 097 152 |

**Keyword Spotting (KWS):** a standard problem in the field of speech recognition. We considered a simplified version of the problem[3]. The reference dataset is the Speech Commands Dataset [25]; it counts of $65k$ 1 s-long audio samples collected during the repetition of 30 different words by thousands of different people. The goal is to recognize 10 specific keywords, i.e. "Yes", "No", "Up", "Down","Left", "Right", "On", "Off", "Stop", "Go", out of the 30 available words; samples that do not fall in these 10 categories are labeled as "unknown". There is also an additional "silence" class made up of background noise samples (pink noise, white noise, and human-made sounds). The training set and test set collect 56196 and 7518, respectively. The adopted ConvNet is the *cnn-one-fstride4* described in [42]; it has two convolutional layers, one max-pooling layer and four fully-connected layers. The ConvNet is fed with the spectrogram of the recorded signal which is obtained through the pre-processing pipeline introduced in [42] (extraction of $time \times frequency = 32 \times 40$ inputs w/o any data augmentation).

[3] `https://www.tensorflow.org/tutorials/sequences/audio_recognition`

**Facial Expression Recognition (FER):** it is about inferring the emotional state of people from their facial expression. Quite popular in the field of vision reasoning, this task is very challenging as many face images might convey multiple emotions. The reference dataset is the *Fer2013* dataset given by the Kaggle competition [26]. It collects 32297 $48 \times 48$ gray-scale facial images split into 7 categories, i.e. "Angry", "Disgust", "Fear", "Happy", "Sad", "Surprise", "Neutral". The training set counts of 28708 examples, while the remaining 3589 are in the test set. The adopted ConvNet[4] consists of nine convolutional layers evenly spaced by three max-polling layers, and one fully-connected layer.
Each benchmark is powered by a different model whose topology is described in Table 4. Within the same table we also collected additional information: the top-1 classification accuracy achieved with the original 32-bit floating-point model (Top-1 Acc.) training w/o any optimization; the overall number of MAC instructions for one inference run using 32-bit floating-point representations (#MAC); the number of possible precision configurations, namely the number of possible operating points in the parameters space (#Op. Points).
Concerning the Top-1 accuracy reported in Table 4, the results are consistent with the state-of-the-art. They were obtained with a dedicated training and testing framework integrated into PyTorch, version 0.4.1, with the following settings: 150 training epochs using the Adam algorithm [43]; learning rate 1e-3; linear decay 0.1 every 50 epochs; batch size of 128 samples randomly picked from the training set; non-overlapping testing set and training set.

### 5.3 Results

Table 5 shows the top-1 prediction accuracy achieved with a coarse *per-net* precision scaling scheme in which all the layers share the same precision.

**Table 5:** Per-Net Precision Scaling: Top-1 Accuracy

| | 32-bit FP | *full* 16×16 Fix | *mixed* 8×16 Fix | *mixed* 16×8 Fix | *half* 8×8 Fix |
|---|---|---|---|---|---|
| **IC** | 83.04% | 83.04% | 82.27% | 73.07% | 73.93% |
| **KWS** | 80.94% | 80.92% | 79.99% | 77.26% | 76.97% |
| **FER** | 65.67% | 65.70% | 64.31% | 62.78% | 59.04% |

The table collects the results for the original 32-bit floating-point model and the four fixed-point precision options made available with the multi-precision arithmetic described in Section 3. To notice that we do not run any retraining after quantization. This allows storing a single set of weights for any desired precision. Previous works suggest a re-training stage to recover the loss due to quantization and this would imply that each precision is coupled with a different

[4] Inspired by `https://github.com/JostineHo/mememoji`

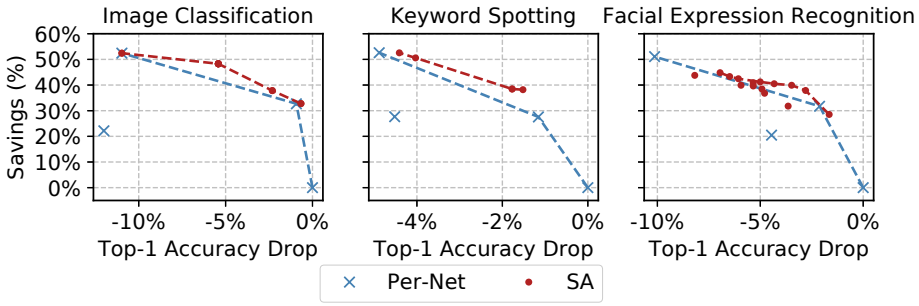


**Fig. 4:** Operating points. Accuracy drop normalized w.r.t. full precision (16 × 16).

fine-tuned model. What we propose instead is the use of a unique set of weights trained at full-precision (i.e. 16-bit for both weights and activations), then, at run-time, data are fetched and processed with the right precision. This is the key advantage of the proposed multi-precision scheme and the main enabler for adaptive ConvNets.

As reported in the table, the full-precision fixed-point ConvNets (column 16×16 Fix) keeps almost the same accuracy of the original floating-point model (the maximum relative drop is 0.02% for KWS). The results are in line with previous works and motivate the choice of 16×16 as the baseline for comparison. Concerning the mixed-precision options, 8×16 assigns 8-bit to input maps ($I$) and 16-bit to the weights ($W$); 16×8 does the opposite. The 8×16 option is by far more accurate than 16×8: minimum drop of 0.93% for **IC**; maximum drop of 2.07% for **FER**. The half-precision (column 8×8) shows larger loss: minimum drop of 4.90% for **KWS**; maximum drop 10.97% for **IC**. These numbers suggest the per-net granularity is too weak for effective deployment of adaptive ConvNets. Among the four available precision options, a very small set per se, only three are of practical use, i.e. 16×16, 8×16, 16×8. Indeed, when precision is reduced to 8 × 8 all the three benchmarks show a dramatic quality degradation. For instance, when shifted from 8 × 16 to 8 × 8, the **IC** shows a 10× drop (from 0.93% to 10.97%). This calls for a finer precision assignment policy, which is the technique proposed in this work.

A detailed analysis of the results is provided by means of a Pareto analysis, Fig. 4. The plots show the possible operating points in the energy-accuracy space achieved with a per-net precision scaling (blue ×) and the proposed per-layer precision scaling (red •). Each point comes with a different precision setting. The accuracy drop and the energy savings are normalized with respect to full-precision (rightmost × marker with 0% accuracy drop). The dotted lines connect the points at the Pareto frontier. As aforementioned, with the per-net granularity only three among four points are Pareto. Moreover, the shift from one operating point to another is very coarse with substantial accuracy drop. The advantage of the per-layer is twofold. First, the Pareto curve is more dense and hence it gives more options for a finer control; this aspect is evident in larger ConvNets

**Table 6:** Comparison between the Per-Net Precision scaling and the Per-Layer Precision Scaling with the proposed SA optimization: the collected statistics refer to the Pareto curves of the two solutions (full-precision excluded).

| | Optimization | # Op. Points | Av. Drop | Av. Savings | Av. Exec. Time |
|---|---|---|---|---|---|
| **IC** | Per-Net | 2 | -5.95 | 42.56 | - |
| | SA | 4 | -4.84 | 42.83 | 8 s |
| **KWS** | Per-Net | 2 | -3.52 | 40.09 | - |
| | SA | 4 | -2.93 | 44.95 | 13 s |
| **FER** | Per-Net | 2 | -6.13 | 41.41 | - |
| | SA | 8 | -4.60 | 39.83 | 66 min 18 s |

(e.g. **FER**). Second, the Pareto curve is dominating the per-net solutions, thus enabling larger (or comparable) average energy savings.

Table 6 reports some statistics over the subset of Pareto points, both per-net and per-layer. The column *#Op. Points* gives the number of Pareto Points; column *Av. Drop* refer to the accuracy drop averaged over the Pareto points; column *Av. Savings* does the same for the energy savings. For all the three benchmarks the energy-accuracy scaling operated with an optimal per-layer multi-precision assignment ensures optimality and usability on several context scenarios. Table 6 also shows the average execution time taken by the SA engine to draw a Pareto point, column *Av. Exec. Time*. Results are collected on a workstation powered by an Intel i7-8700K CPU and an NVIDIA GTX-1080 GPU with CUDA 9.0. As expected, time gets larger with network complexity. For the largest benchmark (**FER**) the tool consumes 66 min and 18 s.

A viable option to improve performance is to reduce the granularity at which the SA explores the parameters space. This can be achieved by constraining the number of iterations for each explored temperature $T$ (parameter *iter* in Table 2). A quantitative comparison is given in Fig. 5, whose plot shows the Pareto curves obtained with *iter*=1000 (the original value), 500 and 250 for the **FER** benchmark. The execution time reduces linearly, i.e. (66m, 18s) with *iter*=1000, (33m, 35s) with *iter*=500, (16m, 36s) with *iter*=250, while the quality of results reveal more interesting trends. Whereas it is generally true that a larger *iter* leads to better absolute numbers, the gain practically fades when considering the relative distance between the obtained curves. With *iter*=1000 the average savings across the Pareto points (39.3%) is just 5% larger than that obtained using *iter*=500 (34.6%) and *iter*=250 (34.4%); both *iter*=1000 and *iter*=500 collects the same number of Pareto points, 7 overall; only with *iter*=250 the number of Pareto points reduces from 7 to 5. This analysis suggests that for larger ConvNets there's a margin for tuning the SA to reasonable execution time w/o degrading much the quality.

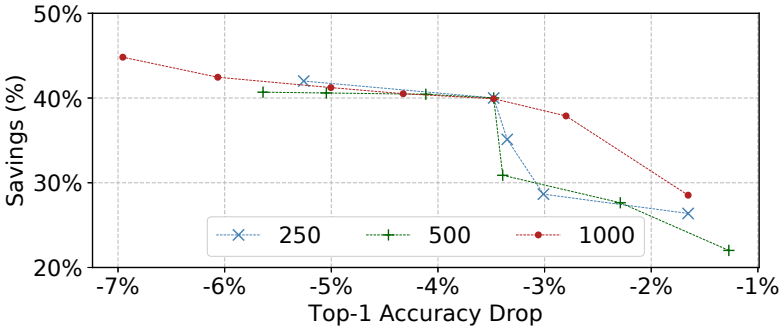


**Fig. 5:** Pareto analysis for **FER** benchmark using different number of iterations during the SA evolution: 1000, 500, 250.

## 6 Conclusions

The evolution of ConvNets has been driven by accuracy improvement. High accuracy reflected on large-scale network topologies which turned the inference into a too expensive task for low-power, energy-constrained embedded systems. ConvNets compression is therefore an urgent need for the growth of neural computing at the edge. While most of the existing techniques mainly focus on static optimizations, dynamic resource management represents a viable option to further improve energy efficiency. This chapter introduced a practical implementation of adaptive ConvNets. The proposed strategy allows ConvNets to relax their computational effort, and hence their energy consumption, leveraging the accuracy margin typical of non-critical applications. The technique is built upon a low overhead implementation of dynamic multi-precision arithmetic. The resulting ConvNets are free to move in the energy-accuracy space achieving better trade-offs. A Pareto analysis conducted on three representative applications (Image Recognition, Keyword Spotting, Facial Expression Recognition) quantified the energy savings suggesting potential improvement for the Simulated Annealing (SA) optimization engine. Future works will bring this adaptive strategy to larger ConvNets deployed on real HW implementations.

## References

1. Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.
2. A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.
3. O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh *et al.*, "Imagenet large scale visual recognition challenge," *International Journal of Computer Vision*, vol. 115, no. 3, pp. 211–252, 2015.
4. G. Hinton, L. Deng, D. Yu, G. E. Dahl, A.-r. Mohamed *et al.*, "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups," *IEEE Signal processing magazine*, vol. 29, no. 6, pp. 82–97, 2012.

5. X. Xu, Y. Ding, S. X. Hu, M. Niemier, J. Cong *et al.*, "Scaling for edge inference of deep neural networks," *Nature Electronics*, vol. 1, no. 4, p. 216, 2018.
6. L. Bottou, "Large-scale machine learning with stochastic gradient descent," in *Proceedings of COMPSTAT'2010*. Springer, 2010, pp. 177–186.
7. A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang *et al.*, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017.
8. V. Sze, Y.-H. Chen, T.-J. Yang, and J. Emer, "Efficient processing of deep neural networks: A tutorial and survey," *arXiv preprint arXiv:1703.09039*, 2017.
9. M. Grimaldi, V. Tenace, and A. Calimera, "Layer-wise compressive training for convolutional neural networks," *Future Internet*, vol. 11, no. 1, 2018. [Online]. Available: http://www.mdpi.com/1999-5903/11/1/7
10. C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed *et al.*, "Going deeper with convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1–9.
11. Y.-H. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *IEEE Journal of Solid-State Circuits*, vol. 52, no. 1, pp. 127–138, 2017.
12. M. Courbariaux, Y. Bengio, and J.-P. David, "Binaryconnect: Training deep neural networks with binary weights during propagations," in *Advances in Neural Information Processing Systems*, 2015, pp. 3123–3131.
13. E. Flamand, D. Rossi, F. Conti, I. Loi, A. Pullini *et al.*, "Gap-8: A risc-v soc for ai at the edge of the iot," in *2018 IEEE 29th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*. IEEE, 2018, pp. 1–4.
14. B. Moons and M. Verhelst, "A 0.3–2.6 tops/w precision-scalable processor for real-time large-scale convnets," in *VLSI Circuits (VLSI-Circuits), 2016 IEEE Symposium on*. IEEE, 2016, pp. 1–2.
15. J. Albericio, A. Delmás, P. Judd, S. Sharify, G. O'Leary *et al.*, "Bit-pragmatic deep neural network computing," in *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*. ACM, 2017, pp. 382–394.
16. B. Moons, B. De Brabandere, L. Van Gool, and M. Verhelst, "Energy-efficient convnets through approximate computing," in *Applications of Computer Vision (WACV), 2016 IEEE Winter Conference on*. IEEE, 2016, pp. 1–8.
17. M. Shafique, R. Hafiz, M. U. Javed, S. Abbas, L. Sekanina *et al.*, "Adaptive and energy-efficient architectures for machine learning: Challenges, opportunities, and research roadmap," in *VLSI (ISVLSI), 2017 IEEE Computer Society Annual Symposium on*. IEEE, 2017, pp. 627–632.
18. M. Alioto, V. De, and A. Marongiu, "Energy-quality scalable integrated circuits and systems: Continuing energy scaling in the twilight of moores law," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 8, no. 4, pp. 653–678, 2018.
19. V. Peluso and A. Calimera, "Weak-mac: Arithmetic relaxation for dynamic energy-accuracy scaling in convnets," in *Circuits and Systems (ISCAS), 2018 IEEE International Symposium on*. IEEE, 2018, pp. 1–5.
20. V. Peluso and A. Calimera, "Energy-driven precision scaling for fixed-point convnets," in *Very Large Scale Integration (VLSI-SoC), 2018 IFIP/IEEE International Conference on*. IEEE, 2018, pp. 1–6.
21. L. Lai and N. Suda, "Enabling deep learning at the iot edge," in *Proceedings of the International Conference on Computer-Aided Design*. ACM, 2018, p. 135.

22. N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal *et al.*, "In-datacenter performance analysis of a tensor processing unit," in *Proceedings of the 44th Annual International Symposium on Computer Architecture*, ser. ISCA '17. New York, NY, USA: ACM, 2017, pp. 1–12. [Online]. Available: http://doi.acm.org/10.1145/3079856.3080246
23. B. Moons and M. Verhelst, "An energy-efficient precision-scalable convnet processor in 40-nm cmos," *IEEE Journal of Solid-State Circuits*, vol. 52, no. 4, pp. 903–914, 2017.
24. A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," Citeseer, Tech. Rep., 2009.
25. P. Warden, "Speech commands: A dataset for limited-vocabulary speech recognition," *arXiv preprint arXiv:1804.03209*, 2018.
26. Challenges in representation learning: Facial expression recognition challenge. [Online]. Available: http://www.kaggle.com/c/challenges-in-representation-learning-facial-expression-recognition-challenge
27. R. Andri, L. Cavigelli, D. Rossi, and L. Benini, "Yodann: An architecture for ultra-low power binary-weight cnn acceleration," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2017.
28. J. Gu, Z. Wang, J. Kuen, L. Ma, A. Shahroudy *et al.*, "Recent advances in convolutional neural networks," *Pattern Recognition*, 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0031320317304120
29. T. J. Yang, Y. H. Chen, and V. Sze, "Designing energy-efficient convolutional neural networks using energy-aware pruning," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017, pp. 6071–6079.
30. F. Fleuret and D. Geman, "Coarse-to-fine face detection," *International Journal of Computer Vision*, vol. 41, no. 1, pp. 85–107, Jan 2001.
31. P. Panda, A. Sengupta, and K. Roy, "Conditional deep learning for energy-efficient and enhanced pattern recognition," in *Proceedings of the 2016 Conference on Design, Automation & Test in Europe*, ser. DATE '16. San Jose, CA, USA: EDA Consortium, 2016, pp. 475–480. [Online]. Available: http://dl.acm.org/citation.cfm?id=2971808.2971918
32. Z. Yan, H. Zhang, R. Piramuthu, V. Jagadeesh, D. DeCoste *et al.*, "Hd-cnn: hierarchical deep convolutional neural networks for large scale visual recognition," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 2740–2748.
33. K. Neshatpour, F. Behnia, H. Homayoun, and A. Sasan, "Icnn: An iterative implementation of convolutional neural networks to enable energy and computational complexity aware dynamic approximation," in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2018*. IEEE, 2018, pp. 551–556.
34. B. Moons, R. Uytterhoeven, W. Dehaene, and M. Verhelst, "14.5 envision: A 0.26-to-10tops/w subword-parallel dynamic-voltage-accuracy-frequency-scalable convolutional neural network processor in 28nm fdsoi," in *Solid-State Circuits Conference (ISSCC), 2017 IEEE International*. IEEE, 2017, pp. 246–247.
35. V. Peluso and A. Calimera, "Scalable-effort convnets for multilevel classification," in *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2018, pp. 1–8.
36. D. Lin, S. Talathi, and S. Annapureddy, "Fixed point quantization of deep convolutional networks," in *International Conference on Machine Learning*, 2016, pp. 2849–2858.

37. L. Shan, M. Zhang, L. Deng, and G. Gong, *A Dynamic Multi-precision Fixed-Point Data Quantization Strategy for Convolutional Neural Network*. Singapore: Springer Singapore, 2016, pp. 102–111.
38. S. R. Jahnke and H. Hamakawa, "Micro-controller direct memory access (dma) operation with adjustable word size transfers and address alignment/incrementing," Nov. 9 2004, uS Patent 6,816,921.
39. M. Courbariaux, Y. Bengio, and J.-P. David, "Training deep neural networks with low precision multiplications," *arXiv preprint arXiv:1412.7024*, 2014.
40. G. Desoli, N. Chawla, T. Boesch, S.-p. Singh, E. Guidetti *et al.*, "14.1 a 2.9 tops/w deep convolutional neural network soc in fd-soi 28nm for intelligent embedded systems," in *Solid-State Circuits Conference (ISSCC), 2017 IEEE International*. IEEE, 2017, pp. 238–239.
41. Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long *et al.*, "Caffe: Convolutional architecture for fast feature embedding," in *Proceedings of the 22nd ACM international conference on Multimedia*. ACM, 2014, pp. 675–678.
42. T. N. Sainath, O. Vinyals, A. Senior, and H. Sak, "Convolutional, long short-term memory, fully connected deep neural networks," in *Acoustics, Speech and Signal Processing (ICASSP), 2015 IEEE International Conference on*. IEEE, 2015, pp. 4580–4584.
43. D. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.